大数据处理综合实验

课程设计 金庸的江湖 实验报告

组别：2020st39

组长：171860662 山越 1025331716@qq.com

目录

一．任务一 数据预处理

（一） 任务设计

1. 设计思路

从已经做好分词的金庸小说文本中，通过利用人名列表抽取出与人物互动相关的数据，而屏蔽掉与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。

由于操作较为简单，只需在 map 阶段处理即可。

通过重写 setup，从 cacheFile 读入人名列表文件，并存储在名为 people_list 的数组中。

通过重写 map，key 是读入的小说段落的偏移量，value 是该段落的的分词结果。通过文件名筛选出金庸所写的小说。利用 people_list 判断每一个分词是否为人名，将筛选出的人名以空格为间隔拼接在一个字符串 name 中即可，注意要删除末尾的空格。

2. 源代码说明

Job1_Driver.java

```
public class Job1_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 3) {
            System.err.println("Job1: <novels> <people_list> <job1_out>");
            System.exit(2);
        }
        Configuration conf = new Configuration();
        Job job1 = Job.getInstance(conf, "Job1");
        job1.setJarByClass(Job1_Driver.class);
        job1.addCacheFile(new Path(args[1]).toUri());
        job1.setMapperClass(Job1_Mapper.class);
        job1.setMapOutputKeyClass(Text.class);
        job1.setMapOutputValueClass(NullWritable.class);
        job1.setOutputKeyClass(Text.class);
        job1.setOutputValueClass(NullWritable.class);
        FileInputFormat.addInputPath(job1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job1, new Path(args[2]));
        job1.waitForCompletion(true);
    }
}
```

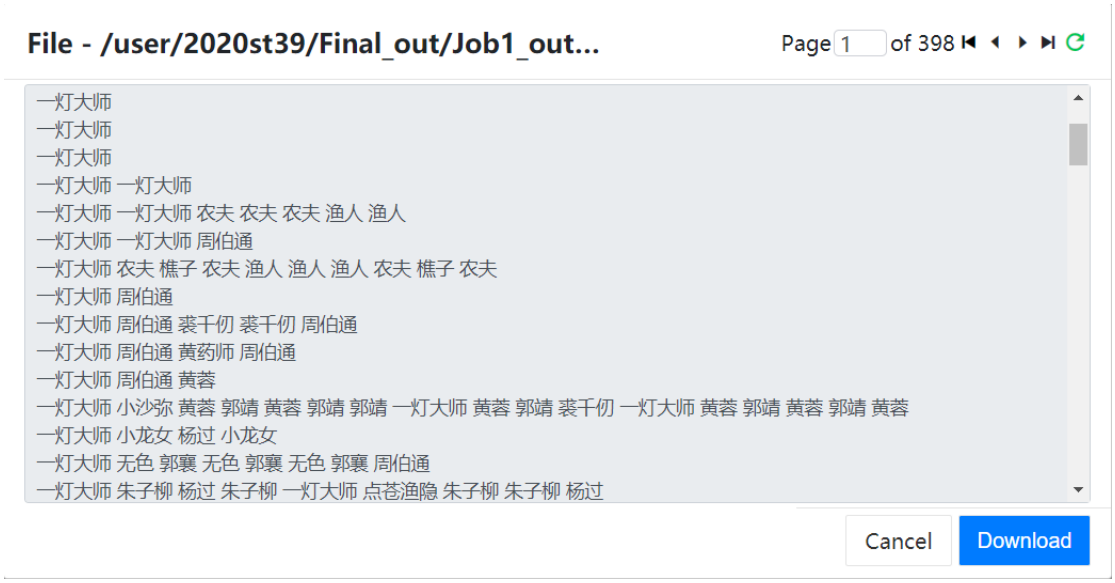
Job1_Mapper.java

```java
public class Job1_Mapper extends Mapper<LongWritable, Text, Text, NullWritable>
{
    private String filename;
    private List<String> people_list = new ArrayList<>();

    @Override
    public void setup(Context context) throws IOException, InterruptedException {
        FileSplit fs = (FileSplit) context.getInputSplit();
        filename = fs.getPath().getName();
        try {
            Path[] cacheFiles = context.getLocalCacheFiles(); //人名列表文件作为cacheFile读入
            if (cacheFiles != null && cacheFiles.length > 0) {
                String line;
                BufferedReader fileReader = new BufferedReader(new FileReader(cacheFiles[0].toString()));
                try {
                    while ((line = fileReader.readLine()) != null) { //循环读入人名列表文件中的每一行
                        people_list.add(line); //将所有人名添加到people_list中
                    }
                }
                finally {
                    fileReader.close();
                }
            }
        }
        catch (IOException e) {
            System.err.println("Exception reading DistributedCache: " + e);
        }
    }


    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String author = filename.split("[0-9]{2}")[0]; //从文件名分割出作者
        if (!author.equals("金庸")) //筛选出非金庸写的小说
            return;

        String names = new String(); //存放文本中筛选的名字拼接而成的字符串
        StringTokenizer itr = new StringTokenizer(value.toString()); //指向文本中各个分词token的指针
        boolean hasName = false; //判断该小说中是否至少存在一个人名的标志位
        while (itr.hasMoreTokens()) { //遍历所有分词token
            String word = itr.nextToken(); //存储一个分词
            if (people_list.contains(word)) { //判断该分词是否为人名
                names = names + word + " "; //若是人名则添加在names末尾，并置标志位为true
                hasName = true;
            }
        }
        if (hasName) {
            names = names.substring(0, names.length() - 1); //去除names末尾的最后一个空格
            context.write(new Text(names), NullWritable.get()); //发射筛选人名的结果
        }
    }
}
```

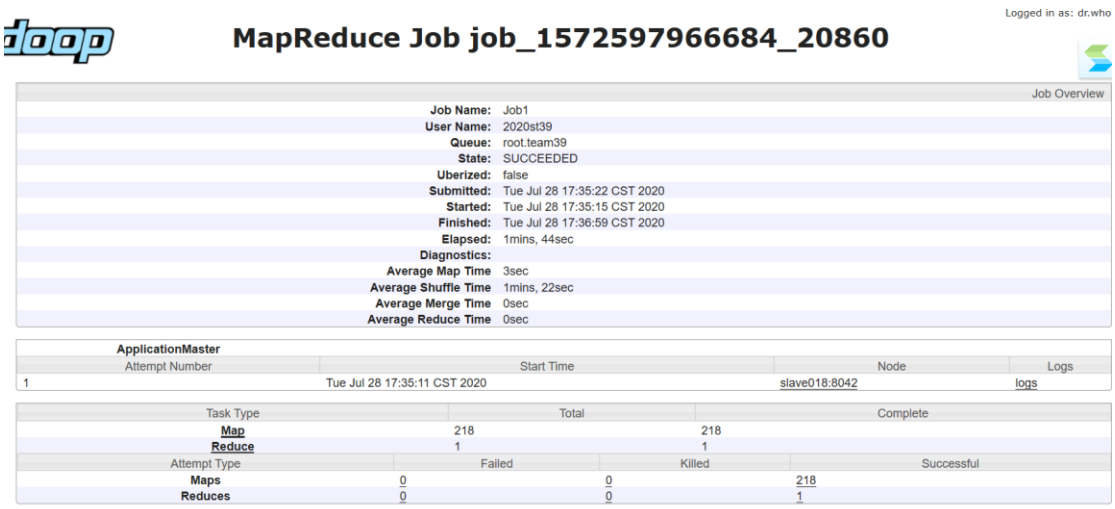
（二） 程序运行和结果说明

1. 输出结果

输出文件在 HDFS 上的路径：/user/2020st39/Final_out/Job1_out



2. WebUI 执行报告



二．任务二 特征抽取：人物同现统计

（一） 任务设计

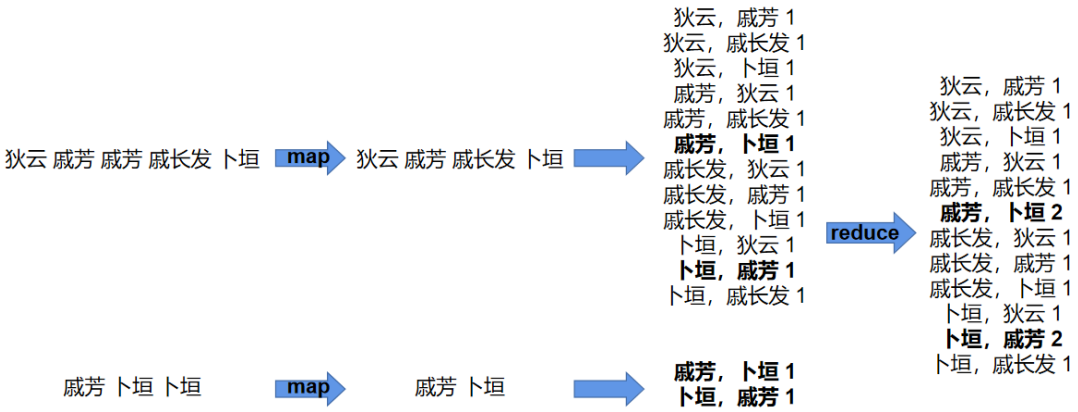
1. 设计思路

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。

在 map 中，key 是读取任务 1 结果的偏移量，value 是文本内容，即某一段落中筛选出来的人名所拼接成的字符串。将 value 用空格分割并存储在 temp_names 数组中，去除其中重复的人名，获得新的人名 list 命名为 names。双层循环遍历 names 中的每两个名字作为 key，数值 1 作为 value，将该键值对发送出去。

通过 combiner 进行优化，在 mapper 端将所有 key 相同的 value 加和，以达到减少发送信息量的目的，优化从 mapper 到 reducer 的传递过程。

Reduce 的代码与 combiner 完全相同，都是将 key 相同的 value 加和，但这是在 reducer 端做的加和，以保证程序正确性。



2. 源代码说明

Job2_Driver.java

```java
public class Job2_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 2) {
            System.err.println("Job2: <job1_out> <job2_out>");
            System.exit(2);
        }
        Configuration conf = new Configuration();
        Job job2 = Job.getInstance(conf, "Job2");
        job2.setJarByClass(Job2_Driver.class);
        job2.setMapperClass(Job2_Mapper.class);
        job2.setCombinerClass(Job2_Combiner.class);
        job2.setReducerClass(Job2_Reducer.class);
        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(IntWritable.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job2, new Path(args[0]));
        FileOutputFormat.setOutputPath(job2, new Path(args[1]));
        job2.waitForCompletion(true);
    }
}
```

Job2_Mapper.java

```java
public class Job2_Mapper extends Mapper<LongWritable, Text, Text, IntWritable>
{
    @Override
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String[] temp_names = value.toString().split(" "); //读入任务一结果，将用空格分隔的字符串存储在字符数组中
        List<String> names = new ArrayList<>(); //存储去除重复后的某段落中所有人名
        for (int i = 0; i < temp_names.length; i++) { //去除重复的人名
            if (!names.contains(temp_names[i])) {
                names.add(temp_names[i]);
            }
        }
        if (names.size() < 2)
            return;
        IntWritable one = new IntWritable(1); //初始化基本计数单位为 1
        for (int i = 0; i < names.size(); i++) { //双层循环遍历所有组合
            for (int j = i + 1; j < names.size(); j++) { //以两个人名拼接为key，计数1为value发送
                context.write(new Text(names.get(i) + "," + names.get(j)), one);
                context.write(new Text(names.get(j) + "," + names.get(i)), one);
            }
        }
    }
}
```

Job2_Combiner.java

```java
public class Job2_Combiner extends Reducer<Text, IntWritable, Text, IntWritable>
{
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sum = 0;
        Iterator<IntWritable> iter = values.iterator();
        while (iter.hasNext()) {
            sum += iter.next().get(); //将key值相同的value加和
        }
        context.write(key, new IntWritable(sum));
    }
}
```

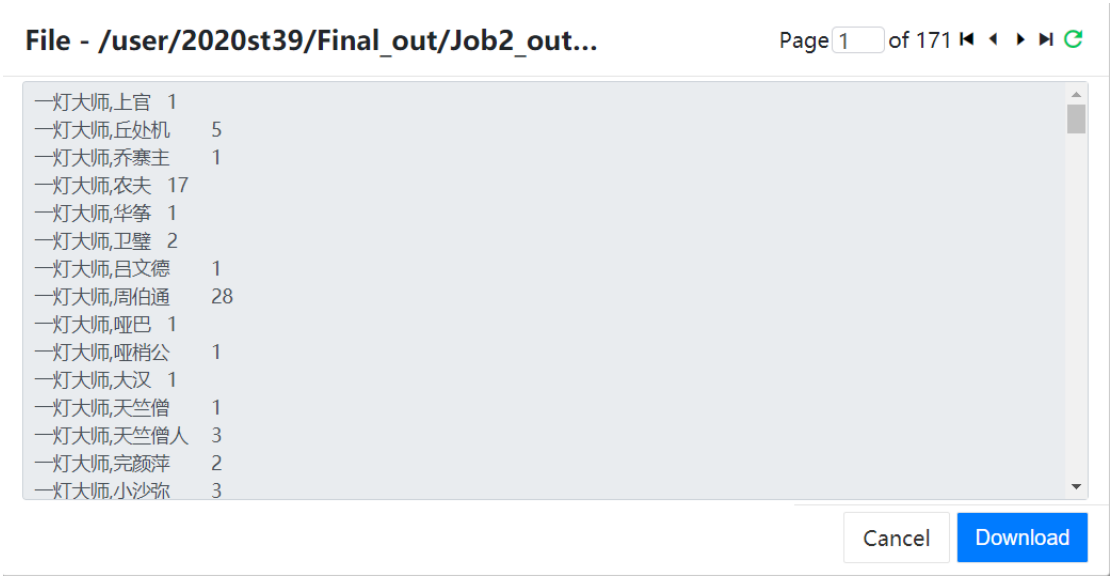
Job2_Reducer.java

```java
public class Job2_Reducer extends Reducer<Text, IntWritable, Text, IntWritable>
{
    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
        int sum = 0;
        Iterator<IntWritable> iter = values.iterator();
        while (iter.hasNext()) {
            sum += iter.next().get(); //将key值相同的value加和
        }
        context.write(key, new IntWritable(sum));
    }
}
```

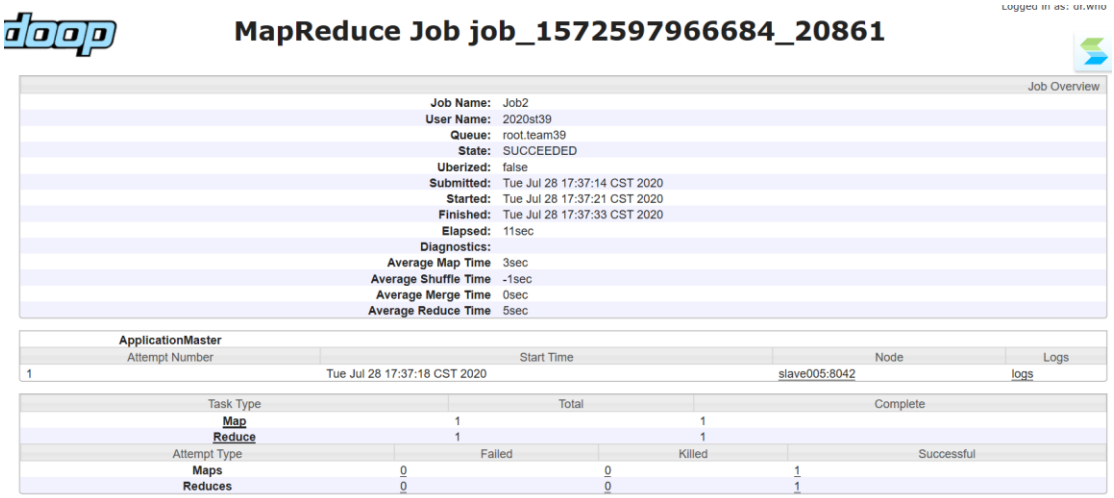
（二）程序运行和结果说明

1. 输出结果

输出文件在 HDFS 上的路径：/user/2020st39/Final_out/Job2_out



2. WebUI 执行报告



MapReduce Job job_1572597966684_20861

Job Overview

Job Name:	Job2
User Name:	2020st39
Queue:	root.team39
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Jul 28 17:37:14 CST 2020
Started:	Tue Jul 28 17:37:21 CST 2020
Finished:	Tue Jul 28 17:37:33 CST 2020
Elapsed:	11sec
Diagnostics:	
Average Map Time	3sec
Average Shuffle Time	-1sec
Average Merge Time	0sec
Average Reduce Time	5sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jul 28 17:37:18 CST 2020	slave005:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

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

三．任务三 人物关系图构建与特征归一化

（一） 任务设计

1. 设计思路

当从任务二得到人物同现统计的结果后，可以根据共现关系生成人物之间的关系图。在人物关系图中，人物是顶点，人物之间的互动关系是边。两个人物之间的共现次数体现出两个人物关系的密切程度，反映到人物关系图中就是边的权重。为了方便后面任务对数据进行处理，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频率。

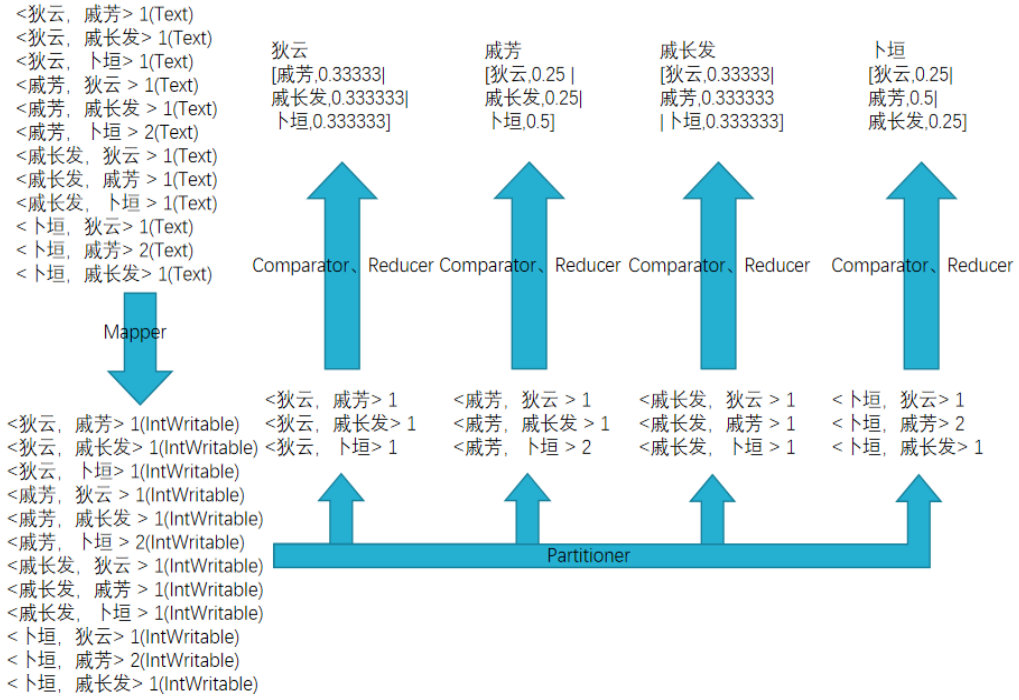
数据的转换如下图所示。

<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

→

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]

任务三中有 Mapper、Partitioner、Comparator、Reducer 四部分代码，各部分对数据的作用如下所示：



2. 源代码说明

Job3_Driver.java

```
public class Job3_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 2)
        {
            System.err.println("Job3: <job2_out> <job3_out>");
            System.exit(2);
        }

        Configuration conf = new Configuration();
        Job job3 = Job.getInstance(conf, "Job3");
        job3.setJarByClass(Job3_Driver.class);
        job3.setInputFormatClass(KeyValueTextInputFormat.class);
        job3.setMapperClass(Job3_Mapper.class);
        job3.setPartitionerClass(Job3_Partitioner.class);
        job3.setGroupingComparatorClass(Job3_Comparator.class);
        job3.setReducerClass(Job3_Reducer.class);
        job3.setMapOutputKeyClass(Text.class);
        job3.setMapOutputValueClass(IntWritable.class);
        job3.setOutputKeyClass(Text.class);
        job3.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job3, new Path(args[0]));
        FileOutputFormat.setOutputPath(job3, new Path(args[1]));
        job3.waitForCompletion(true);
    }
}
```

Job3_Mapper.java

将 Text 类型的 value 变量转化为 IntWritable 类型方便后面进行加和操作。

```
public class Job3_Mapper extends Mapper<Text, Text, Text, IntWritable>
{
    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {
        IntWritable newvalue = new IntWritable(Integer.parseInt(value.toString()));//将读取的Text类型value转化为IntWritable
        context.write(key, newvalue);
    }
}
```

Job3_Partitioner.java

将键值对的第一个名字相同的数据发送到同一个节点中。

```
public class Job3_Partitioner extends HashPartitioner<Text, IntWritable>
{
    @Override
    public int getPartition(Text key, IntWritable value, int numReduceTasks)
    {
        String first = key.toString().split(",")[0];//将例如<狄云，戚芳>键值对的第一个名字存入first
        return super.getPartition(new Text(first), value, numReduceTasks);//将first相同的数据发到同一个reducer中
    }
}
```

Job3_Comparator.java

比较两条数据的第一个名字是否相同。如果相同则将两条数据放到同一趟 Reducer 循环中处理。

```
public class Job3_Comparator extends WritableComparator
{
    protected Job3_Comparator() { super(Text.class, true); }

    @Override
    public int compare(WritableComparable a, WritableComparable b)
    {
        Text ta = (Text) a;
        Text tb = (Text) b;
        String a_first = ta.toString().split(",")[0];//将例如<狄云，戚芳>键值对的第一个名字存放到a_first中
        String b_first = tb.toString().split(",")[0];//将例如<狄云，戚长发>键值对的第一个名字存放到b_first中
        return a_first.compareTo(b_first);//比较两个变量是否相同，相同的数据在同一次Reducer循环中处理
    }
}
```

Job3_Reducer.java

```java
public class Job3_Reducer extends Reducer<Text, IntWritable, Text, Text>
{
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException
    {
        List<String> keylist=new ArrayList<String>();//存储key值
        List<Integer> valuelist=new ArrayList<Integer>();//存储value值

        int sum = 0;
        Iterator<IntWritable> iter1 = values.iterator();//迭代器遍历values
        while (iter1.hasNext())
        {
            int cur = iter1.next().get();//cur存储当前的value值
            keylist.add(key.toString());//将当前key值添加到keylist中
            valuelist.add(cur);//将cur中存储的当前value添加到valuelist中
            sum += cur;//加和操作
        }

        String newkey = keylist.get(0).split(",")[0];//将例如<狄云，戚芳>键值对的第一个名字放到newkey中，且一次reducer函数中第一个名字相同
        String newvalue = new String();//newvalue存储最终输出字符串
        for(int i = 0; i < keylist.size(); i++)
        {
            float prob = valuelist.get(i) / (float)sum;//计算同现概率，进行归一化处理
            newvalue = newvalue + keylist.get(i).split(",")[1] + "," + Float.toString(prob) + "|";//将键值对第二个名字+同现概率添加到newvalue中
        }
        newvalue = newvalue.substring(0, newvalue.length() - 1);//去掉最后一个分隔符'|'
        context.write(new Text(newkey), new Text(newvalue));
    }
}
```

（二） 程序运行和结果说明

1. 输出结果

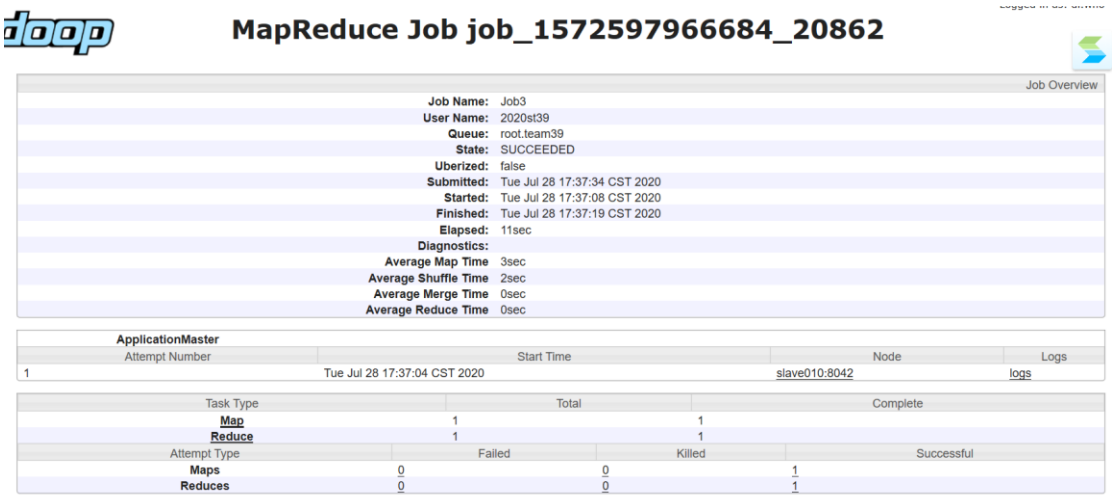
输出文件在 HDFS 上的路径：/user/2020st39/Final_out/Job3_out

File - /user/2020st39/Final_out/Job3_out... Page 1 of 176

```
一灯大师 上官,0.0023474179|丘处机,0.01173709|乔寨主,0.0023474179|农夫,0.039906103|华筝,0.0023474179|卫
璧,0.0046948357|吕文德,0.0023474179|周伯通,0.065727696|哑巴,0.0023474179|哑梢公,0.0023474179|大
汉,0.0023474179|天竺僧,0.0023474179|天竺僧人,0.0070422534|完颜萍,0.0046948357|小沙弥,0.0070422534|小龙
女,0.018779343|尹克西,0.0023474179|尼摩星,0.0070422534|张无忌,0.0046948357|无色,0.0023474179|明
月,0.0023474179|朱九真,0.0046948357|朱子柳,0.030516433|朱长龄,0.0023474179|李莫愁,0.01173709|杨
康,0.0070422534|杨过,0.07981221|柯镇恶,0.0046948357|梅超风,0.0023474179|樵子,0.02112676|欧阳
克,0.0070422534|武三娘,0.0046948357|武三通,0.032863848|武修文,0.0070422534|武敦儒,0.0046948357|武青
婴,0.0046948357|汉子,0.0046948357|洪七公,0.025821596|渔人,0.028169014|潇湘子,0.0023474179|点苍渔
隐,0.0070422534|琴儿,0.0023474179|穆念慈,0.0023474179|老头子,0.0023474179|耶律燕,0.0070422534|耶律
齐,0.01173709|裘千丈,0.0023474179|裘千仞,0.030516433|裘千尺,0.014084507|觉远,0.0023474179|觉远大
师,0.0023474179|达尔巴,0.0046948357|郝大通,0.0046948357|郭芙,0.0093896715|郭襄,0.04225352|郭靖,0.11737089|
金轮法王,0.0093896715|陆乘风,0.0023474179|陆无双,0.016431924|霍都,0.0070422534|马钰,0.0046948357|鲁有
脚,0.0093896715|黄药师,0.03521127|黄蓉,0.16901408
丁不三 丁不四,0.16564417|侍剑,0.006134969|史小翠,0.006134969|大悲老人,0.006134969|大汉,0.006134969|孙万
```

Cancel Download

2. WebUI 执行报告



MapReduce Job job_1572597966684_20862

Job Overview

Job Name:	Job3
User Name:	2020st39
Queue:	root.team39
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Jul 28 17:37:34 CST 2020
Started:	Tue Jul 28 17:37:08 CST 2020
Finished:	Tue Jul 28 17:37:19 CST 2020
Elapsed:	11sec
Diagnostics:	
Average Map Time	3sec
Average Shuffle Time	2sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jul 28 17:37:04 CST 2020	slave010:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

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

四．任务四 基于人物关系图的 PageRank 计算

（一） 任务设计

1. 设计思路

根据课件中的 PageRank 算法，将代码分为三个部分：

Job4_Driver	2020/7/15 13:13	IntelliJ IDEA Co...	2 KB
Job4_GraphBuilder_Driver	2020/7/5 0:38	IntelliJ IDEA Co...	2 KB
Job4_GraphBuilder_Mapper	2020/7/15 12:12	IntelliJ IDEA Co...	1 KB
Job4_PageRankIter_Comparator	2020/7/3 15:05	IntelliJ IDEA Co...	1 KB
Job4_PageRankIter_Driver	2020/7/15 13:03	IntelliJ IDEA Co...	2 KB
Job4_PageRankIter_Mapper	2020/7/3 15:05	IntelliJ IDEA Co...	2 KB
Job4_PageRankIter_Partitioner	2020/7/4 9:37	IntelliJ IDEA Co...	1 KB
Job4_PageRankIter_Reducer	2020/7/15 12:12	IntelliJ IDEA Co...	1 KB
Job4_PageRankViewer_Driver	2020/7/4 14:56	IntelliJ IDEA Co...	2 KB
Job4_PageRankViewer_Mapper	2020/7/3 15:05	IntelliJ IDEA Co...	1 KB

(1) GraphBuilder:建立网页之间的超连接图，即根据 job3 的输出，对每个人物赋 PageRank 初值

(2) PageRankIlter：迭代计算各个网页的 PageRank 值

(3) PageRankViewer：根据 PageRank 值将结果从大到小输出

整个过程的流程图如下。

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]

GraphBuilder设置初始值

狄云 100#[戚芳,0.33333|戚长发,0.333333| 卜垣,0.333333]
戚芳 100#[狄云,0.25|戚长发,0.25| 卜垣,0.5]
戚长发 100#[狄云,0.33333|戚芳,0.333333| 卜垣,0.333333]
卜垣 100#[狄云,0.25|戚芳,0.5|戚长发,0.25]

狄云 200#[戚芳,0.33333|戚长发,0.333333| 卜垣,0.333333]
戚芳 100#[狄云,0.25|戚长发,0.25| 卜垣,0.5]
戚长发 300#[狄云,0.33333|戚芳,0.333333| 卜垣,0.333333]
卜垣 50#[狄云,0.25|戚芳,0.5|戚长发,0.25]

PageRankIter多次迭代计算Pagerank值

PageRankViewer对PageRank值排序从大到小输出

300 戚长发
200 狄云
100 戚芳
50 卜垣

2. 源代码说明

Job4_Driver

```
public static void main(String[] args) throws Exception//实现多趟MapReduce处理
{
    if (args.length != 3)
    {
        System.err.println("Job4: <job3_out> <job4_out> <times>");
        System.exit(2);
    }

    int times = Integer.parseInt(args[2]);

    String[] GraphBuilder_args = { args[0], args[1] + "/Data_000" };
    Job4_GraphBuilder_Driver.main(GraphBuilder_args);//运行GraphBuilder

    String[] PageRankIter_args = { "", "", "" };
    for (int i = 0; i < times; i++)
    {
        PageRankIter_args[0] = args[1] + "/Data_" + String.format("%03d", i);
        PageRankIter_args[1] = args[1] + "/Data_" + String.format("%03d", i + 1);
        PageRankIter_args[2] = Integer.toString(i + 1);
        Job4_PageRankIter_Driver.main(PageRankIter_args);//运行PageRankIter
    }

    String[] PageRankViewer_args = { args[1] + "/Data_" + String.format("%03d", times), args[1] + "/FinalRank" };
    Job4_PageRankViewer_Driver.main(PageRankViewer_args);//运行PageRankViewer
}
```

Job4_GraphBuilder_Driver.java

```java
public class Job4_GraphBuilder_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 2)
        {
            System.err.println("Job4-GraphBuilder: <in> <out>");
            System.exit(2);
        }

        Configuration conf = new Configuration();
        Job job4_1 = Job.getInstance(conf, "Job4_GraphBuilder");
        //设定一系列格式
        job4_1.setJarByClass(Job4_GraphBuilder_Driver.class);
        job4_1.setInputFormatClass(KeyValueTextInputFormat.class);
        job4_1.setMapperClass(Job4_GraphBuilder_Mapper.class);
        job4_1.setMapOutputKeyClass(Text.class);
        job4_1.setMapOutputValueClass(Text.class);
        job4_1.setOutputKeyClass(Text.class);
        job4_1.setOutputValueClass(Text.class);
        job4_1.setNumReduceTasks(0);//由于不需要reducer节点，因此将reducer节点数置0以提高运行效率
        FileInputFormat.addInputPath(job4_1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job4_1, new Path(args[1]));
        job4_1.waitForCompletion(true);
    }
}
```

Job4_GraphBuilder_Mapper.java

```java
public class Job4_GraphBuilder_Mapper extends Mapper<Text, Text, Text, Text>
{
    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {
        String newvalue = Job4_Driver.InitValue + "#" + value.toString();//对Job3的每条输出添加PageRank初值作为后续迭代部分的输入
        context.write(key, new Text(newvalue));
    }
}
```

PageRankBean.java

```java
public class PageRankBean implements WritableComparable<PageRankBean>//自定义数据结构，作为job4迭代部分传输的基本结构
{
    private String name;//人物名
    private float rank;//rank值
    private String linklist;//人物的链出表，即job3每条输出的邻接边以及权重
```

Job4_PageRankIter_Driver.java

```java
public class Job4_PageRankIter_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 3)
        {
            System.err.println("Job4-PageRankIter: <in> <out> <cur_time>");
            System.exit(2);
        }

        Configuration conf = new Configuration();
        Job job4_2 = Job.getInstance(conf, "Job4_PageRankIter_" + args[2]);
        //设定一系列格式
        job4_2.setJarByClass(Job4_PageRankIter_Driver.class);
        job4_2.setInputFormatClass(KeyValueTextInputFormat.class);
        job4_2.setMapperClass(Job4_PageRankIter_Mapper.class);
        job4_2.setPartitionerClass(Job4_PageRankIter_Partitioner.class);
        job4_2.setGroupingComparatorClass(Job4_PageRankIter_Comparator.class);
        job4_2.setReducerClass(Job4_PageRankIter_Reducer.class);
        job4_2.setMapOutputKeyClass(PageRankBean.class);
        job4_2.setMapOutputValueClass(NullWritable.class);
        job4_2.setOutputKeyClass(Text.class);
        job4_2.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job4_2, new Path(args[0]));
        FileOutputFormat.setOutputPath(job4_2, new Path(args[1]));
        job4_2.waitForCompletion(true);
    }
}
```

Job4_PageRankIter_Mapper.java

传输两种结构数据：<URL,linklist>以及<u,val>。

```java
public class Job4_PageRankIter_Mapper extends Mapper<Text, Text, PageRankBean, NullWritable>
{
    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {
        PageRankBean pr_linklist = new PageRankBean();
        pr_linklist.setName(key.toString());//设置人物名
        pr_linklist.setLinklist(value.toString().split("#")[1]);//设置该人物的链接信息
        context.write(pr_linklist, NullWritable.get());//传递该组链接信息

        float cur_rank = Float.parseFloat(value.toString().split("#")[0]);
        String[] linklist = value.toString().split("#")[1].split("\\|");//根据符号"|"将一整条链接记录分开并逐个传输
        for (int i = 0; i < linklist.length; i++)
        {
            String name = linklist[i].split(",")[0];
            float prob = Float.parseFloat(linklist[i].split(",")[1]);

            PageRankBean pr_rank = new PageRankBean();
            pr_rank.setName(name);//设置人物名
            pr_rank.setRank(cur_rank * prob);//设置链入网页贡献的PageRank值
            context.write(pr_rank, NullWritable.get());
        }
    }
}
```

Job4_PageRankIter_Partitioner.java

分配 Reducer 节点

```
public class Job4_PageRankIter_Partitioner extends HashPartitioner<PageRankBean, NullWritable>
{
    @Override
    public int getPartition(PageRankBean key, NullWritable value, int numReduceTasks)
    {
        return (key.getName().hashCode() & Integer.MAX_VALUE) % numReduceTasks;//根据key中的name信息的哈希码将数据传到对应的reducer节点
    }
}
```

Job4_PageRankIter_Comparator.java

将一个 Reducer 中的多条人名相同的信息在一遍 reduce 循环中处理。

```
public class Job4_PageRankIter_Comparator extends WritableComparator
{
    protected Job4_PageRankIter_Comparator() { super(PageRankBean.class, true); }

    @Override
    public int compare(WritableComparable a, WritableComparable b)
    {
        PageRankBean pa = (PageRankBean) a;
        PageRankBean pb = (PageRankBean) b;
        return pa.getName().compareTo(pb.getName());//使一个reducer节点的数据中name相同的进入同一遍循环进行迭代操作
    }
}
```

Job4_PageRankIter_ Reducer.java

计算每个节点新的 PageRank 值。

```
public class Job4_PageRankIter_Reducer extends Reducer<PageRankBean, NullWritable, Text, Text>
{
    @Override
    public void reduce(PageRankBean key, Iterable<NullWritable> values, Context context)
            throws IOException, InterruptedException
    {
        String newkey = "", linklist = "";
        float cur_rank = 0;
        Iterator<NullWritable> iter = values.iterator();
        while (iter.hasNext())
        {
            iter.next();
            newkey = key.getName();
            if (key.getRank() == -1)//如果读出PageRank值为-1，则说明是<URL,url_list>型数据
            {
                linklist = key.getLinklist();
            }
            else
            {
                cur_rank += key.getRank();//将该人物的链入节点对其贡献的PageRank值进行累加
            }
        }
        float d = 0.85f;//设置d为0.85
        cur_rank = d * cur_rank + (1 - d) * Job4_Driver.InitValue;//将贡献的PageRank值的和与常数相加得出该人物的新PageRank值
        String newvalue = Float.toString(cur_rank) + "#" + linklist;//将value置为new_rank#url_list格式
        context.write(new Text(newkey), new Text(newvalue));//输出<URL,nwerank#url_list>
    }
}
```

Job4_PageRankViewer_Driver.java

```java
public class Job4_PageRankViewer_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 2)
        {
            System.err.println("Job4-PageRankViewer: <in> <out>");
            System.exit(2);
        }

        Configuration conf = new Configuration();
        Job job4_3 = Job.getInstance(conf, "Job4_PageRankViewer");
        //设定一系列格式
        job4_3.setJarByClass(Job4_PageRankViewer_Driver.class);
        job4_3.setInputFormatClass(KeyValueTextInputFormat.class);
        job4_3.setMapperClass(Job4_PageRankViewer_Mapper.class);
        job4_3.setMapOutputKeyClass(DecFloatWritable.class);
        job4_3.setMapOutputValueClass(Text.class);
        job4_3.setOutputKeyClass(DecFloatWritable.class);
        job4_3.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job4_3, new Path(args[0]));
        FileOutputFormat.setOutputPath(job4_3, new Path(args[1]));
        job4_3.waitForCompletion(true);
    }
}
```

DecFloatWritable.java

```java
public class DecFloatWritable extends FloatWritable
{
    public DecFloatWritable() { super(); }

    public DecFloatWritable(float f)
    {
        super();
        super.set(f);
    }

    @Override//重载key的比较函数
    public int compareTo(FloatWritable f)
    {
        return -super.compareTo(f);//取负实现从大到小排序
    }
}
```

Job4_PageRankIter_Mapper.java

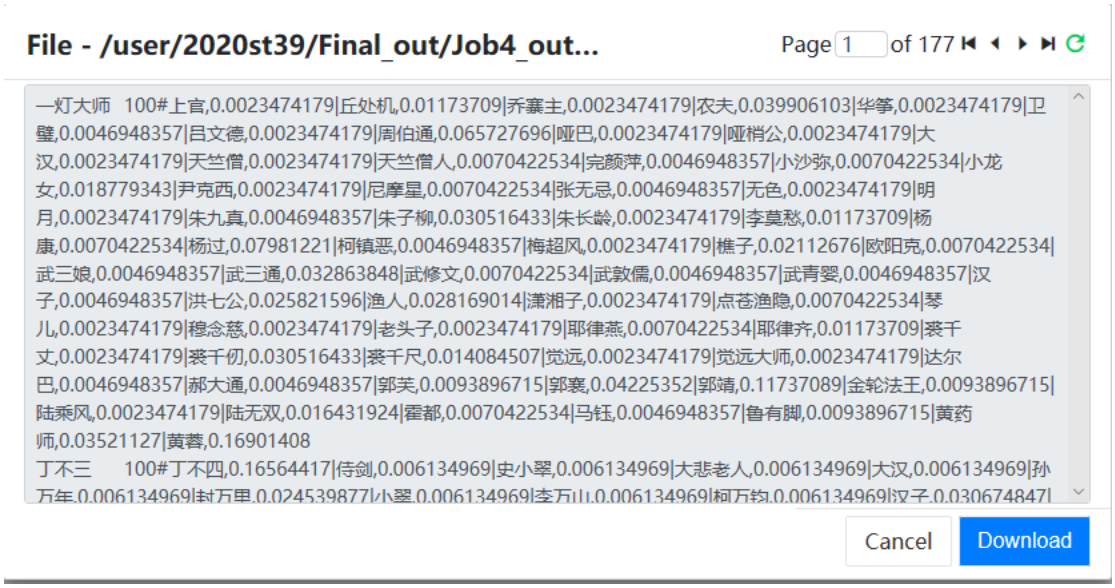
```java
public class Job4_PageRankViewer_Mapper extends Mapper<Text, Text, DecFloatWritable, Text>
{
    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {
        DecFloatWritable newkey = new DecFloatWritable(Float.parseFloat(value.toString().split("#")[0]));//将PageRank值置入自定义DecFloatWritable结构实现从大到小排序
        context.write(newkey, key);//将PageRank值作为新key，输入的key作为新value输出
    }
}
```

（二） 程序运行和结果说明

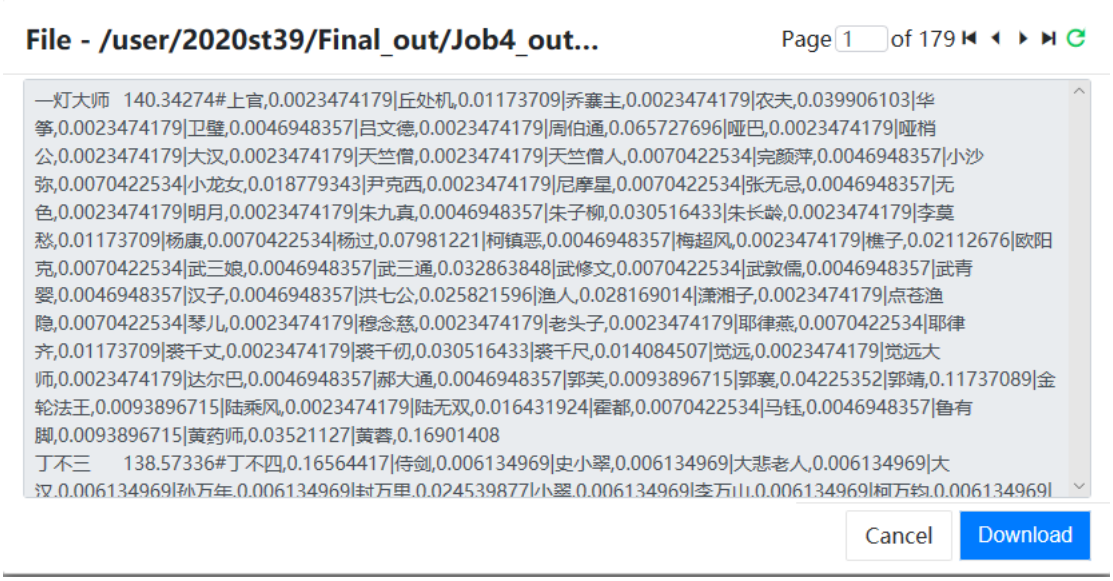
1. 输出结果

输出文件在 HDFS 上的路径：/user/2020st39/Final_out/Job4_out

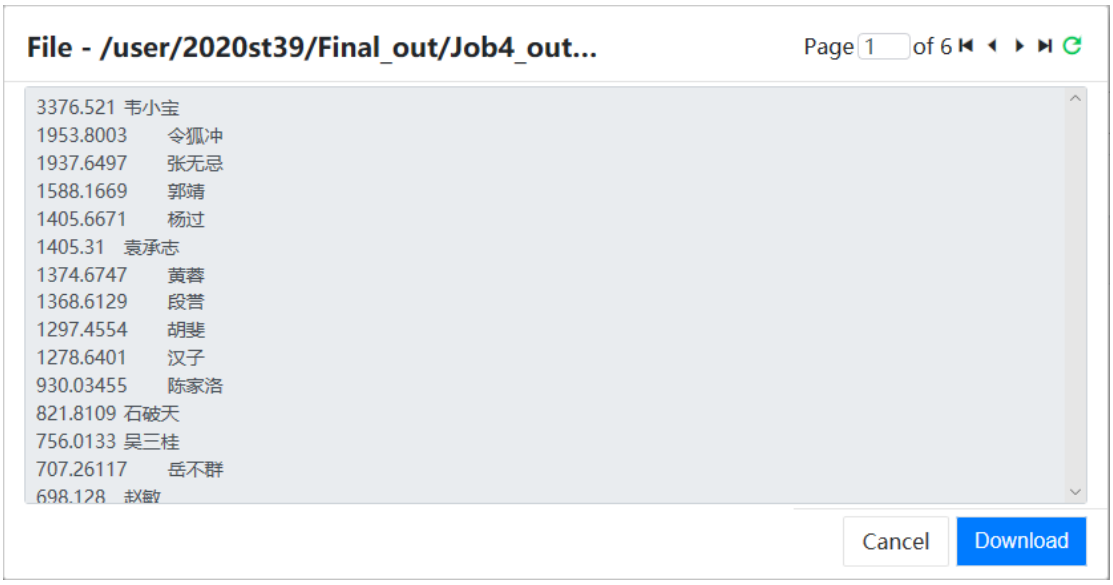
GraphBuilder 结果 Data_000



PageRankIter 最后结果 Data_030



PageRankViewer 结果 FinalRank



2. WebUI 执行报告

GraphBuilder 执行记录

MapReduce Job job_1572597966684_20863

Job Overview	
Job Name:	Job4_GraphBuilder
User Name:	2020st39
Queue:	root.team39
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Jul 28 17:37:53 CST 2020
Started:	Tue Jul 28 17:37:18 CST 2020
Finished:	Tue Jul 28 17:37:24 CST 2020
Elapsed:	5sec
Diagnostics:	
Average Map Time	3sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jul 28 17:37:14 CST 2020	slave014:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	0	0

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

PageRankIter 第一次迭代执行记录

Logged in as: dr.who

MapReduce Job job_1572597966684_20864

Job Overview	
Job Name:	Job4_PageRankIter_1
User Name:	2020st39
Queue:	root.team39
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Jul 28 17:38:06 CST 2020
Started:	Tue Jul 28 17:37:59 CST 2020
Finished:	Tue Jul 28 17:38:11 CST 2020
Elapsed:	11sec
Diagnostics:	
Average Map Time	3sec
Average Shuffle Time	0sec
Average Merge Time	0sec
Average Reduce Time	2sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jul 28 17:37:56 CST 2020	slave020:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

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

PageRankViewer 执行记录

Logged in as: dr.who

MapReduce Job job_1572597966684_20894

Job Overview	
Job Name:	Job4_PageRankViewer
User Name:	2020st39
Queue:	root.team39
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Jul 28 17:47:32 CST 2020
Started:	Tue Jul 28 17:47:35 CST 2020
Finished:	Tue Jul 28 17:47:46 CST 2020
Elapsed:	11sec
Diagnostics:	
Average Map Time	2sec
Average Shuffle Time	2sec
Average Merge Time	0sec
Average Reduce Time	0sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jul 28 17:47:31 CST 2020	slave002:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

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

五．任务五 在人物关系图上的标签传播

（一） 任务设计

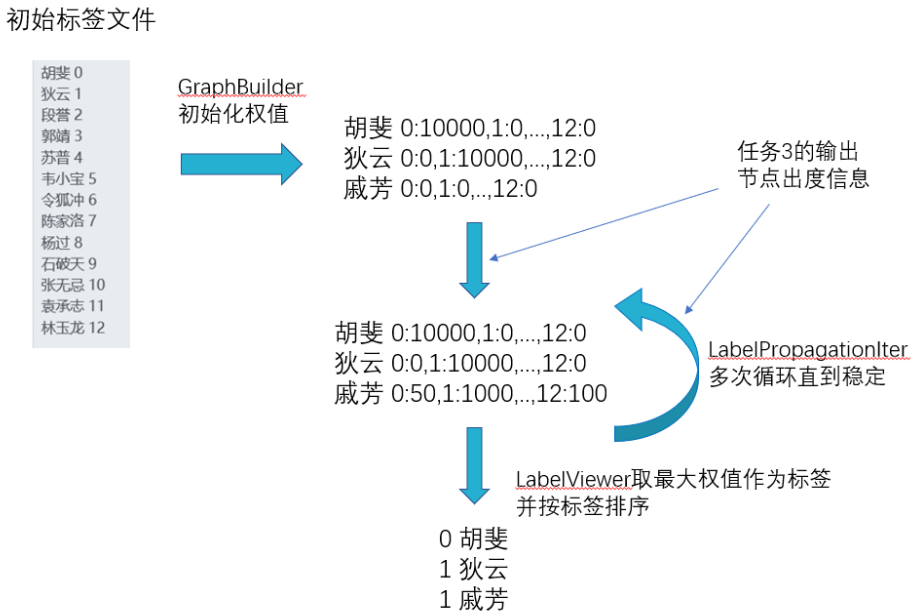
1. 设计思路

与 Job4 类似，将代码分为三个部分：

Job5_Driver	2020/7/11 0:42	IntelliJ IDEA Co...	2 KB
Job5_GraphBuilder_Driver	2020/7/9 16:32	IntelliJ IDEA Co...	2 KB
Job5_GraphBuilder_Mapper	2020/7/9 16:36	IntelliJ IDEA Co...	2 KB
Job5_LabelPropagationIter_Comp...	2020/7/9 18:21	IntelliJ IDEA Co...	1 KB
Job5_LabelPropagationIter_Driver	2020/7/9 22:05	IntelliJ IDEA Co...	2 KB
Job5_LabelPropagationIter_Mapper	2020/7/9 18:16	IntelliJ IDEA Co...	3 KB
Job5_LabelPropagationIter_Partitio...	2020/7/9 18:19	IntelliJ IDEA Co...	1 KB
Job5_LabelPropagationIter_Reducer	2020/7/9 23:54	IntelliJ IDEA Co...	3 KB
Job5_LabelViewer_Driver	2020/7/9 19:49	IntelliJ IDEA Co...	2 KB
Job5_LabelViewer_Mapper	2020/7/9 19:45	IntelliJ IDEA Co...	1 KB
LabelPropagationBean	2020/7/9 17:00	IntelliJ IDEA Co...	2 KB

(1) GraphBuilder:初始化每个节点在各个标签下的权值
(2) LabelPropagationIter：多轮计算每个节点在各个标签下的权值
(3) LabelViewer：权值确定后，选取权值最高的标签为角色标签

整个过程的流程图如下（下例中，人物戚芳标签未知）。



2. 源代码说明

Job5_Driver

```
public class Job5_Driver
{
    public static int label_num = 13;

    public static void main(String[] args) throws Exception//实现多趟MapReduce处理
    {
        if (args.length != 4)
        {
            System.err.println("Job4: <job3_out> <job5_out> <labelled_data> <times>");
            System.exit(2);
        }

        int times = Integer.parseInt(args[3]);

        String[] GraphBuilder_args = { args[0], args[2], args[1] + "/Data_000" };
        Job5_GraphBuilder_Driver.main(GraphBuilder_args);//运行GraphBuilder

        String[] LabelPropagationIter_args = { "", "", args[0] + "/part-r-00000", args[2], "" };
        for (int i = 0; i < times; i++)
        {
            LabelPropagationIter_args[0] = args[1] + "/Data_" + String.format("%03d", i);
            LabelPropagationIter_args[1] = args[1] + "/Data_" + String.format("%03d", i + 1);
            LabelPropagationIter_args[4] = Integer.toString(i + 1);
            Job5_LabelPropagationIter_Driver.main(LabelPropagationIter_args);//多趟运行LabelPropagationIter
        }

        String[] LabelViewer_args = { args[1] + "/Data_" + String.format("%03d", times), args[1] + "/FinalLabel" };
        Job5_LabelViewer_Driver.main(LabelViewer_args);//运行LabelViewer
    }
}
```

Job5_GraphBuilder_Driver.java

```java
public class Job5_GraphBuilder_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 3)
        {
            System.err.println("Job5-GraphBuilder: <job3_out> <labelled_data> <out>");
            System.exit(2);
        }

        Configuration conf = new Configuration();
        Job job5_1 = Job.getInstance(conf, "Job5_GraphBuilder");
        job5_1.setJarByClass(Job5_GraphBuilder_Driver.class);
        job5_1.addCacheFile(new Path(args[1]).toUri());
        job5_1.setInputFormatClass(KeyValueTextInputFormat.class);
        job5_1.setMapperClass(Job5_GraphBuilder_Mapper.class);
        job5_1.setMapOutputKeyClass(Text.class);
        job5_1.setMapOutputValueClass(Text.class);
        job5_1.setOutputKeyClass(Text.class);
        job5_1.setOutputValueClass(Text.class);
        job5_1.setNumReduceTasks(0);
        FileInputFormat.addInputPath(job5_1, new Path(args[0]));
        FileOutputFormat.setOutputPath(job5_1, new Path(args[2]));
        job5_1.waitForCompletion(true);
    }
}
```

Job5_GraphBuilder_Mapper.java

对初始标签角色赋 10000 的初始权值。

```java
public class Job5_GraphBuilder_Mapper extends Mapper<Text, Text, Text, Text>
{
    Map<String, Integer> labels = new HashMap<String, Integer>();

    @Override
    public void setup(Context context) throws IOException, InterruptedException//读入初始标签文本，其中有13个人物已被打好0-12标签，分别对应金庸各个小说中的主角
    {...}

    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {
        String name = key.toString();
        int[] init_value = new int[Job5_Driver.label_num];
        if (labels.get(name) != null)
        {
            init_value[labels.get(name)] = 10000;//将初始标签人物在其对应标签下的权值设为10000，取10000是因为其他人物权值很小，防止写入文件时出现小数截断问题导致比较结果不精准
        }

        String newvalue = new String();
        for (int i = 0; i < Job5_Driver.label_num; i++)
        {
            newvalue = newvalue + Integer.toString(i) + ":" + Integer.toString(init_value[i]) + ",";//初始化每个节点在各个标签下的权值
        }
        newvalue = newvalue.substring(0, newvalue.length() - 1);
        context.write(key, new Text(newvalue));
    }
}
```

LabelPropagationBean.java

```java
public class LabelPropagationBean implements WritableComparable<LabelPropagationBean>//自定义数据结构，作为job5迭代部分传输的基本结构
{
    private String name;//人物名称
    private int label_id;//标签id
    private float prob;//该标签下的权值
```

Job5_LabelPropagatonIter_Driver.java

```java
public class Job5_LabelPropagationIter_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 5)
        {
            System.err.println("Job5-LabelPropagationIter: <in> <out> <link_matrix> <labelled_data> <cur_time>");
            System.exit(2);
        }

        Configuration conf = new Configuration();
        Job job5_2 = Job.getInstance(conf, "Job5_LabelPropagationIter_" + args[4]);
        job5_2.setJarByClass(Job5_LabelPropagationIter_Driver.class);
        job5_2.addCacheFile(new Path(args[2]).toUri());
        job5_2.addCacheFile(new Path(args[3]).toUri());
        job5_2.setInputFormatClass(KeyValueTextInputFormat.class);
        job5_2.setMapperClass(Job5_LabelPropagationIter_Mapper.class);
        job5_2.setPartitionerClass(Job5_LabelPropagationIter_Partitioner.class);
        job5_2.setGroupingComparatorClass(Job5_LabelPropagationIter_Comparator.class);
        job5_2.setReducerClass(Job5_LabelPropagationIter_Reducer.class);
        job5_2.setMapOutputKeyClass(LabelPropagationBean.class);
        job5_2.setMapOutputValueClass(NullWritable.class);
        job5_2.setOutputKeyClass(Text.class);
        job5_2.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job5_2, new Path(args[0]));
        FileOutputFormat.setOutputPath(job5_2, new Path(args[1]));
        job5_2.waitForCompletion(true);
    }
}
```

Job5_LabelPropagatonIter_Mapper.java

对 Job3 的每组人物关系信息，计算每个节点对其每个链出节点的各个标签所贡献的权值。

```java
public class Job5_LabelPropagationIter_Mapper extends Mapper<Text, Text, LabelPropagationBean, NullWritable>
{
    HashMap<String, HashMap<String, Float>> links = new HashMap<String, HashMap<String, Float>>();

    @Override
    public void setup(Context context) throws IOException, InterruptedException//将Job3的输出进行处理，拆分为(from,(toName,prob))的形式存在links中，(toName,prob)存放在weight中
    {...}

    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {
        float[] cur_prob = new float[Job5_Driver.label_num];
        String[] probs = value.toString().split(",");
        for (int i = 0; i < probs.length; i++)
        {
            int label_id = Integer.parseInt(probs[i].split(":")[0]);//获取标签id
            float prob = Float.parseFloat(probs[i].split(":")[1]);
            cur_prob[label_id] = prob;//获取当前标签下的权值
        }

        String name = key.toString();
        HashMap<String, Float> weight = links.get(name);
        for (int i = 0; i < Job5_Driver.label_num; i++)
        {
            for (Map.Entry<String, Float> entry : weight.entrySet())
            {
                LabelPropagationBean lp = new LabelPropagationBean();
                lp.setName(entry.getKey());//设置出节点人名
                lp.setLabel(i);//设置标签id
                lp.setProb(cur_prob[i] * entry.getValue());//设置链入节点对该节点该标签下贡献的权值
                context.write(lp, NullWritable.get());
            }
        }
    }
}
```

Job5_LabelPropagatonIter_Partitioner.java

分配 Reducer 节点

```java
public class Job5_LabelPropagationIter_Partitioner extends HashPartitioner<LabelPropagationBean, NullWritable>
{
    @Override
    public int getPartition(LabelPropagationBean key, NullWritable value, int numReduceTasks)
    {
        return (key.getName().hashCode() & Integer.MAX_VALUE) % numReduceTasks;
    }
}
```

Job5_LabelPropagatonIter_Comparator.java

将一个 Reducer 中的多条人名相同的信息在一遍 reduce 循环中处理。

```java
public class Job5_LabelPropagationIter_Comparator extends WritableComparator
{
    protected Job5_LabelPropagationIter_Comparator() { super(LabelPropagationBean.class, true); }

    @Override
    public int compare(WritableComparable a, WritableComparable b)
    {
        LabelPropagationBean la = (LabelPropagationBean) a;
        LabelPropagationBean lb = (LabelPropagationBean) b;
        return la.getName().compareTo(lb.getName());
    }
}
```

Job5_LabelPropagatonIter_ Reducer.java

对同一人物的标签下的多组权值加和得到一次迭代后的结果。

```java
public class Job5_LabelPropagationIter_Reducer extends Reducer<LabelPropagationBean, NullWritable, Text, Text>
{
    Map<String, Integer> labels = new HashMap<String, Integer>();

    @Override
    public void setup(Context context) throws IOException, InterruptedException//读入初始13个标签人物的信息
    {...}

    @Override
    public void reduce(LabelPropagationBean key, Iterable<NullWritable> values, Context context)
            throws IOException, InterruptedException
    {
        String newkey = "", newvalue = "";
        float[] cur_prob = new float[Job5_Driver.label_num];
        Iterator<NullWritable> iter = values.iterator();
        while (iter.hasNext())//对每个人物的每个标签下的权值做加和处理
        {
            iter.next();
            newkey = key.getName();
            cur_prob[key.getLabel()] += key.getProb();
        }

        if (labels.get(newkey) != null)//初始标签人物的权值不能发生变化，否则会导致迭代的不正确
        {
            for (int i = 0; i < Job5_Driver.label_num; i++)
            {
                cur_prob[i] = 0;
            }
            cur_prob[labels.get(newkey)] = 10000;
        }
        //输出新处理完后的结果
        for (int i = 0; i < Job5_Driver.label_num; i++)
        {
            newvalue = newvalue + Integer.toString(i) + ":" + Float.toString(cur_prob[i]) + ",";
        }
        newvalue = newvalue.substring(0, newvalue.length() - 1);
        context.write(new Text(newkey), new Text(newvalue));
    }
}
```

Job5_LabelViewer_Driver.java

```java
public class Job5_LabelViewer_Driver
{
    public static void main(String[] args) throws Exception
    {
        if (args.length != 2)
        {
            System.err.println("Job5-LabelViewer: <in> <out>");
            System.exit(2);
        }

        Configuration conf = new Configuration();
        Job job5_3 = Job.getInstance(conf, "Job5_LabelViewer");
        job5_3.setJarByClass(Job5_LabelViewer_Driver.class);
        job5_3.setInputFormatClass(KeyValueTextInputFormat.class);
        job5_3.setMapperClass(Job5_LabelViewer_Mapper.class);
        job5_3.setMapOutputKeyClass(IntWritable.class);
        job5_3.setMapOutputValueClass(Text.class);
        job5_3.setOutputKeyClass(IntWritable.class);
        job5_3.setOutputValueClass(Text.class);
        FileInputFormat.addInputPath(job5_3, new Path(args[0]));
        FileOutputFormat.setOutputPath(job5_3, new Path(args[1]));
        job5_3.waitForCompletion(true);
    }
}
```

Job5_LabelViewer_Mapper.java

找出每个节点权值最大的标签作为该节点标签。

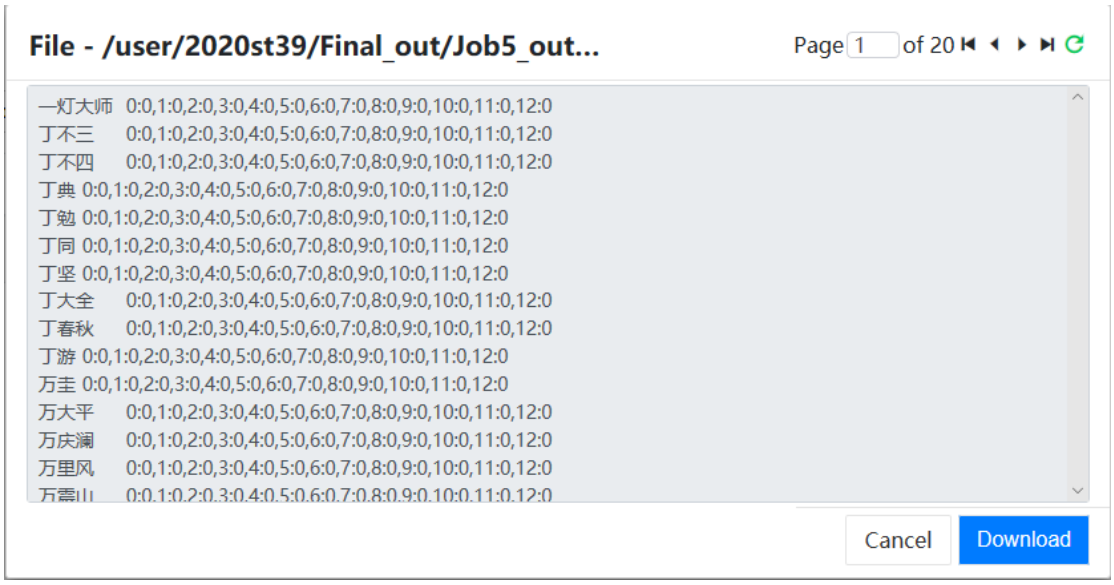
```java
public class Job5_LabelViewer_Mapper extends Mapper<Text, Text, IntWritable, Text>
{
    @Override
    public void map(Text key, Text value, Context context) throws IOException, InterruptedException
    {
        float[] cur_prob = new float[Job5_Driver.label_num];
        String[] probs = value.toString().split(",");
        for (int i = 0; i < probs.length; i++)//读入迭代处理完后的结果
        {
            int label_id = Integer.parseInt(probs[i].split(":")[0]);
            float prob = Float.parseFloat(probs[i].split(":")[1]);
            cur_prob[label_id] = prob;
        }

        int max_id = 0;
        float max = 0;
        for (int i = 0; i < Job5_Driver.label_num; i++)//对每个节点，寻找权值最大的标签作为该节点的标签
        {
            if (cur_prob[i] > max)
            {
                max = cur_prob[i];
                max_id = i;
            }
        }
        context.write(new IntWritable(max_id), key);
    }
}
```

（二） 程序运行和结果说明

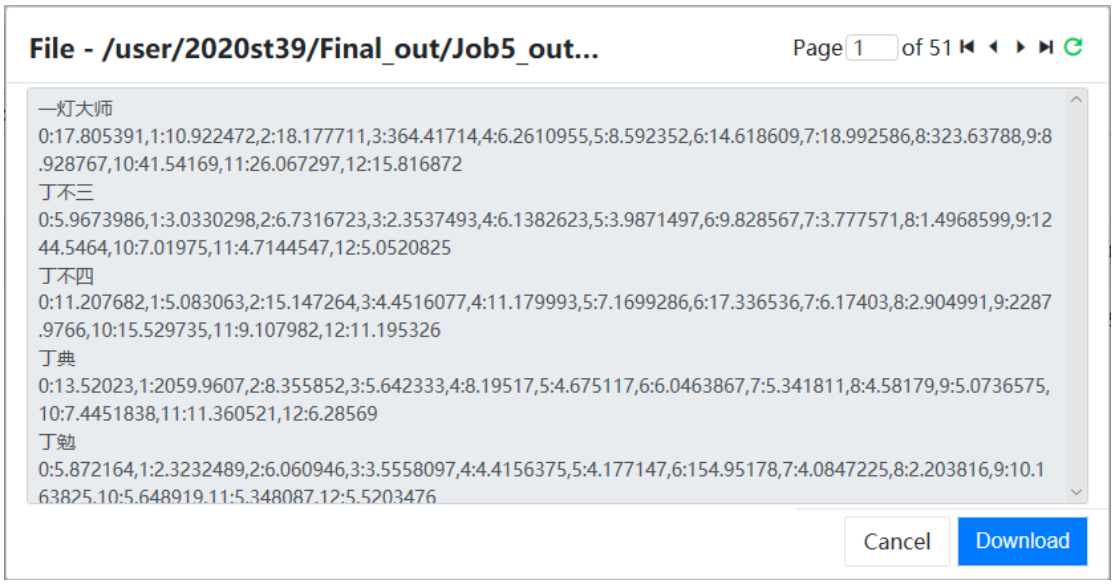
1. 输出结果

输出文件在 HDFS 上的路径：/user/2020st39/Final_out/Job4_out

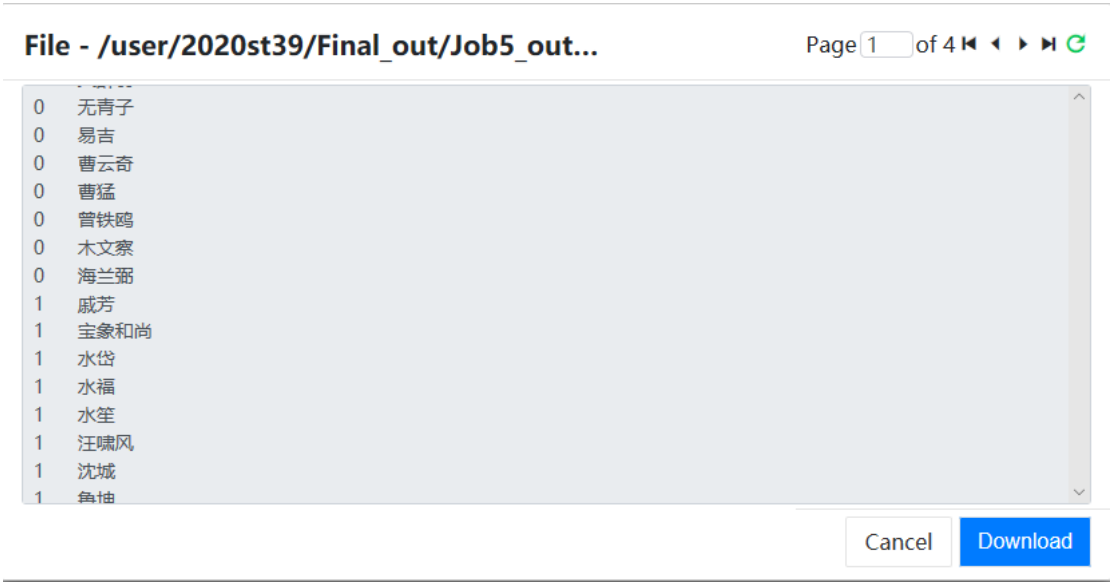
GraphBuilder 结果 Data_000



LabelPropagatonIter 最后结果 Data_015

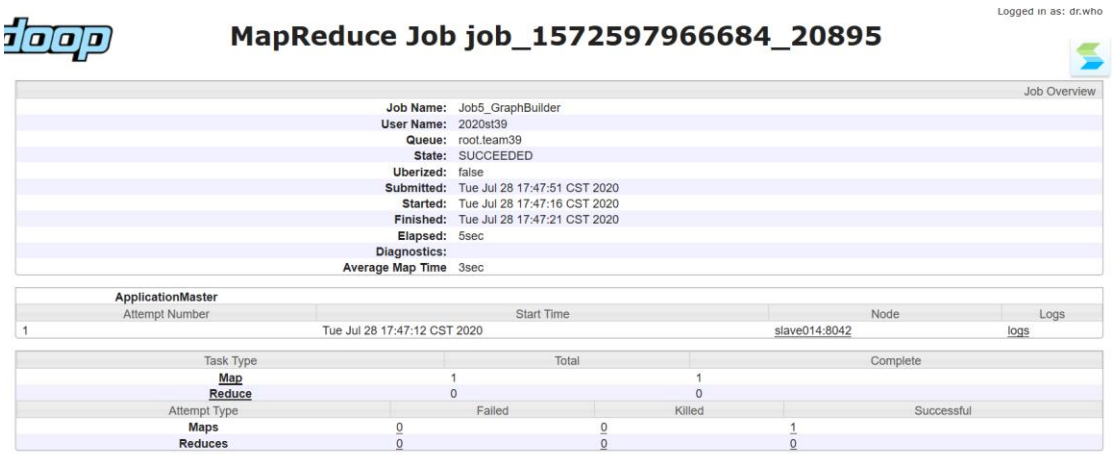


LabelViewer 结果 FinalLabel



2. WebUI 执行报告

GraphBuilder 执行记录



LabelPropagatonIter 第一次迭代执行记录

Logged in as: dr.who

MapReduce Job job_1572597966684_20896

Job Overview

Job Name:	Job5_LabelPropagationIter_1
User Name:	2020st39
Queue:	root.team39
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Jul 28 17:48:04 CST 2020
Started:	Tue Jul 28 17:48:06 CST 2020
Finished:	Tue Jul 28 17:48:23 CST 2020
Elapsed:	16sec
Diagnostics:	
Average Map Time	7sec
Average Shuffle Time	-15sec
Average Merge Time	0sec
Average Reduce Time	19sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jul 28 17:48:02 CST 2020	slave002:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

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

LabelViewer 执行记录

Logged in as: dr.who

MapReduce Job job_1572597966684_20911

Job Overview

Job Name:	Job5_LabelViewer
User Name:	2020st39
Queue:	root.team39
State:	SUCCEEDED
Uberized:	false
Submitted:	Tue Jul 28 17:54:01 CST 2020
Started:	Tue Jul 28 17:54:08 CST 2020
Finished:	Tue Jul 28 17:54:19 CST 2020
Elapsed:	11sec
Diagnostics:	
Average Map Time	2sec
Average Shuffle Time	-11sec
Average Merge Time	0sec
Average Reduce Time	14sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Tue Jul 28 17:54:04 CST 2020	slave005:8042	logs

Task Type	Total	Complete
Map	1	1
Reduce	1	1

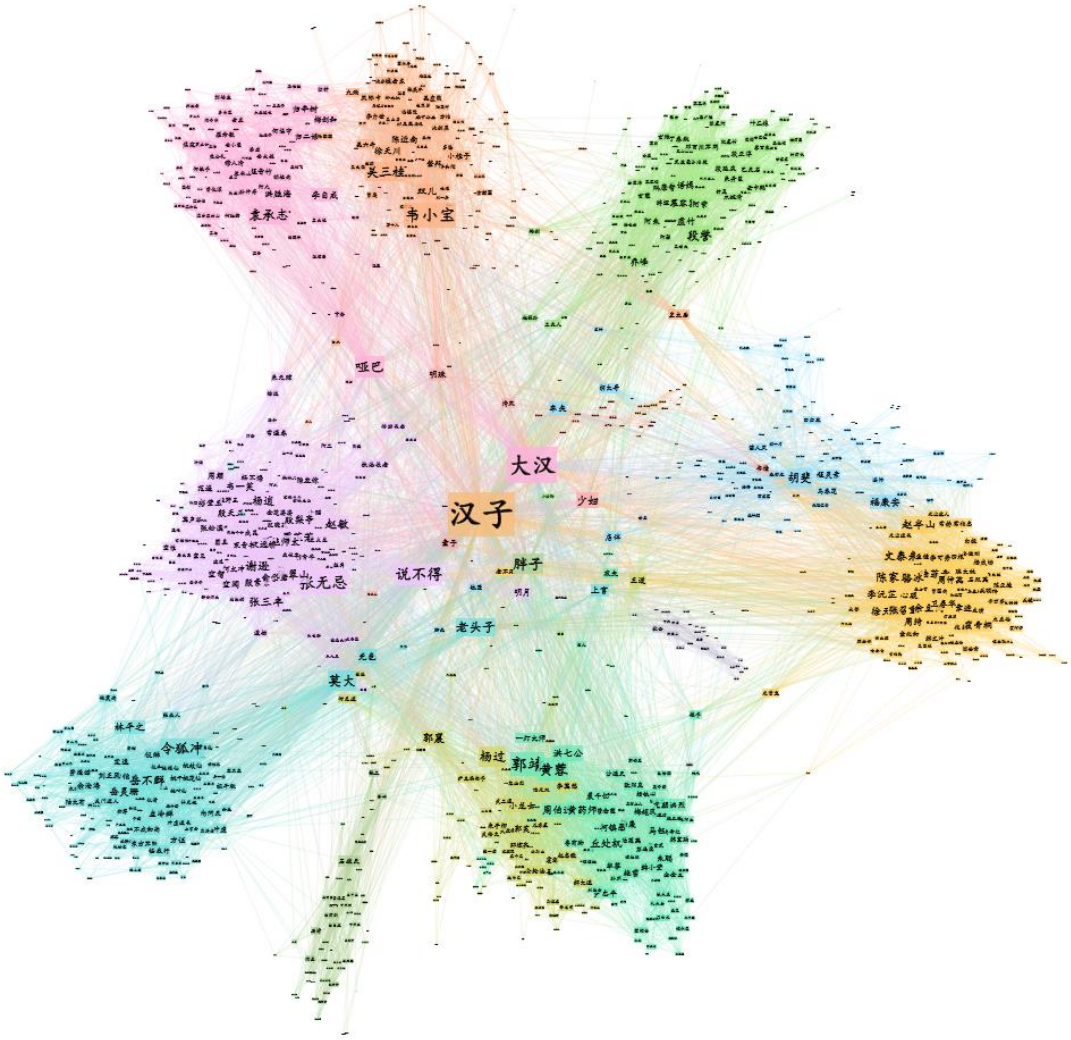
Attempt Type	Failed	Killed	Successful
Maps	0	0	1
Reduces	0	0	1

（三） 数据可视化（Gephi 实现）

将数据导入 Gephi 后，先采用 Force Atlas 布局进行社区发现，然后用标签调整布局使大部分标签清晰可见而不重叠。

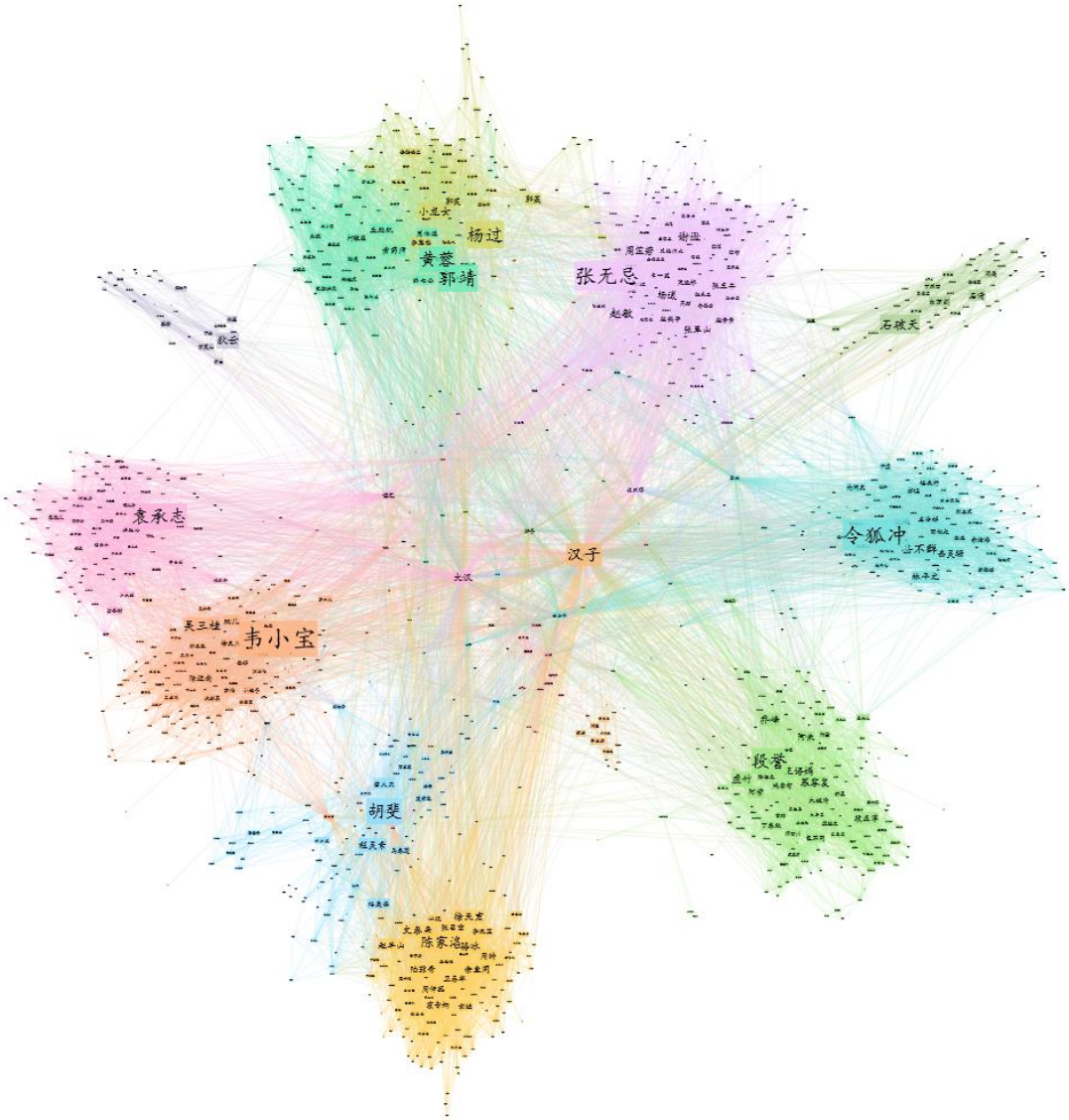
以下两个结果的标签颜色均由标签传播结果决定。

标签大小由节点度数决定



标签大小由节点度数决定是实验手册上的要求，但是我们经过实验发现，这样的设计会使多本书中出现的、普通的名字较为突出，从而降低每本书主角的重要程度。根据这一观点，我们还引入了标签大小由 PageRank 决定的设计。

标签大小由 PageRank 决定



这里的结果使每本书的主角更加突出，弱化了一些很多场合出现过又不怎么重要的角色姓名，明显优于之前的结果。

六．任务六 分析结果整理

这一任务中，有关于任务四、五的结果整理已包含在之前的描述中，此处给出链接，不再赘述。

任务四结果排序　　任务五标签排序

七．运行参数和时间说明

具体的 jar 包运行参数见 README 文件。

经过多次的试验和比较，我们发现，PageRank 算法的迭代次数在 30 次时，结果已经收敛到比较好了，具体而言，与前一次结果排名的差异数量约在 0.3%，与前一次结果的数值差异也在可接受的范围内。

而 LabelPropagaton 算法的迭代次数则是 12-15 次为宜。具体而言，在 12 次之后标签结果已经收敛，而到 15 次时与前一次结果的数值差异就可以忽略了。

根据上述的迭代次数，在集群上总体的运行时间大致在 18-20 分钟（从任务一到任务五）。

八．实验亮点与创新

1. 试图一些任务的 Reducer 数量置为 0 以求更高的效率，多次试验后将有加速效果的置 0 语句保留，累计缩短运行时间 30s 以上。
2. 任务四、五中，使用了自定义的数据类型作为 Mapper 和 Reducer 之间信息传递的载体，增加了代码的逻辑性和可读性。
3. 任务四采取了 PageRank 的随机访问模型，由于设置的初始值是 100 且最开始用的是课件上的公式进行迭代，即使迭代次数超过 60 次时收敛效果都不尽如人意。于是我们改进了迭代公式，使得初始值不会影响迭代次数，且使公式更能切合随机访问的物理模型。改进的迭代公式如下（其中 T1~Tn 为 A 的入度节点）。

$$PR(A) = (1-d) * InitValue + d(\frac{PR(T_1)}{L(T_1)} + \ldots + \frac{PR(T_n)}{L(T_n)})$$

九．小组成员分工

学号	姓名	分工
171860662	山越	代码实现
171860663	马少聪	数据测试和实验报告撰写
171860664	谢鹏飞	代码逻辑优化和实验报告撰写
171860681	冯旭晨	展示 PPT 制作和少量代码实现