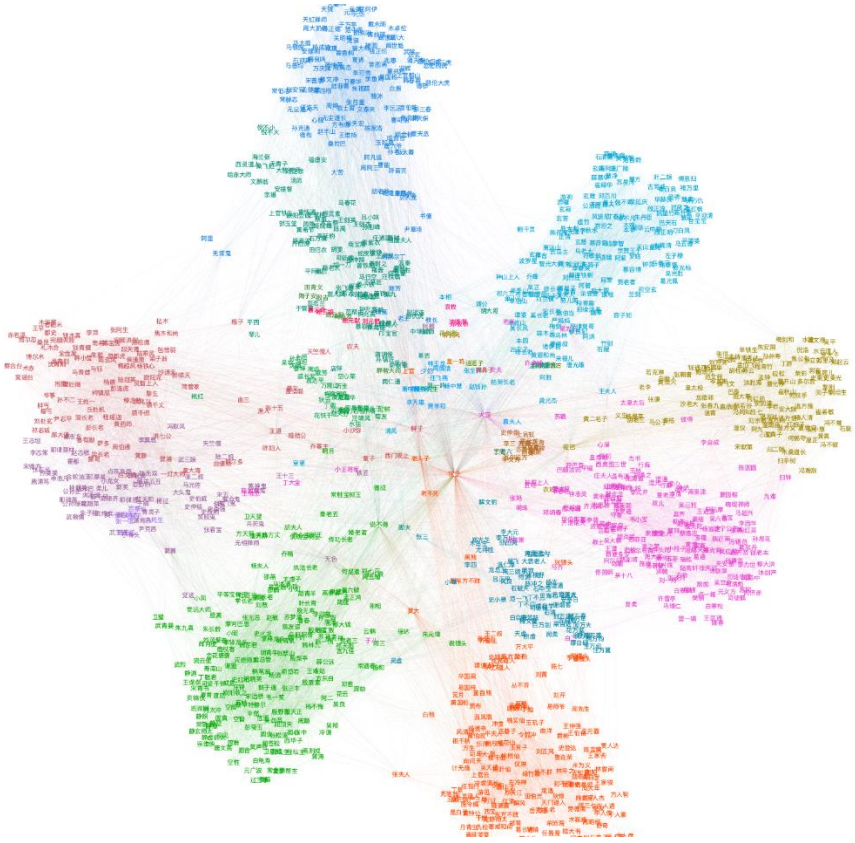
大数据综合实验：金庸的江湖

2019st08

陈 xx　16xxxxxxx
李 xx　16xxxxxxx
汤 xx　16xxxxxxx



目录

一、 简介

1、 完成功能

(1) 使用 Hadoop MapReduce 框架完成了全部 6 个子任务（包括选做部分）。

(2) 使用 Scala 语言，基于 Spark 框架完整地实现了以上 6 个子任务。

2、 运行方法

(1) Hadoop MapReduce 部分使用 Maven 管理，因此可以通过 Maven 直接编译（mvn assembly:assembly）。运行时仅需执行 Jar 文件夹下的 run_mapreduce.sh 脚本，即

./run_mapreduce.sh

更多信息请参考 CodeHadoop 文件夹和 Jar 文件夹中的 README 文件

(2) Spark 部分同样采用了 Maven 管理，因此编译、执行方法与 MapReduce 类似，具体信息请参考 CodeSpark 文件夹和 Jar 文件夹中的 README 文件。

3、 实验报告框架

本文的后续安排如下

(1) 第 2-7 部分分别介绍使用 MapReduce 框架实现每个子任务的思路与实验结果，即：

a) 数据预处理

b) 特征抽取：人物同现统计

c) 特征处理：人物关系图构建与特征归一化

d) 数据分析：基于人物关系图的 PageRank 计算

e) 数据分析：在人物关系图上的标签传播

f) 分析结果整理

(2) 第 8 部分介绍使用 Spark 实现以上 6 个子任务的具体逻辑。

(3) 第 9 部分对本次实验进行总结。

(4) 第 10 部分是附录，包括 WebUI 报告、任务分工、参考资料等内容。

二、 任务 1：数据预处理

1、 任务介绍

由于原始数据是以自然语言形式呈现的文本，而实验目的是分析人物特征与关联。因此，任务 1 首先对原始数据进行分词、过滤，只保留人名信息。

2、 实现方法

我们使用第三方中文分词库 AnsjSeg 完成本任务，版本为 5.1.6（由于该库变化频率较大，使用不同版本可能会得到不同的结果）。

3、 具体实现

(1) 词库加载：DistributedCache

首先，为了使该工具能准确识别出金庸小说中的人名，需要在分词之前将我们的原始数据中的人名列表加入到词库中。AnsjSeg 提供了一个简单的接口 DicLibrary.insert 进行词库的动态添加。但是，由于数据会被分配给多个子节点进行处理，而每个子节点的词库并不共享，因此需要在每个子节点都进行一次词库的动态添加。为此，我们把人名表存放在 DistributedCache 中：

```
job.addCacheFile(new URI(args[0] + "#" + DIC_FILE_LABEL));
```

然后，在 Mapper 的 setup 阶段，从 DistributedCache 读取出该文件

```
URI[] cacheFiles = context.getCacheFiles();
BufferedReader reader = new BufferedReader(new FileReader(DIC_FILE_LABEL));
```

最后逐个加入到词库中：

```
while ((temp = reader.readLine()) != null && temp.trim().length() > 0) {
    DicLibrary.insert(DicLibrary.DEFAULT, temp.trim(), TAG_NAME, Integer.MAX_VALUE);
}
```

此处 TAG_NAME 是用来表示单词类别的名称，便于后续判断每个单词是否人名；Integer.MAX_VALUE 是该词的权重，当同一句话有多种分词方式时，分词器会根据该权重进行选择。由于我们只需要分出人名，因此将所有人名的权重都设定为最大值。

(2) 分词、人名提取：DicAnalysis

在 map 阶段，程序每次处理一个段落的数据，由于词库已经加载完毕，而 AnsjSeg 提供了非常便利的分词接口，因此可以直接进行分词：

```
List<Term> terms = DicAnalysis.parse(value.toString()).getTerms();
```

在结果列表中，每个 term 表示一个单词，并包含类别标签。由于我们只需保留所有的人名，因此只保留标签为人名的单词。最后，把所有词语使用空格为分隔符连接起来作为输出。

(3) 多文件输出：MultipleOutputs

值得一提的是，由于任务 1 要求每个输入文件对应一个输出文件，因此使用 MultipleOutputs 将结果输出到多个文件中。

首先，保留文件的来源信息，map 阶段会把该段落所在的文件名作为 key 输出（value 值是该段落分词、筛选后生成的只包含人名的字符串）；其次，在 reduce 阶段，首先在 setup 中新建一个 MultipleOutputs 对象

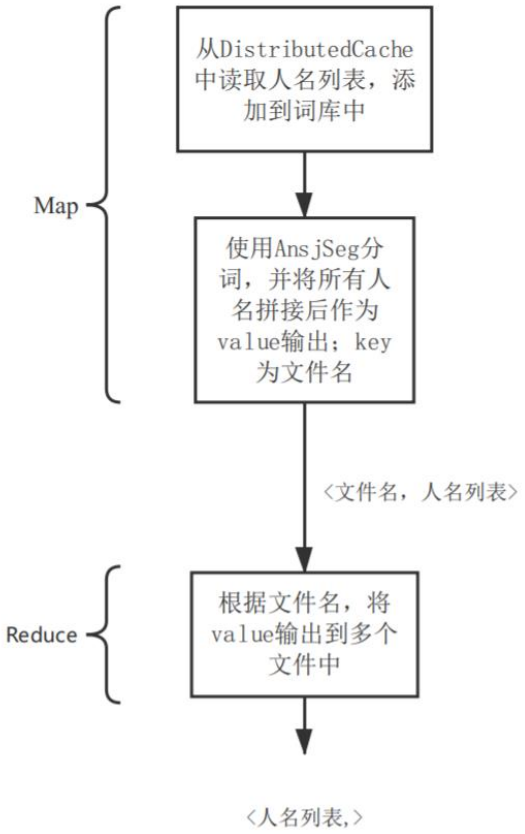
```
outputs = new MultipleOutputs<>(context);
```

然后在 reduce 函数中根据 key 值（即数据源文件名）重定位到对应的输出

文件中，并传递给 MultipleOutputs 对象，实现一个输入文件对应一个输出文件的功能

```
outputs.write(value, new Text(), key.toString().substring(2,4));
```

4、 流程图



5、 输出结果

```
一灯大师 小龙女 杨过 小龙女
一灯大师 无色 郭襄 无色 郭襄 无色 郭襄 周伯通
一灯大师 朱子柳 杨过 朱子柳 一灯大师 点苍渔隐 朱子柳 朱子柳 杨过
一灯大师 朱子柳 觉远
一灯大师 杨过
一灯大师 杨过 郭襄
一灯大师 裘千尺 黄蓉 武三通
一灯大师 郭襄 杨过 一灯大师 郭襄
一灯大师 郭靖
一灯大师 郭靖 一灯大师 黄蓉 郭靖 郭靖 郭靖 一灯大师 黄蓉
一灯大师 郭靖 周伯通 黄蓉 黄药师 郭靖
```

三、 任务 2：特征抽取：人物同现统计

1、 任务介绍

本任务需要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，一次同现关系定义为两个人在原文的同一段落中出现。而两个人物的同现统计次数越多，说明这两个角色之间的关联性越大。

2、 算法思想

(1) Mapper 阶段：

任务一中已经完成了段落的文本分词、人名提取，文本转变成了如下格式的字符串：

[人名 1]　[人名 2] ... [人名 n]

需要先对在段落中的角色名按照"\t"划分得到字符串数组，进行去重，得到角色名字的集合（m≤n）：

（[人名 1], [人名 2], ..., [人名 m]）

再统计角色名集合中的每个名字与其他角色的同现次数，每个段落中的每个角色对的同现次数均计为 1，输出格式如下按照键值对：

<[人名 1],[人名 2]>，1

(2) Reducer 阶段：

将相同角色对的出现次数累加，输出至文件的格式按照：

<[人名 1],[人名 2]>，同现次数

3、 参数类型

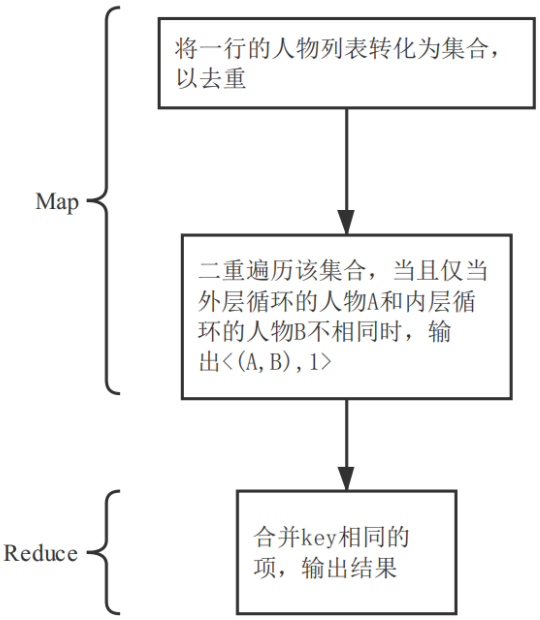
	输入 key	输入 value	输出 key	输出 value
Mapper	Object	Text	Text	IntWritable
Reducer	Text	IntWritable	Text	IntWritable

4、 算法伪代码流程

```
class CoexistenceStatisticsMapper
    method map(Text text, Context context)
        roles ←段落分词后的角色名数组
        roleSet ←去重后的角色名集合
        // 遍历实现该段落的同现统计
        for roleOne in roleSet:
            for roleTwo in roleSet:
                // 角色名和其本身不需要统计同现次数
                if roleTwo not equal to rolwOne:
                    // 按照(<角色1，角色2>，1)的键值对形式输出
                    emit(<roleOne, roleTwo>, 1)
```

```
class CoexistenceStatisticsReducer
    method reduce(tuple <roleOne, roleTwo>, IntWritable[] values)
        //初始化同现次数
        sum ← 0
        for value in values:
            // 累加同现次数
            sum ← sum+value
        // 按照(<角色1，角色2>，同现次数)
        emit(<roleOne, roleTwo>, sum)
```

5、 流程图



6、 输出结果

<一灯大师,上官>	1
<一灯大师,丘处机>	5
<一灯大师,乔寨主>	1
<一灯大师,农夫>	16
<一灯大师,华筝>	1
<一灯大师,卫璧>	2
<一灯大师,吕文德>	1
<一灯大师,周伯通>	26
<一灯大师,哑巴>	1
<一灯大师,哑梢公>	1
<一灯大师,大汉>	1
<一灯大师,天竺僧>	1
<一灯大师,天竺僧人>	3
<一灯大师,完颜萍>	2

四、 任务 3：特征处理：任务关系图构建与特征归一化

1、 任务介绍

完成上述任务后，我们得到了金庸所有武侠小说中，人物之间的同现次数。该任务利用人物之间的共现关系，生成人物之间的关系图，其目的是使用邻接表的形式表示人物关系图，并对共现次数进行归一化处理，从而方便后面 PageRank 的计算及后续分析。其中对于用邻接表表示的人物之间的关系图，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取决于人物之间的

共现关系，而归一化处理类似于计算该名字的同现概率。

2、 算法基本思想

由于输入数据是姓名对和同现次数，且在该任务过程中需要利用同现次数的值进行计算，因此姓名和同现信息可以用 Text 表示，而同现次数类型应为 IntWritable。因此我们可以确定 Mapper、Reducer 的输入输出类型如下表：

	输入 key	输入 value	输出 key	输出 value
Mapper	Object	Text	Text	IntWritable
Reducer	Text	IntWritable	Text	Text

对于 Mapper，其输入 value 为文本文件中的某一行数据，其格式形为

<[人名 1],[人名 2]>　　[同现次数]

举例：

<荻云，戚芳>　1

所以读入某行文本数据之后，首先要做的就是将人物之间的互动关系和同现次数分离，将人物之间的互动关系存储在 key 中，而同现次数则存储在输出格式为 IntWritable 的 value 中，从而便于利用同现次数的值进行数值上的归一化。

对于 Reducer，其输入是 Mapper 的输出结果，即<[人名 1],[人名 2]> [同现次数]简直对，而 Reducer 的输出是整个任务的最终输出。按照任务要求，输出包括人物关系和共现概率，因此输出键值对分别是

[人名 1] [[人名 2],[共现概率]|...]，

举例：

荻云　[戚芳,0.33333|戚长发,0.33333|卜垣,0.333333]

其中 value 是与人名 1 同现的人名及共现概率列表。为此，需要让具有同一个姓名的共现关系输出到同一个 Reducer 节点中，需要一个定制的 Partition。其次，为了计算出共现概率，需要得到与该姓名有关的所有共现次数之和，所以与一个姓名有关的的结果输出应在所有与该姓名有关的同现关系都经历过同一个 Reducer 处理之后，并利用该过程中的累计信息，计算并输出结果。

3、 代码设计

(1) Mapper

由于输出 value 格式为 Text，首先需要将 value 的 Text 格式转化为 String 格式，从而便于进一步分离处理，所以这里用到了 toString()函数得到 String 类型的字符串，再用 trim()函数消除字符串头尾部分的空白字符，从而避免不必要的麻烦。

得到便于处理的字符串后，则需要将同现关系和同现次数分离。由于在

MapReducer 中，存储 key 和 value 到文本文件中时默认按制表符（'\t'）分隔，所以利用 split("\t")函数将人物之间的互动关系和同现次数分离，将分离的结果存储在 String 类型数组 items 中。在输出结果之前，对于 items 个数大于 2 的异常情况进行检测，及时抛出 RunTimeException()异常。

在将结果键值对输出到 context 时，由于 items 的类型为 String，而输出 key 的类型为 Text、输出 value 的类型为 IntWritable，所以需进行相应的类型转换，即输出 key 为 new Text（items[0]），输出 value 为 new IntWritable(Integer.valueOf(items[1]))。

```
String[] items = value.toString().trim().split("\t");
if (items.length != 2) {
    throw new RuntimeException();
}
context.write(new Text(items[0]), new IntWritable(Integer.valueOf(items[1])));
```

(2) Partitioner

由于该人物的输出结果是由邻接表表示的邻接关系，即每一行由该顶点 X 及与其相邻的顶点集 V 组成，因此输出到 Reducer 的键值对应按照该行表示的顶点 X 分类。因为 Partitoner 按照 key 排序，为保证同一个名字下的一组顶点被分到同一个 Reducer，这里需要去重写 Partitioner 中的 getPartitioner()方法，将组合键<name1,name2>临时拆开，让 Partitioner 按照 name1 排序。

与 Mapper 中类似，需要先将 Text 格式的 key 值转换为 String 便于处理，再根据","分割同现关系，提取出同现关系中的第一个名字。同样地，在调用其超类的 getPartitioner()方法前，对于 items 个数和 items[0]第一个字符是否为'<'进行一场情况检测，及时抛出 RunTimeException()异常。

最后将提取出的同现关系中的第一个名字作为 key，也即 getPartitioner()的分类依据。

```
String[] items = key.toString().trim().split(",");
if (items.length != 2 || !items[0].startsWith("<")) {
    throw new RuntimeException();
}
return super.getPartition(new Text(items[0].substring(1)), value, numReduceTasks);
```

(3) Reducer

在该任务中继承的 Reducer 类中，重写了 reduce、cleanup 方法，编写了 writeCurrentName 方法，该类的这些方法共同完成了人物关系图整理和归一化计算的功能。

该类还有三个私有变量，分别是 String 类型的 currentName、int 类型的 currentCount、Map 类型的 postings，其中 currentName 赋初值为 null，用来存储当前正在记录的邻接表头姓名 Name1；currentCount 赋值为 0，用来记录与 Name1 同现的姓名集个数，方便归一化；postings 为<String, Integer>类型的 map，用来记录与 Name1 同现的姓名集，其中 String 用来存储姓名字符串，

Integer 用来存储同现次数。

```
private String currentName = null;
private int currentCount = 0;


private Map<String, Integer> postings = new HashMap<>();
```

a) reduce 方法

首先，对输入 key 的姓名进行检测。先将 Text 类型的 key 转化为 String 类型的字符串，若该字符串没有以“<”开头、以“>“结尾，抛出运行时刻异常。在 String 类型的数组 items 中存储一个共现关系涉及到的两个名字，对于 items 元素大于 2 的情况，抛出运行时刻异常。

若当前名字 currentName 为空（null），则清空 postings，将 currentName 赋值为 items[0]，即键中的首个名字，currentCount 为 0。

若输入 key 与 currentName 相同，则输出与当前名字（currentName）有关的人物关系信息及归一化后的同现概率，然后清空 postings，将 currentName 赋值为 items[0]，currentCount 为 0。

在完成上述分支后，对于每一个输入的键值对，遍历值集合（values），每一个值（value）是其键代表的同现关系的次数，累计他们的同现次数 sum，再把 sum 累计到 currentCount 中，便于下一个键值对累计计算与 currentName 的同现次数。最后将该同现关系的相关信息（包括姓名和同现次数）存储到 postings 中，即在 postings 中存放新的键值对<items[1], sum>。该段伪代码如下：

```
sum ← 0
for value in values:
    sum ← value.get()
currentCount ← currentCount+sum
postins.put(items[1],sum)
```

b) cleanup 方法

当程序运行到 cleanup()方法，说明所有键值对都以在 reduce 中处理过。此时还剩下最后一个名字（即当前 currentName）的同现信息没有输出，因此在该方法中调用了 writeCurrentName 方法，将未输出的信息输出。

c) writeCurrentName 方法

在 writeCurrentName 方法中，StringBuilder 类型的 builder 用来存储输出信息。

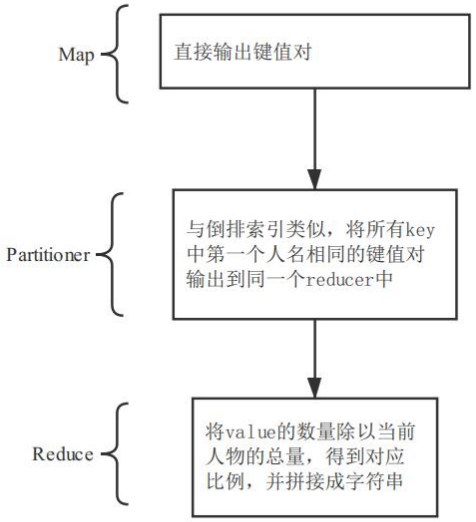
若 currentName 为空，则立即返回。

遍历 postings 中的每一组键值对，依次输出其姓名和同现概率，其中姓名由 posting 键值对中的 key 得到，同现概率由 posting 键值对中的 value 与同现总数（total，即存储了 currentCount 数值的 double 类型）相除获得。在输出规范

部分，为使输出遵循[[人名 2],[共现概率]|...]的格式，在循环之前输出"["，在每次循环结束后输出分隔符“|”。由于这样会使得结束时多一个分隔符，因此在循环全部结束后，若 builder 的长度大于 1，则删除最后一个字符，也即多于的分隔符“|”。

在得到全部的输出信息后，即可输出 Reducer 模块的键值对，其中 key 中存储了 currentName，value 存储了与 currentName 相关联的同现信息，即 builder。

4、 流程图



5、 输出结果

一灯大师　　[鲁有脚, 0.009569377990430622|哑梢公, 0.0023923444976076554|杨康, 0.007177033492822967|耶律燕, 0.007177033492822967|尹克西, 0.0023923444976076554|渔人, 0.02631578947368421|裘千尺, 0.014354066985645933|耶律齐, 0.011961722488038277|丘处机, 0.011961722488038277|汉子, 0.004784688995215311|欧阳克, 0.007177033492822967|哑巴, 0.0023923444976076554|马钰, 0.004784688995215311|……]（略）
丁不四　　[吕正平, 0.04225352112676056|封万里, 0.01056338028169014|范一飞, 0.045774647887323945|汉子, 0.02112676056338028|小翠, 0.03169014084507042|龙岛主, 0.017605633802816902|梅文馨, 0.02464788732394366|李四, 0.0035211267605633804|王万仞, 0.0035211267605633804|石清, 0.02464788732394366|丁不三, 0.09154929577464789|石中玉, 0.01056338028169014|白万剑, 0.09507042253521127|……]（略）

五、 任务 4： 数据分析： 基于人物关系图的 PageRank 计算

1、 任务介绍

对任务 3 得到的人物关系图进行基于 PageRank 值的数据分析。根据得到的 PageRank 值，我们就可以定量地得到金庸武侠江湖中的“主角”。

2、 算法思想

(1) PageRank 算法

PageRank 算法认为：被许多优质网页所链接的网页，多半也是优质网页。那么，在本次实验当中，如果一个角色想要拥有较高的 PR 值，该角色需要有很多其他角色与其同现或者有高质量的角色与其同现。

因为可能存在有些角色没有与其他角色同现而导致的 Rank Leak 和 Rank Sink 问题，我们采取随机浏览模型，将计算 PR 值的表达式修改为 PR = (1-0.85) + 0.85*PR，表示按照 0.85 的概率继续选择与该角色同现，同时按照 0.15 的概率选择另一个角色同现。

a) GraphBuilder / StepOne

需要建立各个角色之间的链接（同现）关系。

Mapper 阶段：逐行分析原始数据，其中角色名作为 key，PR 初始值（PR_INIT=1.0）和角色的归一化同现统计作为 value，用符号“#”分割，以字符串输出。输出格式为：

<role, pr#neighbor1,weight|neighbor2,weight|...>

Reducer 阶段：直接输出 Mapper 阶段处理好的数据即可，不需要再进行额外的处理。

b) PageRankIter / StepTwo

迭代计算角色的 PR 值，直到迭代至预定次数（25 次）。

Mapper 阶段：对 GraphBuilder 阶段的输出产生两种<key, value>对，一种是原始的链出信息，维护图的结构。key 为角色名，value 为同现角色名及对应权重，即不包含 PR 值的键值对，按照格式：

<role, neighbor1,weight|neighbor2,weight|...>

另一种是对每一个同现角色迭代计算其 PR 值，其中 newPR=weight*oldPR，输出的 key 为同现角色名，value 为新 PR 值，按照格式：

<neighbor, #pr>

这两种键值对输出可以利用符号“#”区分。

Reducer 阶段：根据不同的键值对做不同的处理：对于第一种键值对，直接取出其 value 值并记录；对于第二种键值对，需要累加 PR 值，作为该角色的 PR 总和，最后按照 newPr=1.0-DAMPING+DAMPING*prSum 计算新的 PR 值。输出时 key 为角色名，value 为 PR 值和同现及对应权重，用“#”作为连接符，输出格式为：

<role, pr#neighbor1,weight|neighbor2,weight|...>

输出格式与 GraphBuilder.Reducer 的输出格式相同，便于迭代计算 PR 值的实现。

c) RankViewer / StepThree

将最终人物的 PR 值结果排序输出。

Mapper 阶段：对 PageRankIter 最后一次迭代得到的 value 字符串值按照"\t"划分，得到角色的 PR 值，输出 key 为角色名，value 为 PR 值的键值对，输出

格式为：

<role, PR>

Reducer 阶段：不需要做额外的操作，只要将 Mapper 阶段的结果输出即可。

3、 异常检查

在每一个阶段都检查了输入字符串的格式是否正确，如果不正确则会抛出 RuntimeException()，提高了程序的鲁棒性，也便于出错时的定位查找。

4、 参数类型

	输入 key	输入 value	输出 key	输出 value
StepOneMapper	Object	Text	Text	Text
StepOneReducer	Text	Text	Text	Text
StepTwoMapper	Object	Text	Text	Text
StepTwoReducer	Text	Text	Text	Text
StepThreeMapper	Object	Text	Text	DoubleWritable
StepThreeReducer	Text	DoubleWritable	Text	DoubleWritable

5、 算法伪代码流程

(1) GraphBuilder / StepOne

```
class StepOneMapper
    method map(Text text, Context context)
        // items[0]: 角色名，items[1]: 同现角色及其权重
        items←text.split("\t")
        // 设置PR初始值为1.0
        PR_init←1.0
        value←PR_init+"#"+items[1]
        emit(items[0], value)
```

```
class StepOneReducer
    method reducer(Text key, DoubleWritable[] values)
        // 不对Mapper得到的值做额外处理，直接输出
        for value in values:
            emit(key, value);
```

(2) PageRankIter / StepTwo

```
class StepTwoMapper
    method mapper(Text text, Context context)
    // items[0]: 角色名，items[1]: pr值和同现角色及其权重
    items ← text.split("\t")
    role ← items[0]
    // prAndNeighbors [0]: pr值
    // prAndNeighbors [1]: 同现角色及其权重
    prAndNeighbors ← items[1].split("#")
    pr, neighbors  ← prAndNeighbors[0], prAndNeighbors[1]
    //保留原始链出信息
    emit(role, neighbors)

    for neighbor in neighbors:
        neighborAndWeight ← neighbor.split("|")
        emit(neighborAndWeight[0], pr*neighborAndWeight [1])
```

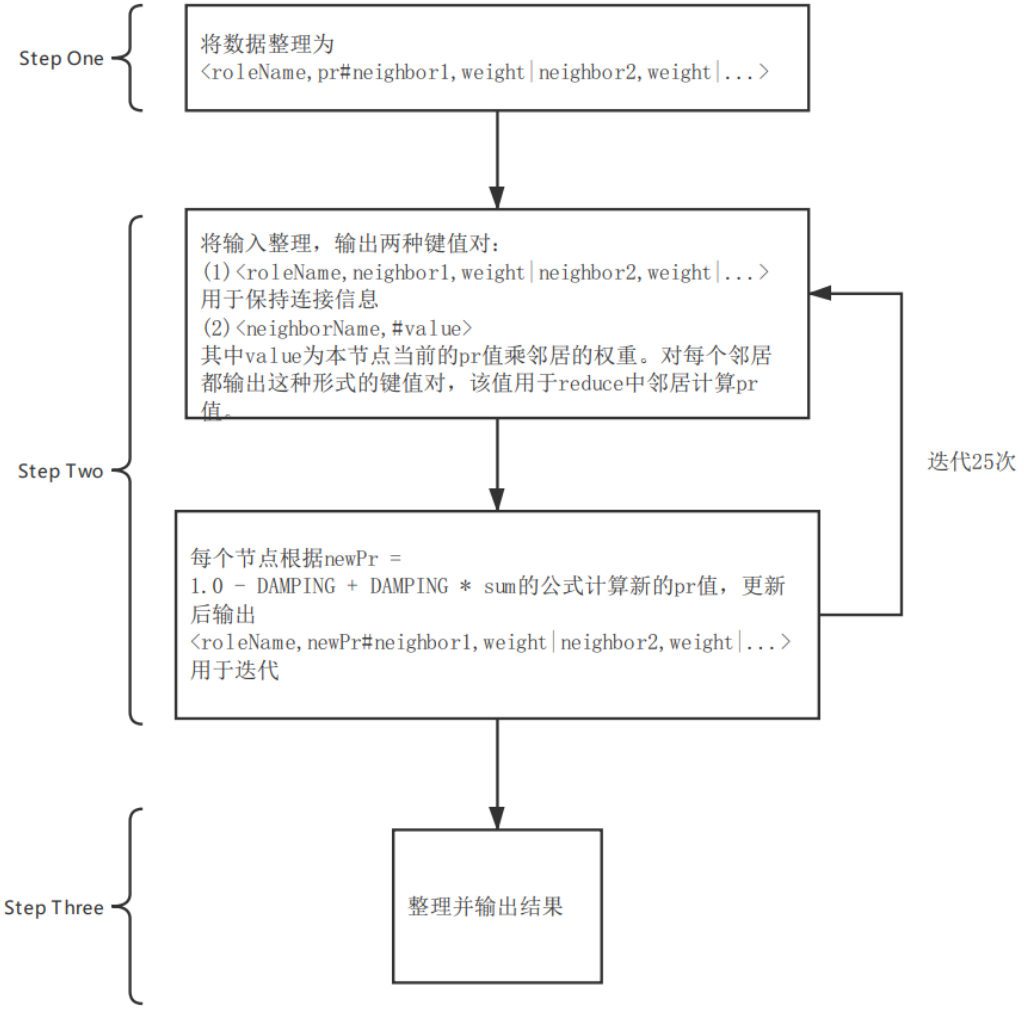
```
class StepTwoReducer
    method reducer(Text key, Text values)
    sum ← 0
    for value in values:
        if value starts with "#":
            value ← value-"#"
            sum ← sum + value.toDouble
        elif:
            neighbors ← value
    // 设置松弛系数为0.85
    DAMPING ← 0.85
    newPr ← 1.0-DAMPING+DAMPING*sum
    emit(key, newPr+"#"neighbors)
```

(3) RankViewer / StepThree

```
class StepThreeMapper
    method map(Text text, Context context)
    // items[0]: 角色名，items[1]: pr值和同现角色及其权重
    items ← text.split("\t")
    role ← items[0]
    pr ← items[1].split("#")[0]
    emit(role, pr)
```

```
class StepThreeReducer
    method reducer(Text key, DoubleWritable values)
    // 不做额外操作，直接输出Mapper处理得到的结果
    for value in values:
        emit(key, value);
```

6、 流程图



7、 输出结果

一灯大师	1.37946
丁不三	1.324171
丁不四	2.275728
丁典	1.657455
丁勉	0.664319
丁同	0.22403
丁坚	0.460754
丁大全	0.487439
丁敏君	1.090182
丁春秋	3.22594

六、 任务 5：在人物关系图上的标签传播

1、 任务描述

使用标签传播（Label Propagation）算法，对任务关系图进行分析，为每个点打标签从而形成聚类。

2、 算法简介

标签传播算法的基本思想非常简单，就是“每个节点的所有邻居的出现频率最高的标签，即为该节点的标签”。由于相邻节点的标签计算是相互依赖的，显然无法直接计算出结果，必须迭代地计算每个节点的标签，直到稳定状态。

根据标签传播算法的原始论文[1]，该算法的基本流程如下:

(1) 初始为每个节点设定一个唯一的标签。此处采用和任务 1 类似的方法，将人名表放在 DistributedCache 中，然后从 0 开始为每个人物编号，作为其初始标签。

(2) 对每个节点，检测它的所有邻居的标签，把出现频率最高的标签作为当前节点的标签。当多个标签同时拥有最大值时，从中随机选取一个。由于具体实现时采用了同步计算的方法，为了防止出现“振荡”现象，在多个标签同时拥有最大值时，会优先选择节点在上一轮中的标签。只有当节点上一轮的标签不在拥有最大频率的标签集合中时，再从中随机挑选。

(3) 重复第二步进行迭代处理，每一轮的输入为前一轮的输出。在实际计算中重复了 40 轮。

(4) 将最后一轮的输出中每个节点的标签作为最终结果。

3、 算法细节

(1) 考虑到 MapReduce 计算模型本身的特点，本次实验采用了同步的计算方法。即：第 k 轮中某个节点的计算，只需要用到第 k-1 轮的计算结果，而不使用第 k 轮中已经计算出来的结果。

(2) 使用加权的方法计算标签。由于节点和它的每个邻居的共现次数不一定相同，显然共现次数多的节点对有更强的关联，更可能出现在同一个聚类中。所以我们采用了加权的方式计算节点的标签，即：当节点有一个邻居的标签为 N 时，标签 N 的计数不是增加 1，而是增加该邻居在节点的所有邻居中所占的比例（即实验三整理的结果）。在遍历完节点的所有邻居后，将计数（即标记为该标签的邻居比例）最高的那个标签作为当前节点的标签。

4、 实现细节

该算法在具体实现时，分为三个步骤进行。

(1) 步骤 1：预处理

该步骤主要是为每个节点先分配一个唯一的标签，即：从 DistributedCache 中读取出人名和标签的映射表，输出标签信息。此处的具体实现与任务 1 类似，不再赘述。

在 Map 阶段，一共会输出两种类型的<key,value>对，即：

```
<roleName,#roleTag#neighbor1,weight|neighbor2,weight|...>
```

和

```
<roleName, neighborName#neighborTag>
```

其中，第一种键值对保存了节点的标签信息、连接信息；第二种键值对保存了节点的所有邻居的标签信息。此处通过在第一类键值对的 value 开头插入一个'#'符号来区分两种类型。由于使用的是全局的“人名->标签”映射表，所以在处理当前节点时可以直接获取其邻居的标签信息，从而输出。

Reduce 阶段不再做其他处理，直接输出键值对即可。

(2) 步骤 2：迭代计算

该步骤是标签传播算法的核心步骤，以迭代的方式不断更新各个节点的值。该步骤中 Map 阶段不做特殊处理，直接读取键值对即可。

在 Reduce 阶段，由于输入中包含两种类型的输出（即步骤 1 中提到的两种类型），因此具备计算当前节点新标签的所有条件。我们采用了两个 HashMap，分别是“邻居名->邻居标签”和“邻居名->邻居权重”，通过对所有 value 的一次变量填充这两个 HashMap。基于这些信息，直接按照算法说明中的过程计算出新节点的标签。计算完毕后，按以下方法输出：

a) 将第 1 种键值对

```
<roleName,#roleTag#neighbor1,weight|neighbor2,weight|...>
```

中的标签（roleId）更新后，再次输出该键值对，以维持图结构和当前标签信息。

b) 对于当前节点的每个邻居 neighbor，输出

```
<roleName, neighborName#neighborTag>
```

即第 2 种键值对.

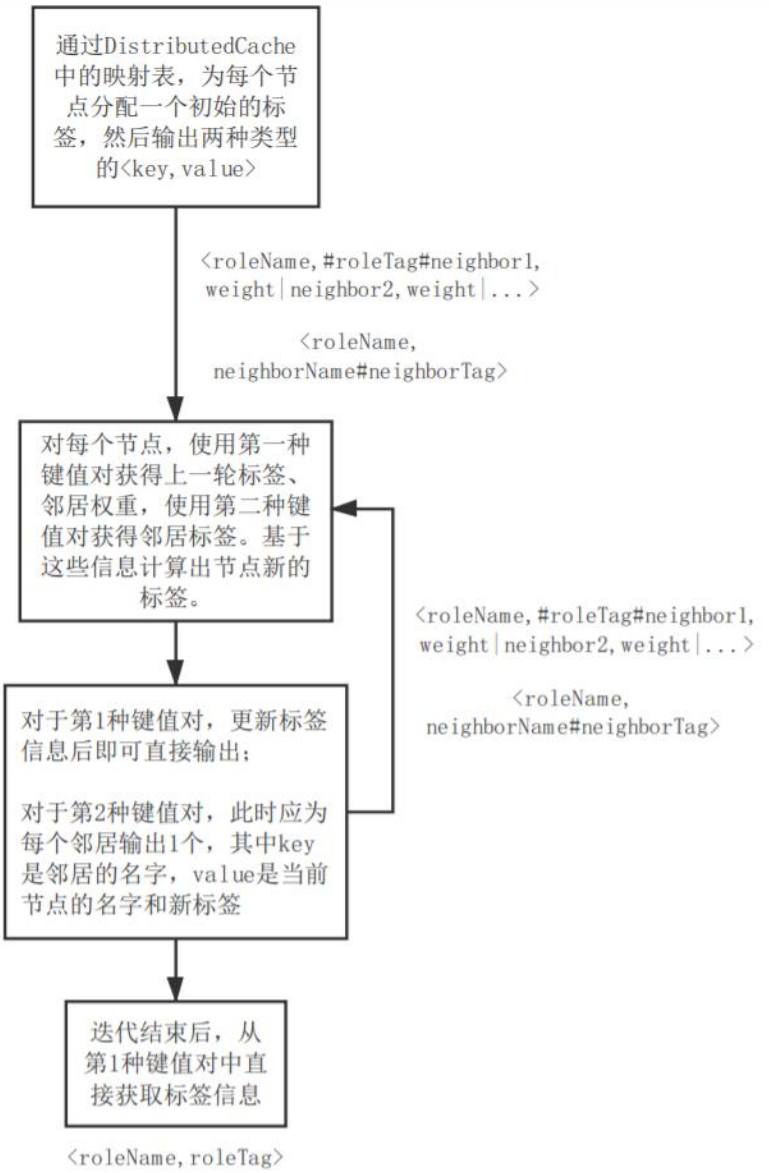
值得一提的是，此处输出的第 2 种标签，在下一轮中并不用于当前节点的计算，而是用于其邻居节点的计算，即“通知当前节点的邻居，当前节点的标签已经更新了”，以用于邻居下一轮的计算。同理，下一轮中当前节点的计算数据，来源于本轮中各个邻居的输出。

可以看出，步骤 2 的输入、输出具有相同的形式，所以可以进行迭代计算。

(3) 步骤 3：结果抽取

在迭代计算完毕后，输出结果同样拥有和步骤 1 相同的两种键值对，其中第 1 种存储了节点的标签信息。所以，本轮只需识别出这种键值对，提取出标签信息并输出即可。

5、 算法流程图



6、 输出结果

一灯大师	303
丁不三	819
丁不四	819
丁典	182
丁勉	543
丁同	878
丁坚	543
丁大全	1003
丁敏君	1282
丁春秋	214

七、 任务 6：分析结果整理

完成上述任务后，为了更容易地从中得到一些有趣的结论，我们需要对上述分析的结果进行整理。

对于任务四（PageRank）的输出，由于它并非全局排序的，所以这里通过一个排序 MapReduce 程序完成排序工作，它对人物的 PageRank 值进行排序，从而可以很容易地发现 PageRank 值最高的几个任务。

对于任务五（标签传播）的输出，按照标签进行整理，即将属于同一个标签的人物输出到一起，从而便于查看标签传播的结果。

1、 对 PageRank 全局排序

(1) 算法基本思想

在 MapReduce 中， 通常一个 reduce 节点会按照一定排序规则处理键值对，而 reduce 节点间的数据则未必是排序后的结果。因此为使得 PageRank 按照全局由大到小排序，每个 reduce 节点的数据范围应是按照一定顺序安排过的，这样所有 reduce 节点的输出结合到一起就是全局排序的结果。对于 reduce 节点的数据范围安排，由于 key 的分布未知，若是人为则有可能造成各 reduce 节点数据规模不一、甚至某个 reduce 节点过载的情况，因此这里我们使用到了 TotalOrderPartition。

TotalOrderPartition 是一个提供全序划分的 Partitioner，从 Hadoop v0.19.0 开始正式发布在库类中。它通过采样获取数据的分布，从而构建高效的划分模型。具体来讲，TotalOrderPartition 依赖于 partition file 来分配 key 值在各 reduce 节点上的范围，而 partition file 是对输入文件进行取样创建而成。

由于程序应是对 PageRank 进行排序，而 PageRank 对应任务四输出的 value 部分，因此该 MapReduce 程序可分为两部分完成，第一部分将任务四输出的键值对互换，即

[人物] [对应 pagerank 值]

转换为

[pagerank 值] [对应人物]。

在第二部分，对第一部分的输出进行抽样，得到 partition file，并利用 TotalOrderPartition 对其进行高效划分，并得到最终的 PageRank 排序结果。

综上，我们可以得到该任务中的数据类型：

	输入 key	输入 value	输出 key	输入 value
StepOneMapper	Object	Text	DoubleWritable	Text
StepOneReducer	DoubleWritable	Text	DoubleWritable	Text
StepTwoMapper	Text	Text	Text	Text
StepTwoReducer	Text	Text	Text	DoubleWritable

(2) 第一部分

第一部分的代码实现主要分为 Mapper 和 Reducer。在 Mapper 部分，读取传入 Map 节点的一行数据，将其转化为字符串并分割，得到人物姓名和 PageRank，并发送新的键值对<Pagerank,name>到下一个环节。在 Reducer 部分，遍历 values，即遍历同一个 pagerank 值的姓名集，并输出<PageRank,Name>到结果。Reduce 部分的伪代码如下：

```
reduce(DoubleWritable key, Iterable<Text> values, Context context)
    for value in values：
        Emit(key, value)
```

(3) 第二部分

第二部分的代码可分为采样、Mapper、Shuffle and Sort、Reducer 几个部分。在采样部分，采取的是随机采样 RandomSampler，其中 RandomSampler 的第一个参数表示 key 会被选中的概率，第二个参数是采样个数，第三个参数是最大分区数。如下图，这里我们的抽样概率为 0.01，采样个数为 1000，最大分区数为 100。

```
InputSampler.Sampler<Text, Text> sampler = new InputSampler.RandomSampler<>(0.01, 1000, 100);
```

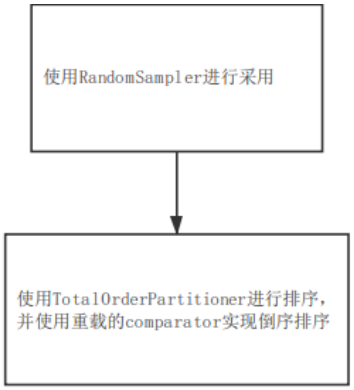
在 Mapper 部分，只需要重新读取 key 和 value 的内容，并将它们传递给下一个步骤。

在 Shuffle and Sort 部分，编写了一个继承了 WritableComparator 的类 MyComparator，重写了里面的 compare 方法。在新的 compare 方法中，将 WritableComparable 类型的数据转化为 String 类型，并从中获取其 double 值，对两个数的 double 值进行比较。

在 Reducer 部分，由于此时传入个 reduce 节点的数据已经是降序排列的，因此只要按序输出即可，所以这里直接输出键值对<Name，PageRank>。由于在前述部分将姓名和 pagerank 值进行了键值互换，因此这里需要转换回来，所以对于 values 中的每一个 value，将其作为 key 输出，而存储在 key 中的 pagerank 值作为 value 输出。伪代码如下：

```
reduce(Text key, Iterable<Text> values, Context context)
    for value in values:
        Emit(value, DoubleWritable类型的key)
```

(4) 流程图



(5) 输出结果

韦小宝	34.470746
令狐冲	20.439687
张无忌	19.880607
郭靖	16.112
杨过	14.157108
袁承志	13.816323
黄蓉	13.802153
段誉	13.713533
胡斐	13.296895
汉子	13.096465
陈家洛	9.345399
吴三桂	7.853268
石破天	7.437129

2、 对标签传播进行整理

(1) 算法基本思想

该人物的输入为任务五的输出，key 对应人物姓名，value 对应人物标签，即

[人物姓名] [人物标签]

由于在 MapReduce 中，同一个 key 的键值对会被分配到同一个 reducer 节点中，因此可以利用这一特性编写相应程序。在该任务中，为使得同一个标签的人物输出到一起，在 Map 中键值互换，从而让具有同一个 key（这里是人物标签）的键值对分配到相同 reduce 节点，再将键值互换回来，依次输出即是任务的目标输出。

综上所述，由于在该任务中的键值都是姓名或标签，不涉及到数值计算，因此数据类型皆为 Text，详情如下表所示。

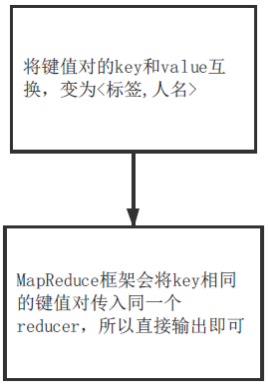
	输入 key	输入 value	输出 key	输出 value
Mapper	Object	Text	Text	Text
Reducer	Text	Text	Text	Text

(2) 代码实现

在 Mapper 部分，从 value 中读取字符串并提取出任务所需要的姓名的标签，并存储在 String 类型的数组 items 中，最后将标签作为 key，姓名作为 value 的键值对输出。在输出键值对之前，对 items 数组元素个数进行检查，及时抛出运行时刻异常。

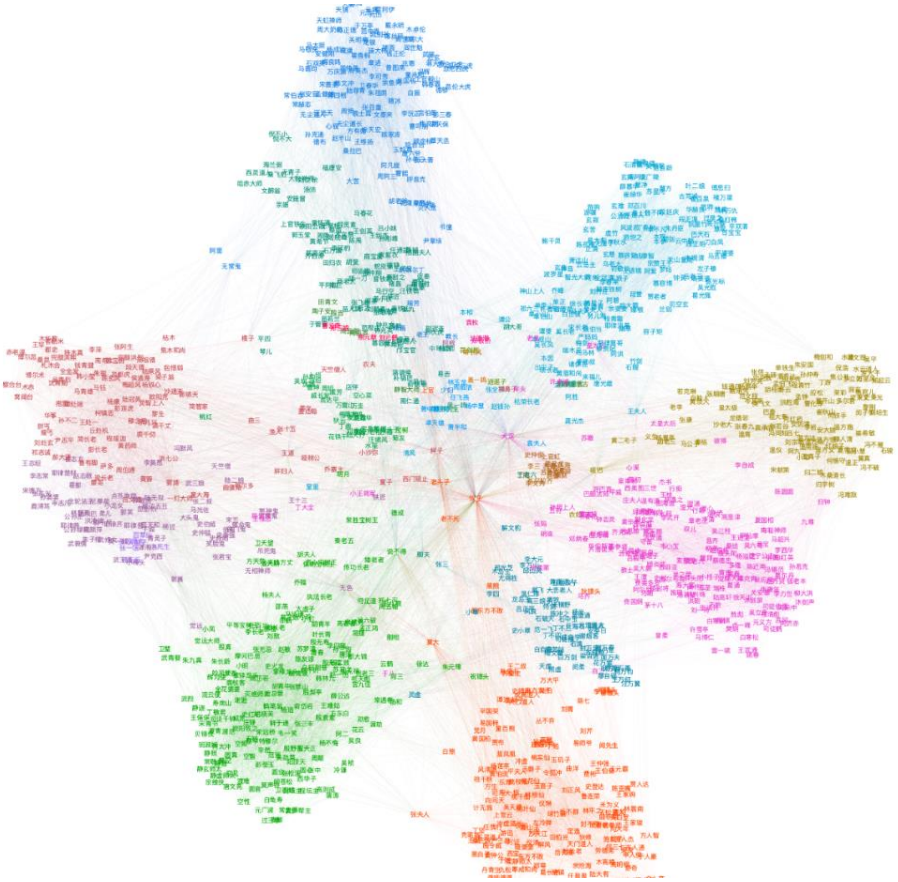
在 Reducer 部分，由于进入 reduce 节点前进行了 Shuffle and Sort 部分，在该阶段中将具有相同 key 值的键值对分配到了同一个 reduce 节点，因此这里只需要将键值对互换并输出，即将传入 reduce 节点的<人物标签，人物姓名>转换为<人物姓名，人物标签>输出。

(3) 流程图



(4) 输出结果

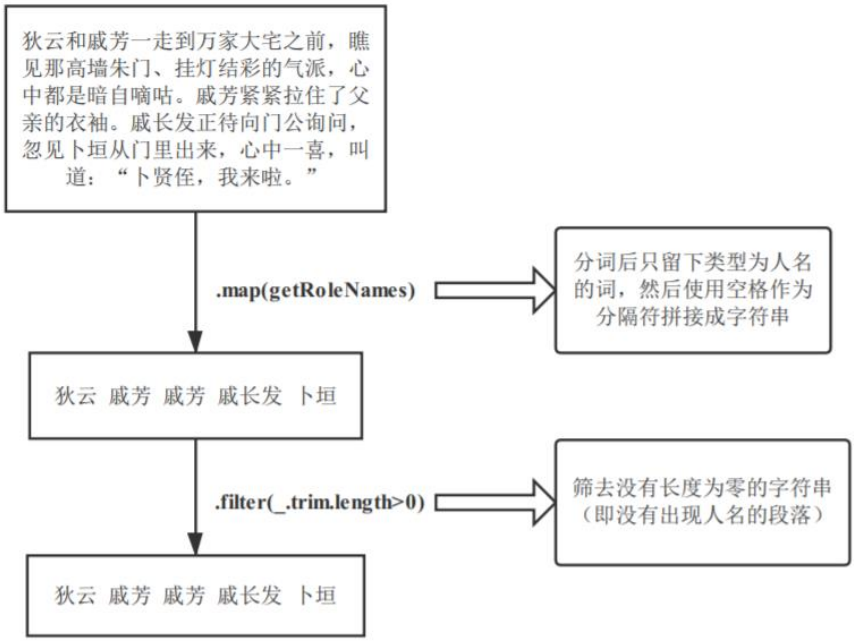
1003	神照上人
1003	晦聪禅师
1003	史松
1003	元义方
1003	杜立德
1003	刘一舟
1003	林兴珠
1003	林永超
1003	许卓诚
1003	许雪亭
1003	慕天颜
1003	九难
1003	韦春花
1003	韦小宝



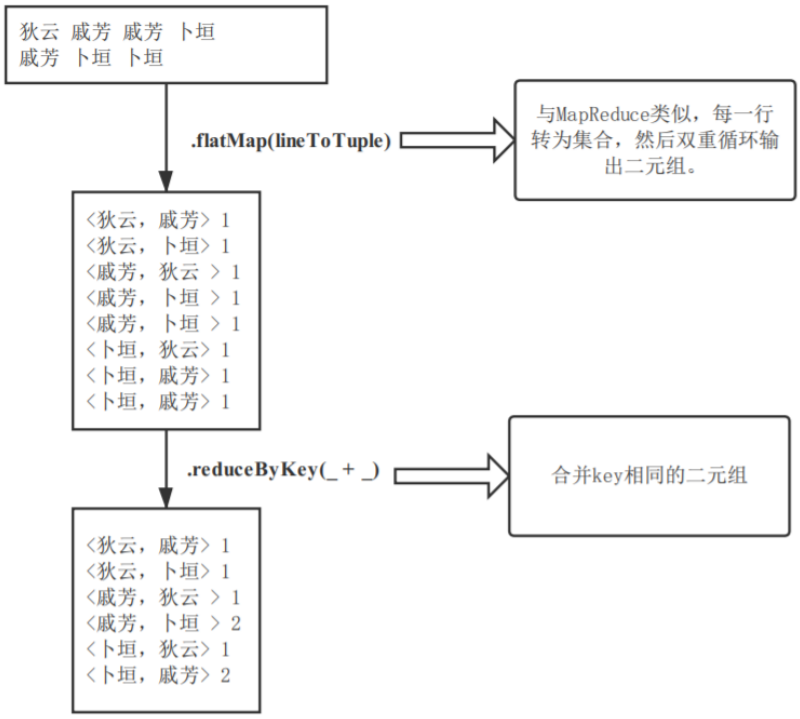
八、 Spark 实现思路及样例

由于算法已在上文详细描述过，此处不再赘述，直接通过例子表示具体的实现方法。

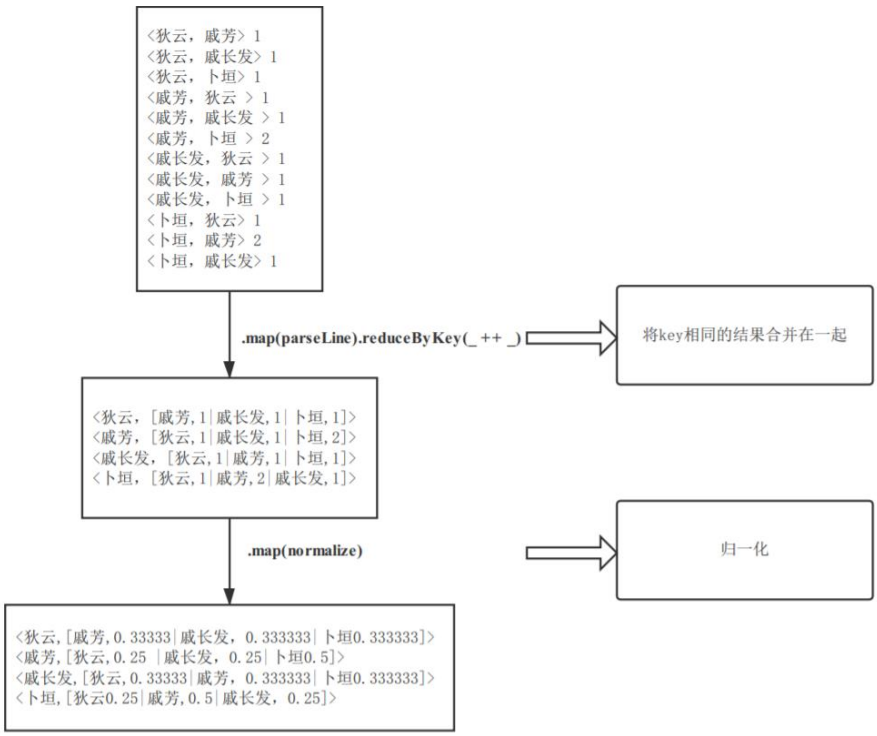
1、 任务 1



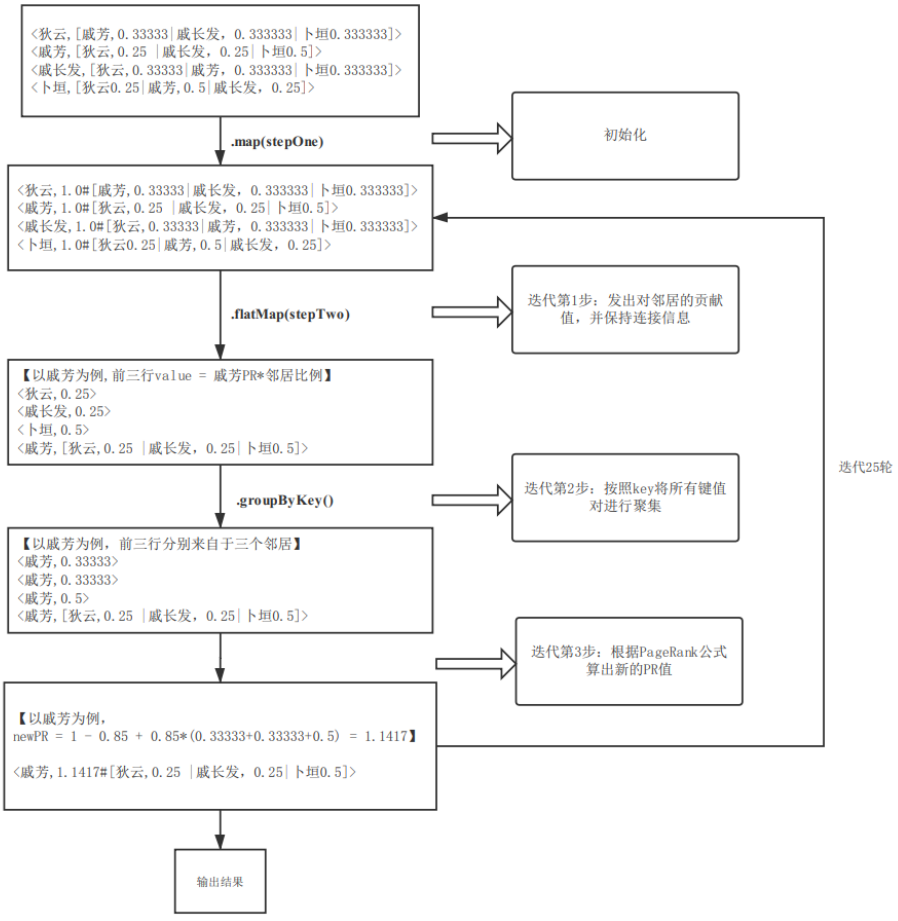
2、 任务 2



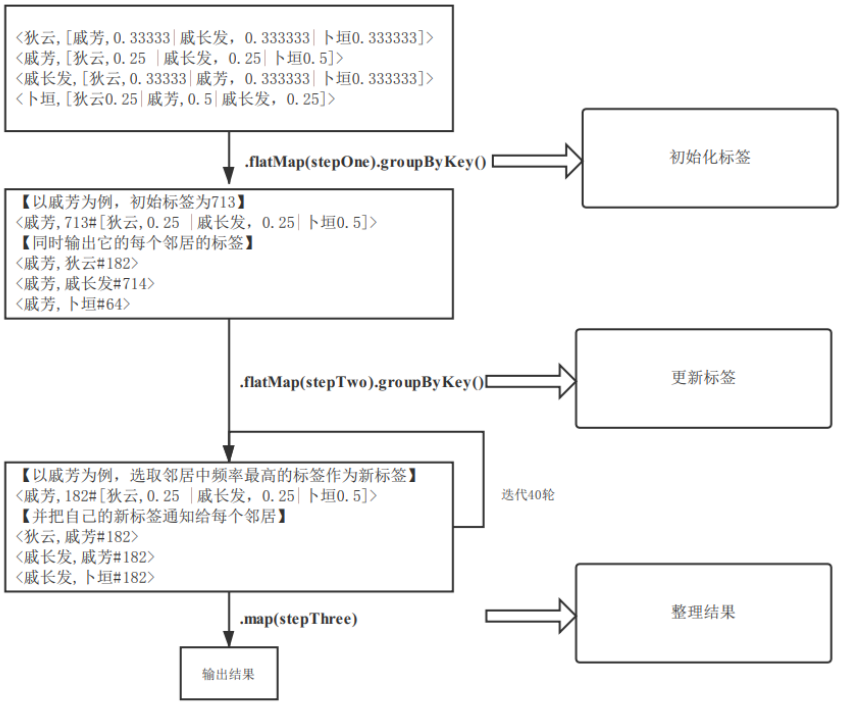
3、 任务 3


〈狄云，戚芳〉1
〈狄云，戚长发〉1
〈狄云，卜垣〉1
〈戚芳，狄云 〉1
〈戚芳，戚长发 〉1
〈戚芳，卜垣 〉2
〈戚长发，狄云 〉1
〈戚长发，戚芳 〉1
〈戚长发，卜垣 〉1
〈卜垣，狄云〉1
〈卜垣，戚芳〉2
〈卜垣，戚长发〉1
.map(parseLine).reduceByKey(_ ++ _)
将key相同的结果合并在一起
〈狄云，[戚芳,1|戚长发,1|卜垣,1]〉
〈戚芳，[狄云,1|戚长发,1|卜垣,2]〉
〈戚长发，[狄云,1|戚芳,1|卜垣,1]〉
〈卜垣，[狄云,1|戚芳,2|戚长发,1]〉
.map(normalize)
归一化
〈狄云,[戚芳,0.33333|戚长发，0.333333|卜垣0.333333]〉
〈戚芳,[狄云,0.25 |戚长发，0.25|卜垣0.5]〉
〈戚长发,[狄云,0.33333|戚芳，0.333333|卜垣0.333333]〉
〈卜垣,[狄云0.25|戚芳,0.5|戚长发，0.25]〉

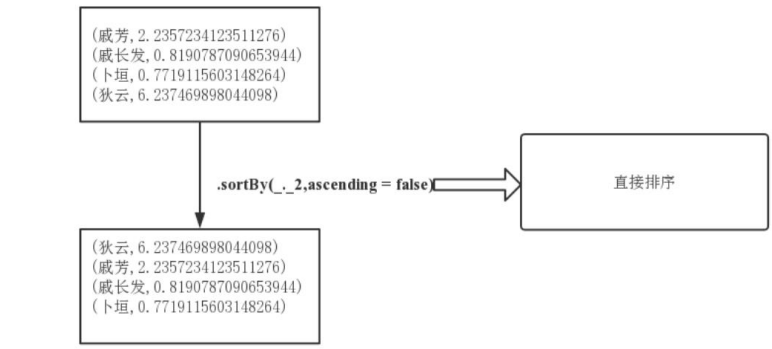
4、 任务 4

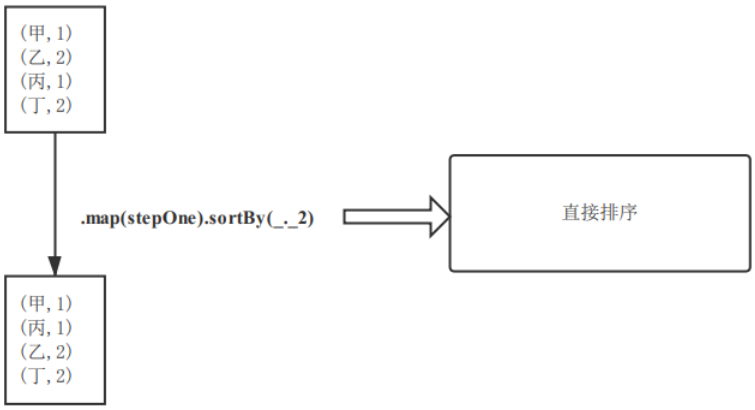

〈狄云,[戚芳,0.33333|戚长发，0.333333|卜垣0.333333]〉
〈戚芳,[狄云,0.25 |戚长发，0.25|卜垣0.5]〉
〈戚长发,[狄云,0.33333|戚芳，0.333333|卜垣0.333333]〉
〈卜垣,[狄云0.25|戚芳,0.5|戚长发，0.25]〉
.map(stepOne)
初始化
〈狄云,1.0#[戚芳,0.33333|戚长发，0.333333|卜垣0.333333]〉
〈戚芳,1.0#[狄云,0.25 |戚长发，0.25|卜垣0.5]〉
〈戚长发,1.0#[狄云,0.33333|戚芳，0.333333|卜垣0.333333]〉
〈卜垣,1.0#[狄云0.25|戚芳,0.5|戚长发，0.25]〉
.flatMap(stepTwo)
迭代第1步：发出对邻居的贡献值，并保持连接信息
【以戚芳为例,前三行value = 戚芳PR*邻居比例】
〈狄云,0.25〉
〈戚长发,0.25〉
〈卜垣,0.5〉
〈戚芳,[狄云,0.25 |戚长发，0.25|卜垣0.5]〉
.groupByKey()
迭代第2步：按照key将所有键值对进行聚集
【以戚芳为例，前三行分别来自于三个邻居】
〈戚芳,0.33333〉
〈戚芳,0.33333〉
〈戚芳,0.5〉
〈戚芳,[狄云,0.25 |戚长发，0.25|卜垣0.5]〉
迭代第3步：根据PageRank公式算出新的PR值
【以戚芳为例，
newPR = 1 - 0.85 + 0.85*(0.33333+0.33333+0.5) = 1.1417】
〈戚芳,1.1417#[狄云,0.25 |戚长发，0.25|卜垣0.5]〉
迭代25轮
输出结果

5、 任务 5


<狄云,[戚芳,0.33333|戚长发，0.333333|卜垣0.333333]>
<戚芳,[狄云,0.25 |戚长发，0.25|卜垣0.5]>
<戚长发,[狄云,0.33333|戚芳，0.333333|卜垣0.333333]>
<卜垣,[狄云0.25|戚芳,0.5|戚长发，0.25]>
.flatMap(stepOne).groupByKey()
初始化标签
【以戚芳为例，初始标签为713】
<戚芳,713#[狄云,0.25 |戚长发，0.25|卜垣0.5]>
【同时输出它的每个邻居的标签】
<戚芳,狄云#182>
<戚芳,戚长发#714>
<戚芳,卜垣#64>
.flatMap(stepTwo).groupByKey()
更新标签
【以戚芳为例，选取邻居中频率最高的标签作为新标签】
<戚芳,182#[狄云,0.25 |戚长发，0.25|卜垣0.5]>
【并把自己的新标签通知给每个邻居】
<狄云,戚芳#182>
<戚长发,戚芳#182>
<戚长发,卜垣#182>
迭代40轮
.map(stepThree)
整理结果
输出结果

6、 任务 6


(戚芳,2.2357234123511276)
(戚长发,0.8190787090653944)
(卜垣,0.7719115603148264)
(狄云,6.237469898044098)
.sortBy(_._2,ascending = false)
直接排序
(狄云,6.237469898044098)
(戚芳,2.2357234123511276)
(戚长发,0.8190787090653944)
(卜垣,0.7719115603148264)


(甲,1)
(乙,2)
(丙,1)
(丁,2)
.map(stepOne).sortBy(_._2)
直接排序
(甲,1)
(丙,1)
(乙,2)
(丁,2)

九、 实验结论

1、通过 PageRank 的运行时间，可以看出 Spark 的确比 Hadoop 拥有更高的效率。Hadoop MapReduce 在集群上跑 25 轮 PageRank 计算大约需要 6 分钟，而 Spark 在个人电脑上即可在 10 秒内完成。尽管由于硬件配置、随机性等问题没有严格地对比，但已经可以看出两者在运行时间上的差异。

2、根据 PageRank 的计算结果，可以发现我们“耳熟能详”的角色通常具有较高的值，说明他们从统计意义上也是金庸小说中绝对的主角。

3、根据标签传播的可视化结果，可以看出所有 1200 多个人物被大致分为 10 多个大群体和若干小群体，说明金庸小说世界的构建有比较大的连贯性。

十、 附录

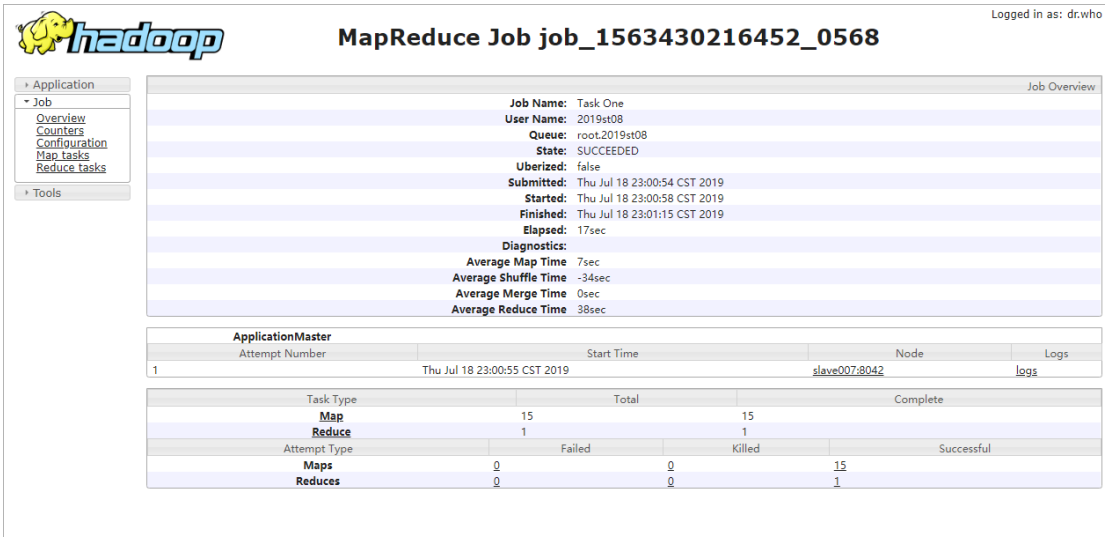
1、 实验分工

均包含代码、实验报告两部分。

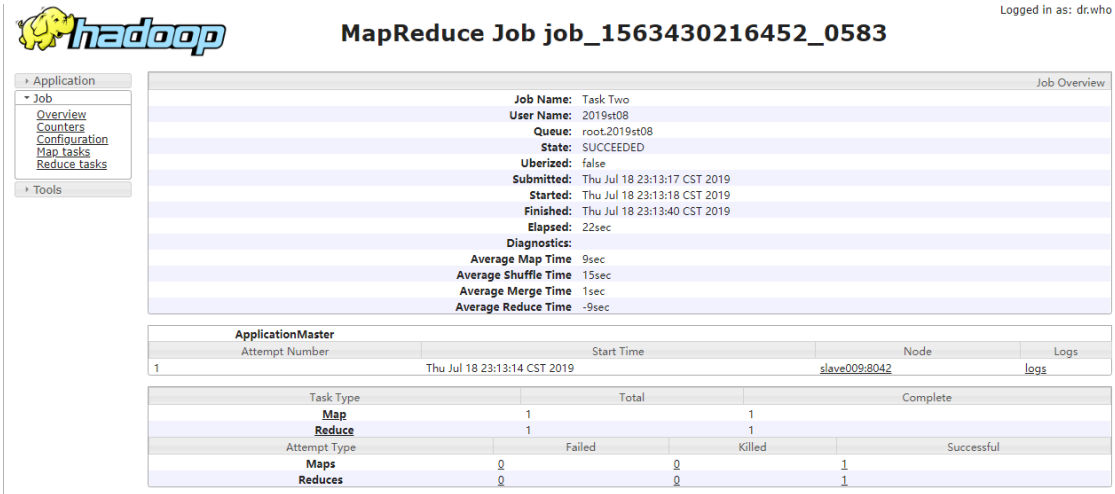
任务 1、任务 5、Spark 部分	陈 xx
任务 2、任务 4	汤 xx
任务 3、任务 6（含标签可视化）	李 xx

2、 WebUI 报告

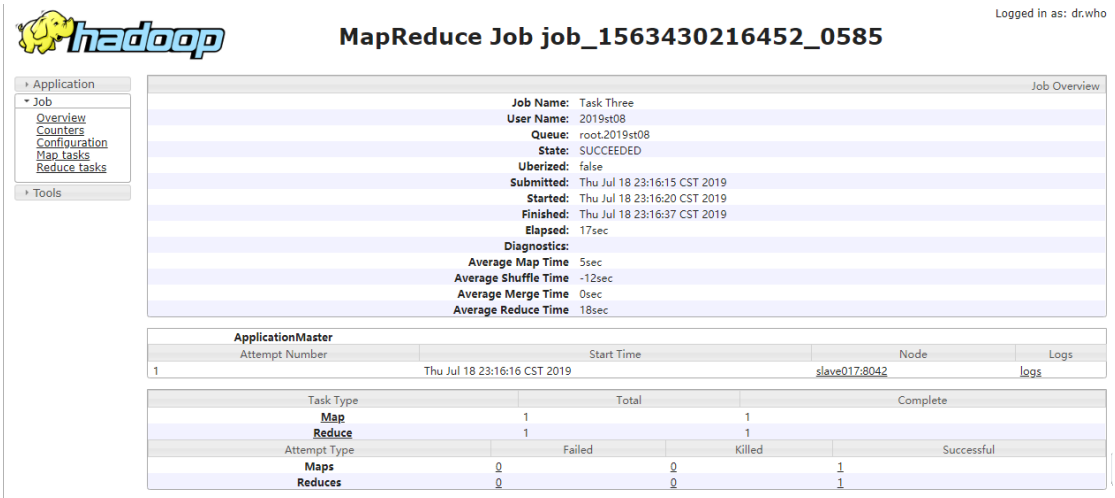
(1) 任务 1



(2) 任务 2

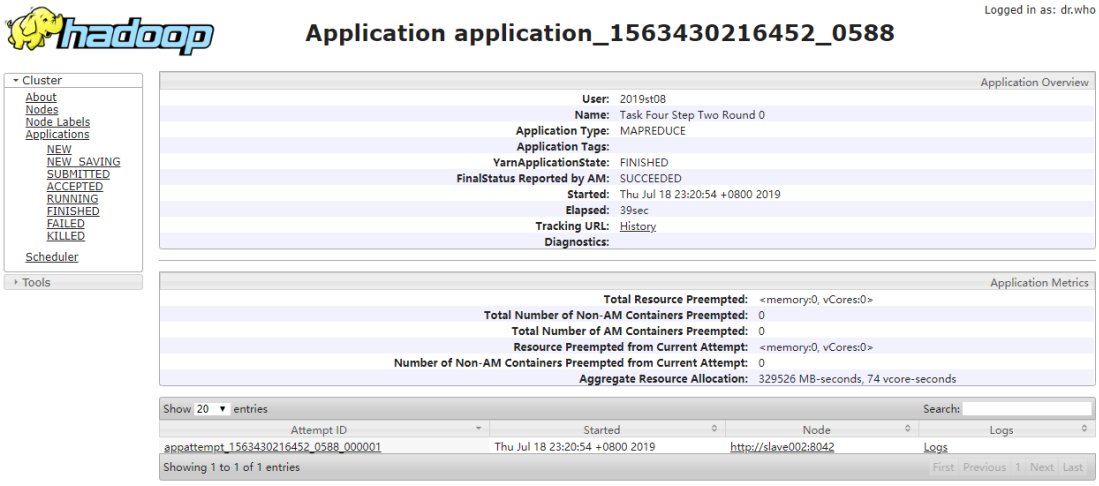


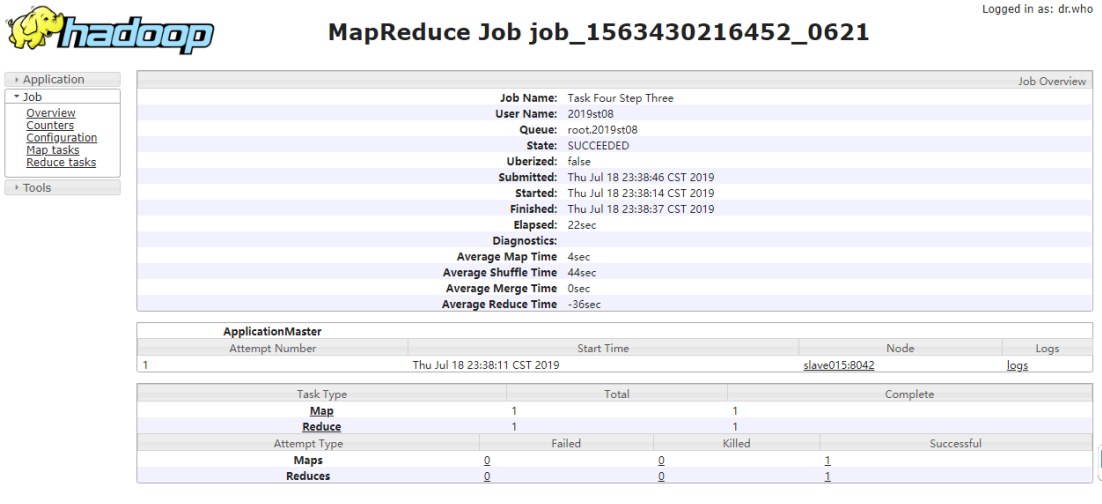
(3) 任务 3



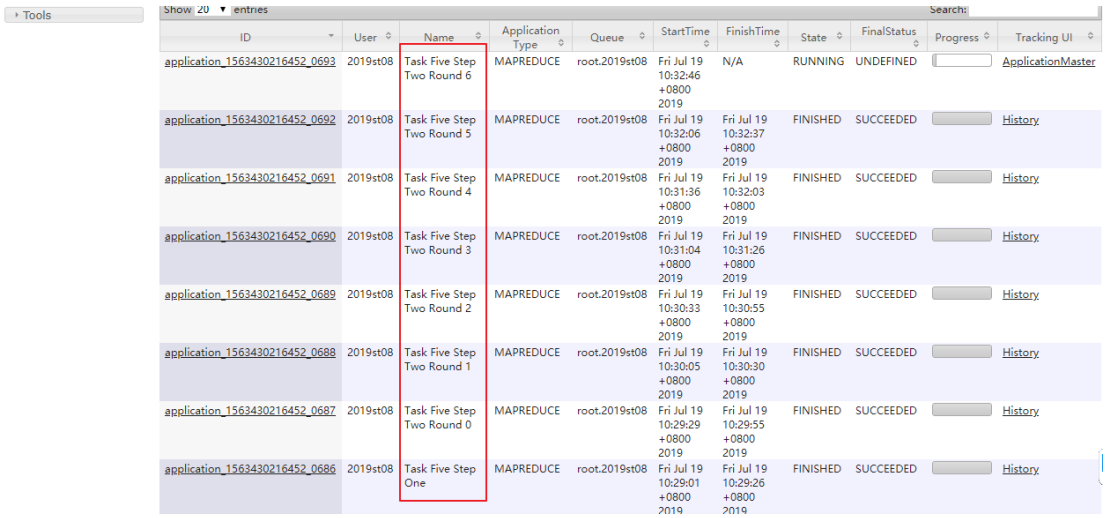
(4) 任务 4

application_1563430216452_0597	2019st08	Task Four Step Two Round 6	MAPREDUCE	root.2019st08	Thu Jul 18 23:24:31 +0800 2019	N/A	ACCEPTED	UNDEFINED		ApplicationMa
application_1563430216452_0596	2019st08	Task Four Step Two Round 5	MAPREDUCE	root.2019st08	Thu Jul 18 23:23:49 +0800 2019	Thu Jul 18 23:24:21 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0595	2019st08	Task Four Step Two Round 4	MAPREDUCE	root.2019st08	Thu Jul 18 23:23:23 +0800 2019	Thu Jul 18 23:23:46 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0594	2019st08	Task Four Step Two Round 3	MAPREDUCE	root.2019st08	Thu Jul 18 23:22:48 +0800 2019	Thu Jul 18 23:23:13 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0593	2019st08	Task Four Step Two Round 2	MAPREDUCE	root.2019st08	Thu Jul 18 23:22:21 +0800 2019	Thu Jul 18 23:22:46 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0592	2019st01	PageRankSort	MAPREDUCE	root.2019st01	Thu Jul 18 23:22:08 +0800 2019	Thu Jul 18 23:22:28 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0591	2019st08	Task Four Step Two Round 1	MAPREDUCE	root.2019st08	Thu Jul 18 23:21:45 +0800 2019	Thu Jul 18 23:22:10 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0590	2019st18	Prediciton	MAPREDUCE	root.2019st18	Thu Jul 18 23:21:40 +0800 2019	Thu Jul 18 23:22:06 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0588	2019st08	Task Four Step Two Round 0	MAPREDUCE	root.2019st08	Thu Jul 18 23:20:54 +0800	Thu Jul 18 23:21:34 +0800 2019	FINISHED	SUCCEEDED		History


Logged in as: dr.who
Application application_1563430216452_0588
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Tools
Application Overview
User: 2019st08
Name: Task Four Step Two Round 0
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
FinalStatus Reported by AM: SUCCEEDED
Started: Thu Jul 18 23:20:54 +0800 2019
Elapsed: 39sec
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: 329526 MB-seconds, 74 vcore-seconds
Show 20 entries
Search:
Attempt ID
Started
Node
Logs
appattempt_1563430216452_0588_000001
Thu Jul 18 23:20:54 +0800 2019
http://slave002:8042
Logs
Showing 1 to 1 of 1 entries
First Previous 1 Next Last


Logged in as: dr.who
MapReduce Job job_1563430216452_0621
Application
Job
Overview
Counters
Configuration
Map tasks
Reduce tasks
Tools
Job Overview
Job Name: Task Four Step Three
User Name: 2019st08
Queue: root.2019st08
State: SUCCEEDED
Uberized: false
Submitted: Thu Jul 18 23:38:46 CST 2019
Started: Thu Jul 18 23:38:14 CST 2019
Finished: Thu Jul 18 23:38:37 CST 2019
Elapsed: 22sec
Diagnostics:
Average Map Time 4sec
Average Shuffle Time 44sec
Average Merge Time 0sec
Average Reduce Time -36sec
ApplicationMaster
Attempt Number
Start Time
Node
Logs
1
Thu Jul 18 23:38:11 CST 2019
slave015:8042
logs
Task Type
Total
Complete
Map
1
1
Reduce
1
1
Attempt Type
Failed
Killed
Successful
Maps
0
0
1
Reduces
0
0
1

(5) 任务 5


Tools
Show 20 entries
Search:
ID
User
Name
Application Type
Queue
StartTime
FinishTime
State
FinalStatus
Progress
Tracking UI
application_1563430216452_0693 2019st08 Task Five Step Two Round 6 MAPREDUCE root.2019st08 Fri Jul 19 10:32:46 +0800 2019 N/A RUNNING UNDEFINED ApplicationMaster
application_1563430216452_0692 2019st08 Task Five Step Two Round 5 MAPREDUCE root.2019st08 Fri Jul 19 10:32:06 +0800 2019 Fri Jul 19 10:32:37 +0800 2019 FINISHED SUCCEEDED History
application_1563430216452_0691 2019st08 Task Five Step Two Round 4 MAPREDUCE root.2019st08 Fri Jul 19 10:31:36 +0800 2019 Fri Jul 19 10:32:03 +0800 2019 FINISHED SUCCEEDED History
application_1563430216452_0690 2019st08 Task Five Step Two Round 3 MAPREDUCE root.2019st08 Fri Jul 19 10:31:04 +0800 2019 Fri Jul 19 10:31:26 +0800 2019 FINISHED SUCCEEDED History
application_1563430216452_0689 2019st08 Task Five Step Two Round 2 MAPREDUCE root.2019st08 Fri Jul 19 10:30:33 +0800 2019 Fri Jul 19 10:30:55 +0800 2019 FINISHED SUCCEEDED History
application_1563430216452_0688 2019st08 Task Five Step Two Round 1 MAPREDUCE root.2019st08 Fri Jul 19 10:30:05 +0800 2019 Fri Jul 19 10:30:30 +0800 2019 FINISHED SUCCEEDED History
application_1563430216452_0687 2019st08 Task Five Step Two Round 0 MAPREDUCE root.2019st08 Fri Jul 19 10:29:29 +0800 2019 Fri Jul 19 10:29:55 +0800 2019 FINISHED SUCCEEDED History
application_1563430216452_0686 2019st08 Task Five Step One MAPREDUCE root.2019st08 Fri Jul 19 10:29:01 +0800 2019 Fri Jul 19 10:29:26 +0800 2019 FINISHED SUCCEEDED History

Logged in as: dr.who

MapReduce Job job_1563430216452_0687

- Application
- Job
  - Overview
  - Counters
  - Configuration
  - Map tasks
  - Reduce tasks
- Tools

Job Overview

Job Name:	Task Five Step Two Round 0
User Name:	2019st08
Queue:	root.2019st08
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Jul 19 10:29:29 CST 2019
Started:	Fri Jul 19 10:29:33 CST 2019
Finished:	Fri Jul 19 10:29:49 CST 2019
Elapsed:	16sec
Diagnostics:	
Average Map Time	5sec
Average Shuffle Time	2sec
Average Merge Time	0sec
Average Reduce Time	3sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Fri Jul 19 10:29:27 CST 2019	slave017:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

Logged in as: dr.who

MapReduce Job job_1563430216452_0737

- Application
- Job
  - Overview
  - Counters
  - Configuration
  - Map tasks
  - Reduce tasks
- Tools

Job Overview

Job Name:	Task Five Step Three
User Name:	2019st08
Queue:	root.2019st08
State:	SUCCEEDED
Uberized:	false
Submitted:	Fri Jul 19 10:52:23 CST 2019
Started:	Fri Jul 19 10:52:27 CST 2019
Finished:	Fri Jul 19 10:52:46 CST 2019
Elapsed:	19sec
Diagnostics:	
Average Map Time	5sec
Average Shuffle Time	-18sec
Average Merge Time	0sec
Average Reduce Time	24sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Fri Jul 19 10:52:22 CST 2019	slave017:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

(6) 任务 6

- NEW
- NEW_SAVING
- SUBMITTED
- ACCEPTED
- RUNNING
- FINISHED
- FAILED
- KILLED
- Scheduler
- Tools

User Metrics for dr.who

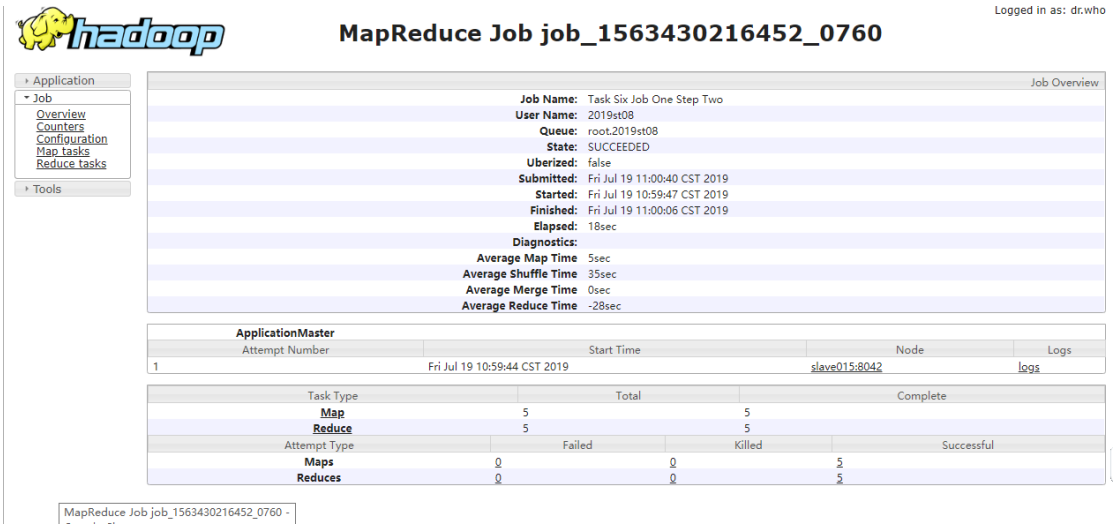
Apps Submitted	Apps Pending	Apps Running	Apps Completed	Containers Running	Containers Pending	Containers Reserved	Memory Used	Memory Pending	Memory Reserved	VCores Used	VCores Pending	VCores Reserve
0	0	0	0	0	0	0	0 B	0 B	0 B	0	0	0

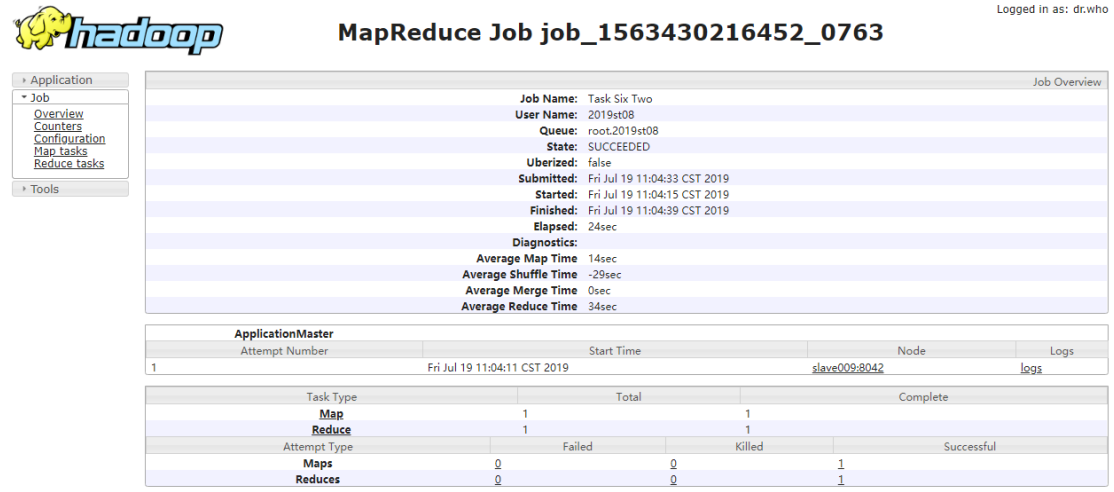
Scheduler Metrics

Scheduler Type	Scheduling Resource Type	Minimum Allocation	Maximum Allocation
Fair Scheduler	[MEMORY, CPU]	<memory:2048, vCores:1>	<memory:10240, vCores:10>

Show 20 entries Search:

ID	User	Name	Application Type	Queue	StartTime	FinishTime	State	FinalStatus	Progress	Tracking UI
application_1563430216452_0761	2019st01	CooccurrenceCounting	MAPREDUCE	root.2019st01	Fri Jul 19 11:01:04 +0800 2019	N/A	ACCEPTED	UNDEFINED		ApplicationMas
application_1563430216452_0760	2019st08	Task Six Job One Step Two	MAPREDUCE	root.2019st08	Fri Jul 19 11:00:40 +0800 2019	Fri Jul 19 11:01:04 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0759	2019st08	Task Six Job One Step One	MAPREDUCE	root.2019st08	Fri Jul 19 11:00:00 +0800 2019	Fri Jul 19 11:00:36 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0758	2019st19	Task5-3	MAPREDUCE	root.2019st19	Fri Jul 19 10:58:49 +0800 2019	Fri Jul 19 10:59:12 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0757	2019st01	CharacterExtracting	MAPREDUCE	root.2019st01	Fri Jul 19 10:58:27 +0800 2019	Fri Jul 19 10:59:01 +0800 2019	FINISHED	SUCCEEDED		History
application_1563430216452_0756	2019st19	Task5-29	MAPREDUCE	root.2019st19	Fri Jul 19 10:58:16 +0800	Fri Jul 19 10:58:41 +0800	FINISHED	SUCCEEDED		History





3、 参考资料

[1]Raghavan, Usha Nandini, Réka Albert, and Soundar Kumara. "Near linear time algorithm to detect community structures in large-scale networks." *Physical review E* 76.3 (2007): 036106.

[2]DistributedCache 使用：https://buhrmann.github.io/hadoop-distributed-cache.html

[3]全局排序：https://zhuanlan.zhihu.com/p/43851100

[4] https://hadoop.apache.org/docs/r2.7.1/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

[5] https://spark.apache.org/docs/latest/rdd-programming-guide.html