

MapReduce 课程设计之金庸的江湖

161220126 王森 161220108 侍林天 161180168 杨丽鹤

Contents

1 概述	3
2 实验分工	4
3 采用 Hadoop 实现的实验细节	4
3.1 任务 1 数据预处理	4
3.2 任务 2 特征抽取: 人物同现统计	7
3.3 任务 3 特征处理: 人物关系图构建与特征归一化	9
3.4 任务 4 数据分析: 基于人物关系图的 PageRank 计算	11
3.5 任务 5 数据分析: 在人物关系图上的标签传播	19
3.6 任务 6 结果整理	24
4 采用 Spark 实现的实验细节	26
4.1 任务 1 数据预处理	26
4.2 任务 2 特征抽取: 人物同现统计	27
4.3 任务 3 特征处理: 人物关系图构建与特征归一化	28
4.4 任务 4 数据分析: 基于人物关系图的 PageRank 计算	29
4.5 任务 5 数据分析: 在人物关系图上的标签传播	30
4.6 任务 6 结果整理	31
5 可视化	31
6 程序运行和实验结果说明与分析	33
6.1 每个 job 的输出结果文件截图	33
7 程序性能优化	36
7.1 参数调优与结果分析	36
7.2 HDFS 上的输出路径	37
7.3 Hadoop 上 WebUI 执行报告	38

8 Hadoop 与 Spark 对比	41
8.1 编程语言	41
8.2 编程框架	41
8.3 程序性能分析	41
9 实验总结	41
10 参考文献与致谢	42

1 概述

本实验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。对于原始的金庸小说文本，对其中的人物进行提取，分析人物与人物之间的关系，挖掘各个人物的在小说中的影响力，发现人物中具有重要影响力的大人物（小说的主要角色）。继而以人物为标签，挖掘人物之间的类簇、群体、纽带关系。

为了解决这一问题，分析出以下解决步骤：

1. 通过分词技术，抽取出与人物互动相关的数据，提取出文本中的人物名，而屏蔽掉与人物关系无关的文本内容。
2. 基于单词同现算法，统计人物同现数据。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。同现关系次数越多，则说明两人的关系越密切。
3. 基于归一化算法，根据共现关系，生成人物之间的关系图。人物关系图使用邻接表的形式表示，人物是顶点，人物之间的互动关系是边。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。
4. 根据 PageRank 算法，定量计算出小说中每个人物的影响值，发现小说中的重要角色。
5. 根据 LPA 算法，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图中完成社区发现，挖掘小说中的类簇群体。
6. 对 4 和 5 的结果进行整理，在对整理后的结果进行分析研究，挖掘金庸小说中的人物关系信息。
7. 对得到的结果进行可视化呈现，通过可视化的形式更加直观的展现挖掘出的规律、知识和信息。

本实验采用 Hadoop 框架和 Spark 框架各实现一遍，进而对两种框架程序进行性能分析，比较两个框架的优劣。对于实验中用到的算法可对参数进行调整，探寻参数与结果之间的关系，确定较优参数。

本实验需要掌握和学习的算法、技术包括：Hadoop 编程，Scala 语言的 Spark 编程，单词同现算法，归一化算法，PageRank 算法和 LPA 标签传播算法。

实验用到的开发环境如下：

Hadoop 框架包括：Hadoop-2.7.1、Java-1.7.0_79

Spark 框架包括：Spark-2.4.3、Scala-2.11.12、Java-11.0.3

2 实验分工

职务	姓名	学号	主要工作
队长	王森	161220126	分别在 Hadoop 和 Spark 下完成任务四 PageRank 和任务六 PageRank 结果的排序整理； 统筹报告的布局，完成对应任务报告内容并完善报告其余内容。
组员	侍林天	161220108	分别在 Hadoop 和 Spark 下完成任务五 LPA 和任务六 LPA 结果的整理； BUG 调试与分析，并完成对应任务报告内容。
组员	杨丽鹤	161180168	分别在 Hadoop 和 Spark 下完成任务一分词、任务二人物同现计算和任务三归一化处理； 完成对应任务报告内容。

3 采用 Hadoop 实现的实验细节

3.1 任务 1 数据预处理

3.1.1 设计思路

任务一是对原始小说进行分词并提取其中的人物名，输入为原始未经处理的小说的每一行，每一行对应小说的一个自然段，输出为该行（自然段）中出现的所有人名。

该任务的输入文件路径是一个目录，该目录下共有 15 本金庸的武侠小说，对于每一本小说，程序需要产生一个对应的输出。

相同的人名在同一行可能会出现多次，为了后续任务的方便，在这里每一行输出的人物都互异。

算法伪代码如下：

Algorithm 1 CharacterExtracting

Input: A line in original novel.

Output: All character names in current line.

```
1: function MAP(lineOffset, line)
2:   Read character file to get user-defined dictionary
3:   Use user-defined dictionary to split the line
4:   Extract character names from all splitted words according to user-defined word nature.
5:   Emit(nameList, Null)
6: end function
7:
8: function REDUCE(nameList, Null)
9:   Emit(nameList, Null)
10: end function
```

该任务借助了 Ansj Seg 分词工具，首先读取人物名文件“people_name_list.txt”，将这些人添加到已有的词典中并定义一个特有的词性“CharacterName”，以跟其他词性区分开，这样就形成了用户自定义的词典，应用该词典对小说进行分词，在分得的词中，保留那些词性为“CharacterName”的词，这些词即是人物名。

3.1.2 程序分析

Mapper 类:

需要重写 map 函数，输入的键值对中，key 是行偏移量，value 表示每一行的文本内容，输出键值对中，key 表示当前行提取到的人物名，value 为空 (即 NullWritable)。在 Hadoop 下，通过 Job 的 Configuration 在 Driver 与 Mapper 间传递全局变量人名列表 NameList。并将人名列表中的人物名插入到用户自定义的人名词典中。通过 Ansj Seg 分词工具将每一行进行分词，获取到词性为 CharacterName 的词，同时也过滤掉了其他与人物名不相关的信息。

具体代码如下:

```
class CharacterMapper extends Mapper<Object, Text, Text, NullWritable> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException
    {
        // names存储该行出现的人名
        String names = "";
        // 使用Job的Configuration获取在Driver端读取的人名列表
        String nameList = context.getConfiguration().get("NameList");
        StringTokenizer itr = new StringTokenizer(nameList);
        // 用户自定义的词典
        while (itr.hasMoreTokens()) {
            DicLibrary.insert(DicLibrary.DEFAULT, itr.nextToken(), "CharacterName", 1000);
        }

        // 分词
        Result parse = DicAnalysis.parse(value.toString());
        List<Term> terms = parse.getTerms();
        for (Term term : terms) {
            // 保留词性为"CharacterName"的词，并且这些词应该在该行还没有出现过（保证输出的人物互异）
            if (term.getNatureStr().equals("CharacterName") && !names.contains(term.getName())) {
                names += term.getName() + " ";
            }
        }

        if (!names.equals("")) {
            context.write(new Text(names), NullWritable.get());
        }
    }
}
```

Reducer 类:

需要重写 reduce 函数，在 reduce 阶段无需对接收到的键值对做任何处理，只需发送即可。

Main 函数：

需要设置 job 的配置信息，通过 Configuration 从 Driver 向 Mapper 传递全局变量 nameList。具体实现时，从人名列表文件中读取小说的所有人物名，并将其存储为一个字符串类型，字符串中的人物名用空格隔开，这样方便后面使用时恢复出所有的人物名，再以字符串的形式通过 Job 的 Configuration 传递。

该人物需要遍历所给的每一个小说文件，逐一处理获取人物名的分词结果。其中采用 HDFS 文件系统，根据用户指定的文件目录，获取该目录下的所有文件状态，再逐一获取小说文本。

该任务要求每一个小说对应一个输出，并且用户指定的输入路径是一个目录，目录下有所有的小说，因此在 main 函数中需要对该目录下的所有文件进行遍历并用上面的 MapReduce 框架进行处理，main 函数中的主要代码如下：

```
public class CharacterExtracting {
    public static void main(String []args) throws Exception {
        // 一些Hadoop的初始化语句，此处省去

        // 从人名列表文件中读取小说的所有人物名，并将其存储为一个字符串类型
        // 字符串中的人物名用空格隔开，这样方便后面使用时恢复出所有的人物名
        // 这些人物名以字符串的形式通过Job的Configuration在Driver和Mapper间传递
        String nameFile = "../data/people_name_list.txt";
        CharacterLoader loader = new CharacterLoader();
        String allNames = loader.load(nameFile);

        // HDFS文件系统
        FileSystem fs = FileSystem.get(URI.create("hdfs://master01:9000"), conf);
        // 根据用户指定的文件目录，获取该目录下的所有文件状态
        FileStatus[] fileStatuses = fs.listStatus(new Path(otherArgs[0]));
        if (!fs.exists(new Path(otherArgs[1]))) {
            fs.mkdirs(new Path(otherArgs[1]));
        }

        // 对于每一个小说文件，分别处理
        for (int i = 0; i < fileStatuses.length; i++) {
            // 一些对于Job和输入输出类型的设置，此处略去
            job.waitForCompletion(true);
        }

        // 假设用户给出的输出目录是/output，每个小说都需要有一个输出
        // 它们是串行处理的并且HDFS文件系统规定输出的目录一定已经不能存在，因此不能直接输出在/output目录中
```

```

// 做法是，首先新建一个目录/output（该目录如果已经存在不会有影响）
// 之后，对于每一本小说，输出文件都输出到/output的一个子目录下，这样每本小说一个子目录
// 最后在程序都输出完成时，再将每个子目录中的文件移至/output目录下，并删除/output下刚刚新建的子目录
for (int i = 0; i < fileStatuses.length; i++) {
    fs.rename(new Path(otherArgs[1] + "/" + fileStatuses[i].getPath().getName() +
        "/part-r-00000"),
        new Path(otherArgs[1] + "/" + fileStatuses[i].getPath().getName().replace(".txt", "")));
    fs.delete(new Path(otherArgs[1] + "/" + fileStatuses[i].getPath().getName()), true);
}
}
}

```

3.2 任务 2 特征抽取：人物同现统计

3.2.1 设计思路

该任务的输入是任务一中得到的人物名，输出是人物之间的同现次数。

此处对于人物同现的定义是，如果两人在原文中同一段落中出现，则认为发生了一次同现关系。但这个定义不是固定的，也可以定义其他的同现范围。

任务的输入是一行人物名，人物之间用空格隔开。大致的思路是：

1) 在 Map 阶段，遍历一行中 n 个人物名所构成的 $\frac{n(n-1)}{2}$ 个人物同现组合，对于每一个组合，输出一个键值对，键值对的 Key 为两个同现的人物名，Value 为 1，表示在此段同现了一次。

2) 在 Reduce 阶段，对于相同 Key 的所有 Values 进行累加，得到总共的同现次数。

算法的伪代码如下：

Algorithm 2 CooccurrenceCounting

Input: A List of character names.

Output: Cooccurrence number of any two characters

```

1: function MAP(lineOffset, line)
2:   Split the line to obtain nameList
3:   Construct all tuples <name1, name2> from nameList.
4:   Emit(< name1, name2 >, 1)
5: end function
6:
7: function REDUCE(< name1, name2 >, values)
8:   Emit(< name1, name2 >, values[0] + ... + values[n])
9: end function

```

3.2.2 程序分析

Mapper 类：

需要重写 map 函数，输入的键值对中，key 表示行偏移量，value 表示每行的内容。而每一行的内容为当前行的人物名列表，即这些人物名同现。因此根据人物名列表构造“人物同现对”，两两组合，分别以其中同现的

两个人物名作为 key，数字 1 作为 value 表示这两个组合同现了一次，构成键值对发送。假设本行有 n 个人物名，则共需发送 $\frac{n(n-1)}{2}$ 个键值对。

具体代码如下：

```
class CooccurrenceMapper extends Mapper<Object, Text, Text, IntWritable> {
    private String cooccurrence;

    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        // 从人物名字符串中获取当前行的人物名列表
        String[] names = value.toString().split(" ");
        // 根据人物名列表构造“人物同现对”，如果当前行中有n个人物，则有n(n-1)/2个“人物同现对”
        for (int i = 0; i < names.length; i++) {
            for (int j = 0; j < names.length; j++) {
                if (i != j) {
                    cooccurrence = "<" + names[i] + "," + names[j] + ">";
                    // 产生的键值对的格式：(<name1, name2>, 1)
                    context.write(new Text(cooccurrence), new IntWritable(1));
                }
            }
        }
    }
}
```

Reducer 类：

需要重写 reduce 函数，根据 map 函数的处理获取到若干同现人物名组合，需要对相同 key(同现组合) 的 value 进行类和，即可获取同现组合的同现次数。

具体代码如下：

```
class CooccurrenceReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
        InterruptedException {
        int sum = 0;
        // 对相同Key的多个Values进行累加，得到总共的同现次数
        for (IntWritable value : values) {
            sum += value.get();
        }
        context.write(key, new IntWritable(sum));
    }
}
```

3.3 任务 3 特征处理：人物关系图构建与特征归一化

3.3.1 设计思路

该任务的输入是任务 2 中的各个人物对的同现次数，根据这些同现关系，可以构建人物之间的关系图。关系图用邻接表表示，方便后面的 PageRank 计算。

该任务的主要目的是将同现次数转化为同现频率，下面举例说明转化过程，假设任务 2 中产生了这样一系列的键值对 (对于任意的 x, y , 都有 $N_{xy} = N_{yx}$, N_{xy} 表示同现次数):

$(\langle a, b \rangle, N_{ab}), (\langle a, c \rangle, N_{ac}), (\langle b, a \rangle, N_{ba}), (\langle b, c \rangle, N_{bc}), (\langle c, a \rangle, N_{ca}), (\langle c, b \rangle, N_{cb})$

上面的这些键值对中的 Value 值都是同现次数，下面根据同现次数计算同现频率 (在关系图中，我们将键值对 $\langle a, b \rangle$ 看成是以 a 为出边, b 为入边):

$$P_{ab} = \frac{N_{ab}}{N_{ab} + N_{ac}}$$

$$P_{ac} = \frac{N_{ac}}{N_{ab} + N_{ac}}$$

....

则最终 Key 为 a 的输出是:

$a \quad [b, P_{ab} \mid c, P_{ac}]$

为了计算同现频率，需要有某个节点的出度之和 (此处的“出度之和”定义为“所有出边的权重之和”，“权重”即为同现次数)，因此我们需要对原来的键值对进行一些改变，原来的 Key 是 $\langle \text{name1}, \text{name2} \rangle$ ，为了能在 Reduce 阶段获得 name1 的出度之和，需要在 Map 阶段将键值对变为 $(\text{name1}, \langle \text{name2}, \text{number} \rangle)$ ，也就是将 Key 变成了单独的 name1 ，这样在 Reduce 阶段时 name1 所有的出边都会出现在 name1 的 Values 中，也就很容易计算出度之和了，同时，因为所有的出边都在 Values 中，也很容易计算每一条出边权重的比例。

算法伪代码如下:

Algorithm 3 RelationBuilding

Input: Cooccurrence number of any two characters.

Output: Cooccurrence frequency.

```
1: function MAP(lineOffset, line)
2:   Split name1, name2, number2 from line.
3:   Emit(name1,  $\langle \text{name2}, \text{number2} \rangle$ )
4: end function
5:
6: function REDUCE(name1, [ $\langle \text{name2}, \text{number2} \rangle$ ,  $\langle \text{name3}, \text{number3} \rangle$ , ...])
7:   total  $\leftarrow$  the sum of all numbers in values
8:   Emit(name1, [ $\langle \text{name2}, \text{number2}/\text{total} \rangle$ ,  $\langle \text{name3}, \text{number3}/\text{total} \rangle$ , ...])
9: end function
```

3.3.2 程序分析

Mapper 类:

需要重写 map 函数，输入的键值对中，key 是行偏移量，value 表示每一行的文本内容，即两两组合的人物名以及他们的同现次数。在函数中通过 String 类的 split 函数分割获取到同现的两个人物名以及同现次数。然后改变原来键值对的结构，以同现组合中第一个人物名作为 key，第二个人物名和同现次数拼接构成 value，构成新的键值对进行发送。

具体代码如下：

```
class RelationMapper extends Mapper<Object, Text, Text, Text> {
    public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
        // value的形式是: <name1, name2> cooccurrence_number
        // 去除不必要的符号
        String str = value.toString().replace("<", "").replace(">", "");
        // names是同现的两个人物名，中间用逗号隔开
        String names = str.split("\\t")[0];
        // number是同现次数
        int number = Integer.parseInt(str.split("\\t")[1]);
        String[] nameList = names.split(",");
        // 改变原来键值对的结构，主要是为了改变Key，方便后面计算出度权重之和
        context.write(new Text(nameList[0]), new Text(nameList[1] + " " + number));
    }
}
```

Reducer 类:

需要重写 reduce 函数，根据 map 函数我们已经获取到对于一个人物名，与他同现的其他人物名以及对应同现次数，所以只需通过迭代遍历即可计算出同现频率。由于迭代器只能遍历一次，所以通过 ArrayList 先存储 values 中的数据（包括人物名和对应同现次数），方便后续遍历。

```
class RelationReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
        InterruptedException {
        String newValue = "[";
        double sum = 0.0;

        // values是Iterable类型，只能迭代一次
        // 因此，为了后面计算出边权重的比例，我们需要保存每一次的同现次数
        ArrayList<Double> numbersCache = new ArrayList<>();
        ArrayList<String> namesCache = new ArrayList<>();
        // 计算出度权重之和
        for (Text value : values) {
            double number = Double.parseDouble(value.toString().split(" ")[1]);
```

```

        sum += number;

        numbersCache.add(number);

        namesCache.add(value.toString().split(" ")[0]);
    }

    // 计算同现频率
    for (int i = 0; i < numbersCache.size(); i++) {
        double percent = numbersCache.get(i) / sum;
        if (i != 0) {
            newValue += "|";
        }
        newValue += namesCache.get(i) + "," + String.format("%.5f", percent);
    }

    newValue += "];";
    context.write(key, new Text(newValue));
}
}

```

3.4 任务 4 数据分析：基于人物关系图的 PageRank 计算

3.4.1 设计思路

任务四的输入为任务三的输出，每一行的数据格式如下：

Source [*Target*₁, *Weight*₁ | *Target*₂, *Weight*₂ | ... | *Target*_{*n*}, *Weight*_{*n*}]

基于人物关系图的 PageRank 计算包括三个阶段：

阶段一：GraghBuilder

算法伪代码如下：

Algorithm 4 GraghBuilder

Input: 人物关系图构建与特征归一化

Output: 人物初始化 Rank 值与关系权重网络图

```

1: function MAP(nid n, Node links)
2:    $N \leftarrow Configuration.get(totalNumber)$ 
3:    $initRank \leftarrow 1/N$ 
4:    $postings \leftarrow initRank + links$ 
5:    $Emit(nid\ n, postings)$ 
6: end function
7:
8: function REDUCE(nid n, postings)
9:    $Emit(nid\ n, postings)$ 
10: end function

```

初始化每个人物节点的 rank 值，并构建人物关系网络图，网络图是出度图，包括指向的人物以及基于同现

次数归一化结果的权重值。

在 Mapper 中，初始化每个人物节点的 rank 值为 $1/N$ ，其中 N 为总的人物个数，通过 Job 的 Configuration 从 Driver 向 Mapper 传递参数 N 。然后将初始化的 rank 值和关系网络图合并，作为键值对的 value 值，源人物名作为 key 值，发送出去。

在 Reducer 中，不做任何处理，直接将得到的 key 和 value 构成的键值对发送出去。

阶段二：PageRank

阶段一获取到了每个人物的初始化 rank 值以及每个人物的出度表。阶段二迭代计算每个人物的 rank 值，直到 rank 值收敛或迭代预定次数。

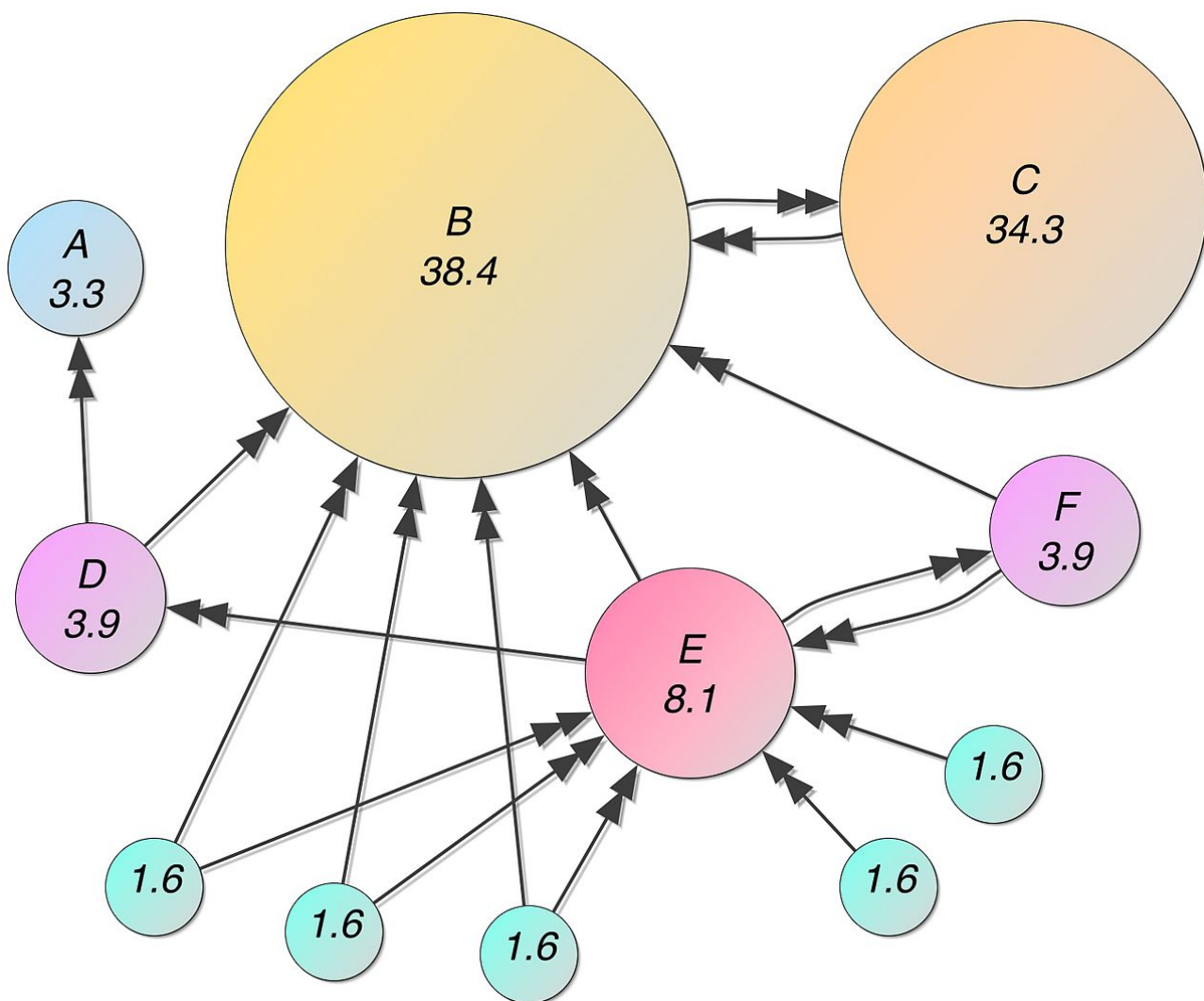


Figure 1: PageRank 示意图

每一轮迭代时的操作是一样的，伪代码如下：

在 Mapper 中，发送两种不同的键值对，由于迭代过程中需要维护人物关系网络图，所以第一种键值对的 key 为源人物名，value 为该源人物名的出度表。另外，每一个源人物的 rank 值都会按照他的出度表及相应权

Algorithm 5 PageRank

Input: 人物旧的 Rank 值与关系权重网络图

Output: 人物新的 Rank 值与关系权重网络图

```
1: function MAP(nid n, postings)
2:   rank  $\leftarrow$  postings.rank
3:   links  $\leftarrow$  postings.links
4:   Emit(nid n, links)
5:   for each node  $\in$  links do
6:     weight  $\leftarrow$  node.weight
7:     target  $\leftarrow$  node.target
8:     Emit(nid target, weight * rank)
9:   end for
10: end function
11:
12: function REDUCE(nid n, List [p1, p2, ..., pn])
13:   postings  $\leftarrow$   $\phi$ 
14:   newRank  $\leftarrow$  0
15:   N  $\leftarrow$  Configuration.get(totalNumber)
16:   d  $\leftarrow$  Configuration.get(d)
17:   for each p  $\in$  [p1, p2, ..., pn] do
18:     if IsNode(p) then
19:       postings.links  $\leftarrow$  p
20:     else
21:       newRank  $\leftarrow$  newRank + p
22:     end if
23:   end for
24:   postings.rank  $\leftarrow$  newRank * d + (1 - d) / N
25:   Emit(nid n, postings)
26: end function
```

重将影响传递给他所指向的人物，所以第二种键值对的 key 为目标人物名，value 为源人物对该目标任务的影响 = 源人物 rank 值乘以出度权重。

值得一提的是，在原始 pagerank 算法中，源网页的 rank 值是平均传递给目标网页的。而在本实验中，由于任务三已经求得每个人物指向其他人物的边的权重，因此源网页的 rank 值不会平均传递给她的出度列表，而是根据权重传递。

在 Reducer 中，收到的键值对的 key 是人物名，相同 key 的 value 通过迭代器传递给 reduce 函数。这些 value 中包括指向他的人物传递给他的 rank 值，也报过当前人物的出度表。所以需要作判断，识别两种 value。对于第一种 value，只需将它们累和，即可得到总的 rank 值。对于第二种 value，需要将其与新得到的 rank 值合并，得到最终发送的键值对的 value。当通过累和求得总 rank 值时，根据以下随机游走公式，计算得到新的 rank 值：

$$NewRank(A) = \frac{1-d}{N} + d * \sum_{x \in T} OldRank(x) * weight(x \rightarrow A)$$

其中 T 表示有边指向人物 A 的人物集合。 $\frac{1-d}{N}$ 即为随机游走的部分。

最终发送的键值对中，key 为输入时的 key 即人物名，value 为由新得到的 rank 值和该人物的出度列表组成的字符串。

需要注意的是，Reducer 的输出中包含出度列表，不仅是为了在迭代中维护图结构，也是为了保持 Reducer 的输出与 Mapper 的输入格式相同，这样前一轮迭代的结果就可以作为下一轮迭代的输入，使得迭代继续下去。

阶段三：GetResult

阶段二得到了迭代一定轮数的 rank 值，但由于需要维护图结构，因此输出的每一行中，还包括每个人物的出度列表。因而在阶段三中提取出每个人物的 rank 值，并输出到文件。伪代码如下：

Algorithm 6 GetResult

Input: 人物的 Rank 值与关系权重网络图

Output: 人物的 Rank 值

```
1: function MAP(nid n, postings)
2:   rank  $\leftarrow$  postings.rank
3:   Emit(nid n, rank)
4: end function
5:
6: function REDUCE(nid n, rank)
7:   Emit(nid n, rank)
8: end function
```

在 Mapper 中，提取出每一个人物的 rank 值，并发送键值对。其中 key 为人物名，value 为 rank 值。

在 Reducer 中，不需要做其他处理，直接将收到的键值对进行发送。

3.4.2 程序分析

程序共分为四个模块，第一个是 GraghBuilder 模块，第二个是 PageRank 模块，第三个是 GetResult 模块，第四个是总调度模块，按序调度前三个模块，并传递输入输出参数。

模块一：GraghBuilder

Mapper 类：

需要重载函数 map，输入键值对的 key 为每一行的偏移量，value 为每一行的文本内容。因此要在 map 函数中对每一行的文本内容进行拆分，提取出源人物名和 links。其中人物名即为待发送键值对的 key 值。然后通过 Configuration 获取预先在 main 函数中设定的全局变量 N(人物总数)，继而求得了每个人物的初始 rank 值 $\frac{1}{N}$ 。最后将初始 rank 值与 links 拼接成字符串，两者间用 # 隔开。得到待发送键值对的 value 值。

具体代码如下：

```
class GraghBuilderMapper extends Mapper<Object, Text, Text, Text> {
    private Text keyInfo = new Text(); // 待发送key
    private Text valueInfo = new Text(); // 待发送value
    @Override
    protected void map(Object key, Text value, Context context) throws IOException,
        InterruptedException {
        String name = value.toString().split("\t")[0]; // 人物名
        String relations = value.toString().split("\t")[1]; // 出度表
        int N = context.getConfiguration().getInt("totalNumber", -1); // 获取N
        keyInfo.set(name);
        Double initRank = 1.0/N;           // 初始化rank值
    }
}
```

```

        valueInfo.set(initRank.toString()+"#"+relations);
        context.write(keyInfo, valueInfo);
    }
}

```

Reducer 类:

需要重载函数 reduce，在该函数中无需对键值对做任何处理，直接发送即可。

Main 函数:

需要设置配置信息，指定输入输出路径，还需设置全局参数 N 的值，从 Driver 向 Mapper 和 Reducer 传递。

模块二: PageRank

Mapper 类:

需要重载函数 map，输入的键值对中，key 表示行偏移量，value 表示一行的内容。输出的键值对有两种，第一种 key 是人物名，value 是该人物名的出度表。另一种 key 是受影响的人物名，value 是由指向他的人物对他传递的影响值，按照 rank 乘以边权重的方式计算。首先需要对一行内容进行拆分，根据模块一的输出可以知道，每一行分为两块，分别是人物名和 postings，这两者之间通过 \t 分隔。其中 postings 又由 rank 值和出度表构成，两者通过 # 分隔。分隔出这三个东西可通过 String 类的 split 进行分割，分割出出度列表后通过遍历每一个指向的对象及其边权重，逐一发送 rank 值贡献。为了区分两种键值对，对于发出度表的键值对的 value，第一个字符前加上 # 以区分。

具体代码如下:

```

class PageRankMapper extends Mapper<Object, Text, Text, Text> {
    private Text keyInfo = new Text(); // 待发送键值对的key
    private Text valueInfo = new Text(); // 待发送键值对的value
    @Override
    protected void map(Object key, Text value, Context context) throws IOException,
        InterruptedException {
        String name = value.toString().split("\t")[0]; // 人物名
        Double rank = Double.parseDouble(value.toString().split("\t")[1].split("#")[0]); // 旧的rank
        String relations = value.toString().split("\t")[1].split("#")[1]; // 出度列表
        keyInfo.set(name);
        valueInfo.set("#"+relations);
        context.write(keyInfo, valueInfo); // 发送value值为出度表的键值对
        String[] slist = relations.split("\\[\\|\\]") [1].split("\\|"); // 对出度表中每一个对象发送rank值影响
        String target = new String();
        Double weight = 0.0;
        Double res = 0.0;
        for(String s : slist) {

```

```

        target = s.split(",")[0]; // 指向的人物名
        weight = Double.parseDouble(s.split(",")[1]); // 边权重
        keyInfo.set(target);
        res = weight*rank;
        valueInfo.set(res.toString());
        context.write(keyInfo, valueInfo); // 发送键值对, key为受影响人物名, value为造成的影响
    }
}
}

```

Reducer 类:

需要重载函数 reduce, 输入的键值对有两种, 因此需要判断, 判断的方式是判断 value 值第一个字符是否为 #。若是, 则该 value 信息表示出度列表。反之, 则该 value 表示当前人物收到的别的只想他的人物传递的 rank 值贡献。对于第二类 value 值进行类和, 得到总的 rank 值, 与随机游走部分的 rank 值结合得到新的 rank 值。

```

class PageRankReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
        InterruptedException {
        Text valueInfo = new Text();
        String relations = new String();
        double sum = 0.0;
        double d = context.getConfiguration().getDouble("d", -1.0); // 获取全局变量d, 控制随机游走的概率
        int N = context.getConfiguration().getInt("totalNumber", -1); // 获取全局变量N, 总人物个数
        for(Text t : values) {
            String value = t.toString();
            if(value.startsWith("#")) { // 出度表
                relations = value;
            }
            else { // rank
                sum += Double.parseDouble(value);
            }
        }
        sum = sum * d;
        sum = sum + (1-d)/N; // 总贡献值与随机游走结合得到新的rank值
        valueInfo.set(String.format("%.6f", sum)+relations);
        context.write(key, valueInfo);
    }
}

```

Main 函数:

需要设置配置信息, 指定输入输出路径, 还需设置全局参数 d 和 N 的值, 从 Driver 向 Mapper 和 Reducer 传递。还需要设置迭代轮数 (15), 并将前一次迭代的输出路径作为后一次迭代的输入路径。值得一提的是, 用户输入时候的输出路径值是一个路径前缀, 迭代过程中会在这个路径的基础上, 拼接上迭代轮数, 得到真正的输出路径。

对于迭代次数, 起初使用 20 轮, 通过输出结果发现, 在十轮以下时, 每轮迭代都会使得结果有很大改变, 但到十轮以上时, 改变越来越小, 可以观察到结果正逐步逼近一个不变值。但正如逼近理论, 越逼近界限, 所需的迭代次数就越大, 所需时间和内存是十分不划算的, 因此设置 15 为最大迭代轮数, 提前收敛结束, 虽未完全收敛, 但此时相邻迭代结果的变化已然很小。

具体代码如下:

```
public static void main(String[] args) throws Exception{
    Configuration conf = new Configuration();

    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    if (otherArgs.length != 2) {
        System.err.println("Usage: PageRank <in> <out>");
        System.exit(2);
    }
    String input = otherArgs[0];           // 输入目录
    String output = new String();
    int max_iter = 15;
    for(Integer i = 0; i < max_iter; i++) { // 迭代15轮
        output = otherArgs[1]+ i.toString(); // 输出目录
        Job job = new Job(conf, "PageRank");
        job.getConfiguration().setInt("totalNumber", 1288); // 设置全局变量N
        job.getConfiguration().setDouble("d", 0.88); // 设置全局变量d
        job.setJarByClass(PageRank.class);
        job.setMapperClass(PageRankMapper.class);
        job.setPartitionerClass(HashPartitioner.class);
        job.setReducerClass(PageRankReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setNumReduceTasks(5);           // 设置五个Reducer节点
        FileInputFormat.addInputPath(job, new Path(input)); // 设置输入路径
        FileOutputFormat.setOutputPath(job, new Path(output)); // 设置输出路径
        job.waitForCompletion(true); // 等待程序运行完毕
        input = output;
    }
}
```

```
}  
}
```

模块三：GetResult

Mapper 类：

需要重写函数 map，在 map 中，输入的键值对的 key 是行偏移量，value 是每一行的内容，需要提取出模块三输出中的人物名和 rank 值，分别作为键值对的 key 和 value，然后发送。对于输入的每一行内容，有人物名和 postings 构成，两者通过 \t 分隔。其中 postings 又由 rank 值和出度表构成，两者通过 # 分隔，可通过 String 类的 split 分隔提取出人物名和 rank 值。具体代码如下：

```
class GetResultMapper extends Mapper<Object, Text, Text, Text> {  
    private Text keyInfo = new Text(); // 待发送键值对的key  
    private Text valueInfo = new Text(); // 待发送键值对的value  
    @Override  
    protected void map(Object key, Text value, Context context) throws IOException,  
        InterruptedException {  
        String name = value.toString().split("\t")[0]; // 人物名  
        Double rank = Double.parseDouble(value.toString().split("\t")[1].split("#")[0]); // rank值  
        keyInfo.set(name);  
        valueInfo.set(rank.toString());  
        context.write(keyInfo, valueInfo); // key为人名名，value为rank值  
    }  
}
```

Reducer 类：

需要重写函数 reduce，在 reduce 中无需对键值对做任何处理，直接发送即可。

Main 函数：

需要设置配置信息，指定输入输出路径。

模块四：总调度模块

Main 函数：

人物总共有三个模块的运行，每个模块都有输入和输出路径，因而共有四个输入的参数。对这四个参数进行预处理，其中需要对于 PageRank 模块的输出路径进行处理。因为 PageRank 模块的输出路径会在迭代时变化，直接在用户输入的路径上加上迭代轮数作为本次迭代的输出路径，所以最后一个字符不能是/，若是，则需去除/。然后将四个参数分别传给前三个模块，且前一个模块的输出路径为后一个模块的输入路径。值得一提的是，模块三的输出路径是模块二的最后一次迭代的输出路径。

具体代码如下：

```

public static void main(String[] args) {
    if(args.length != 4) { // 输入参数个数为4个
        System.err.println("Usage: task4 <in> <out>");
        System.exit(4);
    }
    if(args[2].charAt(args[2].length()-1) == '/') { // 若模块二输出路径最后以/结尾
        args[2] = args[2].substring(0, args[2].length()-1);
    }
    String[] arg1 = {args[0], args[1]}; // 模块一的输入参数
    String[] arg2 = {args[1], args[2]}; // 模块二的输入参数
    String[] arg3 = {args[2]+"14", args[3]}; // 模块三的输入参数
    try {
        GraghBuilder.main(arg1); // 逐一传入三个模块
        PageRank.main(arg2);
        GetResult.main(arg3);
    }catch(Exception e) {
        e.printStackTrace();
    }
}

```

3.5 任务 5 数据分析：在人物关系图上的标签传播

3.5.1 设计思路

在本实验里，我们拟采用 LPA(Label Propagation Algorithm) 标签传播算法对金庸小说中的人物进行聚类。

LPA 是一种半监督的图分析算法，能够为图中的每一个节点打上标签，对图的顶点进行聚类，从而在一张类似社交网络的图中完成社区发现。

Algorithm 7 Label propagation algorithm

Input: Network $G = (V, E)$

Output: Label $C(x)$ for each node

- 1: 初始化网络中所有的顶点，对于顶点 x ，其标签 $C_0(x) = x$
 - 2: **repeat**
 - 3: 每一个顶点 x 向邻接的顶点发送自己标签 $C_t(x)$
 - 4: 每一个顶点 x 从邻接顶点 $\{x_{n1}, x_{n2}, \dots, x_{nk}\}$ 选择出现次数最多的标签作为自己新的标签 $C_{t+1}(x)$
 - 5: **until** 每一个顶点的标签就是其邻接顶点中出现次数最多的标签
-

对于金庸小说中的人物关系分析而言，我们的网络可以看成一个每条边带权值的有向图，因此我们的算法也要在此基础上做一些改变。首先我们还是对每一个顶点进行初始化，然后每一个顶点根据自己的出边向邻接顶点发送自己的标签。最后每个顶点从自己收到的标签里取权值和最大的标签作为自己新的标签。当每一个顶点的标签不再改变时，算法终止。

金庸人物关系图上标签传播的 Mapreduce 实现分为两个阶段。

Algorithm 8 Updated label propagation algorithm

Input: Network $G = (V, E)$

Output: Label $C(x)$ for each node

- 1: 初始化网络中所有的顶点, 对于顶点 x , 其标签 $C_0(x) = x$
 - 2: **repeat**
 - 3: 每一个顶点 x 向自己出边邻接的顶点 v_i 发送自己标签 $C_t(x)$, 和出边的权值 w_{xv_i}
 - 4: 每一个顶点 x 从邻接顶点 $\{x_{n1}, x_{n2}, \dots, x_{nk}\}$ 选择权值和最大的标签作为自己新的标签 $C_{t+1}(x)$
 - 5: **until** 每一个顶点的标签不再改变
-

第一个阶段, 初始化每一个顶点, 将每一个节点的人物姓名作为该节点的标签。

Map: 一行一行分析实验三的输出结果, 直接将每个顶点的人名作为每个顶点的标签, 输出 $\langle \text{人名}, (\text{标签}, [\text{邻接表}]) \rangle$

Reduce: 不做任何处理直接输出 $\langle \text{人名}, (\text{标签}, [\text{邻接表}]) \rangle$

Algorithm 9 Phase 1: Initialize Label

Input: 人物关系网络权重图

Output: 每个人物的标签和人物关系网络权重图

- 1: **function** MAP(nid n , node N)
 - 2: $N.\text{label} \leftarrow N.\text{name}$
 - 3: emit(nid n , N)
 - 4: **end function**
 - 5:
 - 6: **function** REDUCE(nid n , node N)
 - 7: emit(nid n , N)
 - 8: **end function**
-

第二个阶段, 迭代更新每个顶点的标签, 直到每个顶点的标签不再改变或者迭代一定轮数。

Map: 对上阶段的产生的 $\langle \text{人名}, (\text{标签}, [\text{邻接表}]) \rangle$ 输出两种键值对:

1. 对于邻接表里的每一个顶点 u , 输出 $\langle u, (\text{标签}, \text{出边权值}) \rangle$
2. 为了保持迭代过程, 需要传递每一个顶点的邻接表 $\langle \text{人名}, [\text{邻接表}] \rangle$

Reduce: 对 Map 输出的 $\langle \text{人名}, [\text{邻接表}] \rangle$ 和多个 $\langle u, (\text{标签}, \text{出边权值}) \rangle$ 进行处理

其中 $\langle \text{人名}, [\text{邻接表}] \rangle$ 为当前顶点的邻接表, $\langle u, (\text{标签}, \text{出边权值}) \rangle$ 为邻接顶点传来的标签, 选择权值和最大的标签为新的标签。输出 $\langle \text{人名}, (\text{新标签}, [\text{邻接表}]) \rangle$ 。

LPA 在二部图时会出现震荡, 当迭代轮数超过 10 轮后就会开始震荡, 因此我们只迭代 15 轮。

3.5.2 程序分析

程序分为两个模块, 一个是标签初始化阶段, 一个是标签迭代阶段。

模块一: 标签初始化

Mapper 类:

一行行读入任务三输出的结果 $\langle \text{人名}, \text{邻接表} \rangle$, 然后将人名作为该任务的标签以 $\langle \text{人名}, (\text{标签}, \text{邻接表}) \rangle$ 的形式输出。

Reducer 类:

Algorithm 10 Phase 2: Label Iter

Input: 人物关系网络权重图

Output: 每个人物的标签和人物关系网络权重图

```
1: function MAP(nid n, node N)
2:   emit(nid n, N.ADJACENCYLIST)
3:   for each nodeid m in N.ADJACENCYLIST do
4:     emit(m, N.label and weight(n→m))
5:   end for
6: end function
7:
8: function REDUCE(nid m, [p1, p2, ...])
9:   M ← ∅
10:  H ← HASHMAP
11:  for each s in [p1, p2, ...] do
12:    if ISNODE(s) then
13:      M ← s
14:    else
15:      if H.CONTAINS(s) then
16:        Update (s.label, s.oldWeight + s.newWeight)
17:      else
18:        Insert (s.label, s.weight)
19:      end if
20:    end if
21:  end for
22:  M.label ← The key which has the maximum value in H
23:  emit(nid m, node M)
24: end function
```

直接将输入输出。

```
class InitLPAMapper extends Mapper<Object, Text, Text, Text> {
    @Override
    protected void map(Object key, Text value, Context context) throws IOException,
        InterruptedException {
        String v = value.toString();
        String name = v.split("\\t")[0];
        String list = v.split("\\t")[1];
        context.write(new Text(name), new Text(name + "#" + list));
    }
}

class InitLPAReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
        InterruptedException {
        for (Text v : values) {
            context.write(key, v);
        }
    }
}
```

```

    }
}
}

```

模块二：迭代更新标签

Mapper 类:

输入为模块一的输出，格式为 < 人名,(标签, 邻接表)>。输出两种键值对，一种是 < 人名, 邻接表 >，以维护整个网络的结构；一种是遍历邻接表，向邻接的顶点发送该顶点的标签和对应出边的权值 < 邻接顶点人名, (标签, 权值)>。

```

class LPMapper extends Mapper<Object, Text, Text, Text> {
    private Text keyInfo = new Text();
    private Text valueInfo = new Text();

    @Override
    protected void map(Object key, Text value, Context context) throws IOException,
        InterruptedException {
        String name = value.toString().split("\t")[0]; //获取人名
        String postings = value.toString().split("\t")[1];
        String label = postings.split("#")[0]; //获取标签

        String relations = "#" + postings.split("#")[1];
        keyInfo.set(name);
        valueInfo.set(relations);
        context.write(keyInfo, valueInfo); //发送邻接表维护网络结构

        String[] slist = relations.split("\\[\\]") [1].split("\\|");
        for (String s : slist) { //遍历邻接顶点
            String target = s.split(",")[0]; //获取邻接顶点人名
            String weight = s.split(",")[1]; //获取对应出边权重
            keyInfo.set(target);
            valueInfo.set(label + "|" + weight);
            context.write(keyInfo, valueInfo); //向邻接顶点发送标签和权值
        }
    }
}
}

```

Reducer 类:

有两种输入，一种是 < 人名, 邻接表 >；一种是 < 人名,(标签, 权值)>。对于第一种输入，直接保存邻接表维护网络结构。对于第二种输入，借助 HashMap 这种数据结构，如果收到的该标签还没有出现过，以标签为

key, 权值为 value 插入哈希表中, 如果已经出现过, 则标签原来对应的 value 加上新的权值。最后取权值和最大的标签为新的标签。

```
class LPReducer extends Reducer<Text, Text, Text, Text> {
    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
        InterruptedException {
        String name = key.toString();
        HashMap<String, Double> neighbor = new HashMap<>();
        String list = "";
        for (Text t : values) {
            String temp = t.toString();
            if (temp.startsWith("#")) {
                list = temp.substring(1); //如果是邻接表, 保存
            } else {
                String n = temp.split("\\|")[0]; //如果是标签和权值
                Double w = Double.parseDouble(temp.split("\\|")[1]); //获取权值
                if (!neighbor.containsKey(n)) { //加入哈希表中
                    neighbor.put(n, w);
                } else {
                    Double m = neighbor.get(n) + w;
                    neighbor.put(n, m);
                }
            }
        }

        String label = "";
        Double max = 0.0;
        for (Map.Entry<String, Double> entry : neighbor.entrySet()) {
            String k = entry.getKey();
            Double v = entry.getValue();
            if (v > max) { //以权值和最大的标签为新的标签
                max = v;
                label = k;
            }
        }

        String res = label + "#" + list;
        context.write(key, new Text(res)); // 输出结果准备下一轮迭代
    }
}
```

Main 函数:

Main 函数主要负责控制迭代计算。本实验通过迭代一定轮数作为迭代的终止条件。第 i 轮以 $inputPath_i$ 为输入, $inputPath_{i+1}$ 为输出。 $inputPath_0$ 为第一个阶段初始化标签的输入。

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
    int maxIteration = 40;
    for (int i = 0; i < maxIteration; i++) {
        Job job = Job.getInstance(conf, "LabelPropagation");
        job.setJarByClass(LabelPropagation.class);
        job.setMapperClass(LPMapper.class);
        job.setReducerClass(LPReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);
        job.setNumReduceTasks(4);

        FileInputFormat.addInputPath(job, new Path(otherArgs[0] + Integer.toString(i)));
        Path path = new Path(otherArgs[1] + Integer.toString(i+1));
        FileSystem fileSystem = path.getFileSystem(conf); // 根据path找到这个文件
        if (fileSystem.exists(path)) {
            fileSystem.delete(path, true); // true的意思是, 就算output有东西, 也一并删除
        }
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1] + Integer.toString(i+1)));
        job.waitForCompletion(true);
    }
}
```

3.6 任务 6 结果整理

3.6.1 对 PageRank 的结果排序

任务四得到了 PageRank 的结果, 但是没有排序。这里需要对结果进一步加工, 按照降序输出到文件。通过利用 Partion 类的排序功能, 帮助我们对 PageRank 结果进行排序。但是需要自定义 DoubleWritable 类, 因为默认的 DoubleWritable 类比较函数是升序排列的, 因此需要改写 DoubleWritable 类的 compareTo 函数, 使得该函数的返回值与原先的返回值相反。自定义的 DoubleWritable 类如下:

```
class GlobalSortDoubleWritable extends DoubleWritable { // 自定义的DoubleWritable类型
    public GlobalSortDoubleWritable() {
        super();
    }
}
```

```

}

public GlobalSortDoubleWritable(Double d) {
    super(d);
}

@Override
public int compareTo(DoubleWritable o) { // 将返回值置反
    if (this.get() == o.get()) {
        return 0;
    } else if (this.get() > o.get()) {
        return -1;
    } else {
        return 1;
    }
}
}

```

Mapper 类:

需要重载 map 函数，输入键值对的 key 值表示行偏移量，value 值表示每一行的文本信息。需要对每一行的文本信息进行拆分，得到人物名和其 rank 值，然后将 rank 值作为 key，人物名作为 value，发送键值对。这样是因为 Partitioner 会按照 key 值排序，所以将 rank 值作为 key 发送。具体代码如下：

```

class PageRankSortMapper extends Mapper<Object, Text, GlobalSortDoubleWritable, Text> {
    @Override
    protected void map(Object key, Text value, Mapper.Context context) throws IOException,
        InterruptedException {
        String name = value.toString().split("\t")[0]; // 人物名
        String rank = value.toString().split("\t")[1]; // rank值
        context.write(new GlobalSortDoubleWritable(Double.parseDouble(rank)), new Text(name));
    }
}

```

HashPartitioner 类:

需要重载 getPartition 函数，由于最终输出的文件为 1 个，因此这里的 numReduceTasks 值为 1(因为在 main 中设置的 reduce 节点个数为 1)。值得一提的是，我们需要通过使用自定义 DoubleWritable 类作为传入的 key 值类型，这样 Partitioner 按照键值对 key 值排序时，就会调用重写的 compareTo 函数，是的排序结果按照降序排列，而非升序。具体代码如下：

```

class PageRankSortPartitioner extends HashPartitioner<GlobalSortDoubleWritable, Text> { //
    自定义DoubleWritable类
    public int getPartition(GlobalSortDoubleWritable key, Text value, int numReduceTasks) {

```

```

        return super.getPartition(key, value, numReduceTasks);
    }
}

```

Reducer 类:

需要重载函数 reduce，在该函数中需要将 key 值和 value 值互换，再发送。因为 map 中为了方便排序，将 key 值设为了 rank 值，而将 value 值设成了人名。因此在 reduce 中需要对 values 迭代器进行遍历 (可能有相同 rank 值的人物名)，逐一发送键值对，发送的键值对的 key 为人名，value 为 rank 值。具体代码如下：

```

class PageRankSortReducer extends Reducer<GlobalSortDoubleWritable, Text, Text, Text>{
    @Override
    protected void reduce(GlobalSortDoubleWritable key, Iterable<Text> values, Context context) throws
        IOException, InterruptedException {
        for(Text t : values) {
            context.write(t, new Text(key.toString())); // 需要将key与value互相调换发送
        }
    }
}

```

3.6.2 对 LPA 的结果整理

map 阶段以人物的标签作为 key，人名为 value 发送键值对；reduce 阶段将同一标签的人物以人名为 key，标签为 value 发送。

4 采用 Spark 实现的实验细节

4.1 任务 1 数据预处理

4.1.1 程序分析

在 Spark 中，与 Hadoop 类似，需要使用 DicAnalysis 包对每一段进行分词，获得分词结果后。需要对分词结果进行筛选，剔除与人物名无关的信息，使用 filter 函数即可达到过滤剔除的效果。具体代码如下：

```

object CharacterExtracting {
    def main(args: Array[String]): Unit = {
        // Spark环境初始化，用户自定义词典，此部分代码省略

        val numPartitions = 1
        val text = sc.textFile(prefix + args(0) + "/" + fileStatuses(i).getPath().getName(),
            numPartitions).map { line =>
            // 分词结果

```

```

val wordList = DicAnalysis.parse(line)
var names = ""
// 对分出的词进行筛选，必须是人名且在该行还未出现过
for(i <- Range(0,wordList.size()) if wordList.get(i).getNatureStr() == "CharacterName" &&
    !names.contains(wordList.get(i).getName())) {
    names += wordList.get(i).getName() + " "
}
names
}.filter(line => line != "") // 某些行可能没有人名，过滤掉空行

text.saveAsTextFile(prefix + args(1) + "/" + fileStatuses(i).getPath().getName())
}
}

```

4.2 任务 2 特征抽取：人物同现统计

4.2.1 程序分析

对于任务一的输出，通过序列化的 map 操作，分别进行分隔字符串、组合同现人物名、统计同现次数的多个 map 操作，再对最后的键值对进行格式化输出即可。具体代码如下：

```

object CooccurrenceCounting {
  def main(args: Array[String]): Unit = {
    val conf = new SparkConf().setAppName("CooccurrenceCounting")
    val sc = new SparkContext(conf)
    val prefix = "hdfs://localhost:9000"

    val numPartitions = 10
    val text = sc.textFile(prefix + args(0)).
      // 分割字符串，并将单词转化为列表
      map(_._split(" ").map(_._trim).toList).
      // 任取2个进行组合，并且都是同现了1次
      flatMap(_._combinations(2)).
      map(_._(1)).
      // 统计总共的同现次数
      reduceByKey(_ + _).
      // 进行格式化输出
      map(x => "<" + x._1(0) + "," + x._1(1) + ">" + "\t" + x._2)

    text.repartition(numPartitions).saveAsTextFile(prefix + args(1))
  }
}

```

```
}  
}
```

4.3 任务 3 特征处理：人物关系图构建与特征归一化

4.3.1 程序分析

对于任务二的输出，计算每一个结点的所有出边权重之和，产生键值对：<name1, totalNumber>，采用 join 方法，根据 Key 合并两个键值对，得到键值对 <name1, (totalNumber, (name2, number))>，计算当前相邻结点 name2 占 name1 所有相邻结点权重的比例，即可得到同现组合所占的归一化权重。具体代码如下：

```
object RelationBuilding {  
  def main(args: Array[String]): Unit = {  
    val conf = new SparkConf().setAppName("RelationBuilding")  
    val sc = new SparkContext(conf)  
    val prefix = "hdfs://localhost:9000"  
  
    val numPartitions = 2  
    val text = sc.textFile(prefix + args(0)).map(line => line.replace("<", "").replace(">", ""))  
    // 计算每一个结点的所有出边权重之和，产生键值对：<name1, totalNumber>  
    val total = text.map(line => (line.split("\t")(0).split(",")(0),  
      line.split("\t")(1).toDouble)).reduceByKey(_ + _)  
    // 产生键值对：<name1, (name2, number)>  
    val relations = text.map(line => (line.split("\t")(0).split(",")(0),  
      (line.split("\t")(0).split(",")(1), line.split("\t")(1).toDouble)))  
    // 根据Key合并上面两个键值对，得到键值对<name1, (totalNumber, (name2, number))>  
    // 因此可以计算当前相邻结点name2占name1所有相邻结点权重的比例  
    val combined = total.join(relations, 2).  
      map(line => (line._1, line._2._2._1 + "," + line._2._2._2 / line._2._1 + "|")).  
      groupByKey().  
      map(line => line._1 + "\t" + line._2.toString().replace("CompactBuffer", "").replace("|",  
        ", ").replace("(", "[").replace(")", "]"))  
  
    combined.repartition(numPartitions).saveAsTextFile(prefix + args(1))  
  }  
}
```

4.4 任务 4 数据分析：基于人物关系图的 PageRank 计算

4.4.1 程序分析

spark 最大的特性在于内存级操作，因而可以将 Hadoop 中难以存入内存的出度表存入静态内存，在迭代中访问即可，无需再迭代中反复发送出度表这一键值对。因此，首先对输入文件的每一行进行拆分，使用一个 map 对一行结果按 \t 拆分得到人物名，并与 rank 初始值 $\frac{1}{N}$ 构成 name+rank 键值对 (初始状态)。然后采用同样的方式，拆分得到 name+ 出度列表的键值对。在 spark 中，这一信息是不变化也不发送的，存储在静态内存中。最后迭代计算 rank 值，轮数依然为 15。

在 map 部分，需要将出度表键值对与贡献值出度表进行合并，接着自定义构造 flatMap，在 flatMap 中对于每一个复合键值对的源人物名，参照其出度表，结合边权发送 rank 贡献值。

在 reduce 部分，采用 reduceByKey，对同一个 key 的 value 们累和即可得到总的 rank 贡献值，再结合随机游走部分得到新的 rank 值并发送。

结尾部分，迭代完成后，对最后一次迭代结果按照 rank 值 (即键值对的 value 排序)，将结果按降序排列。并保存到输出文件中 (通过 repartition 指定输出文件个数为 1 个)。

具体代码如下：

```
import org.apache.spark.SparkContext
import org.apache.spark.SparkConf

object PageRank {
  def main(args: Array[String]) {
    val conf = new SparkConf().setAppName("PageRank") // 配置信息
    val sc = new SparkContext(conf)
    if(args.length != 2) {
      System.err.println("Usage: PageRank <in> <out>")
      System.exit(1)
    }
    // 获取初始化name+rank键值对
    var nameAndRank = sc.textFile(args(0)).map(line => (line.split("\t")(0),1.0/1288))
    // 获取静态出度列表，一直存储在静态内存中(利用了spark特性)
    val relations = sc.textFile(args(0)).map(line =>
      (line.split("\t")(0),line.split("\t")(1).split("\\[|\\]") (1).split("\\|").toSeq))
    val iter_num = 15 // 最大迭代轮数
    for(i <- 1 to iter_num) {
      // 每个人物对其指向的人物发送rank值贡献
      val contribs = nameAndRank.join(relations,2).flatMap {
        case (_, (rank,links)) => links.map(dest => (dest.split(",")(0),
          dest.split(",")(1).toDouble*rank.toDouble))
      }
    }
```

```

// 将rank值贡献累和并与随机游走合并求得新的rank值
nameAndRank = contribs.reduceByKey(_ + _,2).mapValues(0.12/1288 + 0.88 * _)
}

val sorts = nameAndRank.sortBy(_._2,false) // 将输出结果按照rank值降序输出
sorts.repartition(1).saveAsTextFile(args(1)) // 输出到文件，输出文件个数为1
}
}

```

4.5 任务 5 数据分析：在人物关系图上的标签传播

4.5.1 程序分析

Spark 的一大特性即是 RDD，因此我们在 spark 里无需在 Mapreduce 的过程中维护网络的出度表，而是可以将网络结构的出度表读到内存里，如何通过 join 方法与顶点信息自动匹配，形成 < 人名, (标签, 邻接表)> 的形式输出。

程序首先读取任务三的输出，然后使用 map 方法分别生成顶点信息集合 < 人名, 标签 >，和网络结构信息集合 < 人名, 出度表 >。然后在迭代循环里通过 join 函数，生成顶点信息和网络结构信息的内连接。然后使用 map 生成 <(人名, 标签), 出度权重 > 的集合。接着使用 reduceByKey 方法，以 (人名, 标签) 为 key，将同一人物接收到的同一标签的权重进行求和，生成 <(人名, 标签), 权重和 > 的集合。最后使用 reduceByKey 方法以人名为 key，求出权重和最大的标签作为每个人物新的标签，准备下一轮迭代。

使用 sort 函数，以标签为 key 进行排序，使得同一标签的人物排列在一起。

```

object LPA {
  def func(a : String, b : String) : String = { //用于在reduceByKey里求最大值
    if (a.split(",")(1).toDouble > b.split(",")(1).toDouble) {
      return a
    } else {
      return b
    }
  }
}

def main(args: Array[String]) {
  val conf = new SparkConf().setAppName("LPA")
  val sc = new SparkContext(conf)
  if(args.length != 2) {
    System.err.println("Usage: LPA <in> <out>")
    System.exit(1)
  }

  //读取任务三的输出，生成顶点信息
  var nameAndLabel = sc.textFile(args(0)).map(line => (line.split("\t")(0),line.split("\t")(0)))

```

```

//读取任务三的输出，生成网络结构信息
val relations = sc.textFile(args(0)).map(line =>
    (line.split("\t")(0),line.split("\t")(1).split("\\[|\\]") (1).split("\\|").toSeq))
//开始迭代
val iter_num = 10
for(i <- 1 to iter_num) {
    //通过join生成完整信息
    val contribs = nameAndLabel.join(relations,2).flatMap {
        case(_, (label,links)) => links.map(dest => (dest.split(",")(0)+"."+label,
            dest.split(",")(1).toDouble))
    }
    //每个顶点对收到的同一标签进行权值求和
    val merge = contribs.reduceByKey(_+_ , 2)
    //求出权值和最大的标签
    val mergedLabel = merge.map((kk)=>(kk._1.split(",")(0),
        kk._1.split(",")(1)+"."+kk._2.toString) )
    nameAndLabel = mergedLabel.reduceByKey(func(_,_),2).map(vv=>(vv._1, vv._2.split(",")(0)));
}
//对标签进行排序，使得同一标签的人物排列在一起
val sorts = nameAndLabel.sortBy(_._2,false)
//输出结果
sorts.repartition(1).saveAsTextFile(args(1))
}
}

```

4.6 任务 6 结果整理

本任务已在任务四和任务五的 spark 实现中完成。

5 可视化

使用 gephi 软件，将任务三和任务六的输出作为输入，得到点数据和边数据，通过 python 脚本过滤小簇孤岛点，然后生成点和边的 csv 文件，导入到 gephi 中，并调整布局和相关参数得到如下可视化结果：

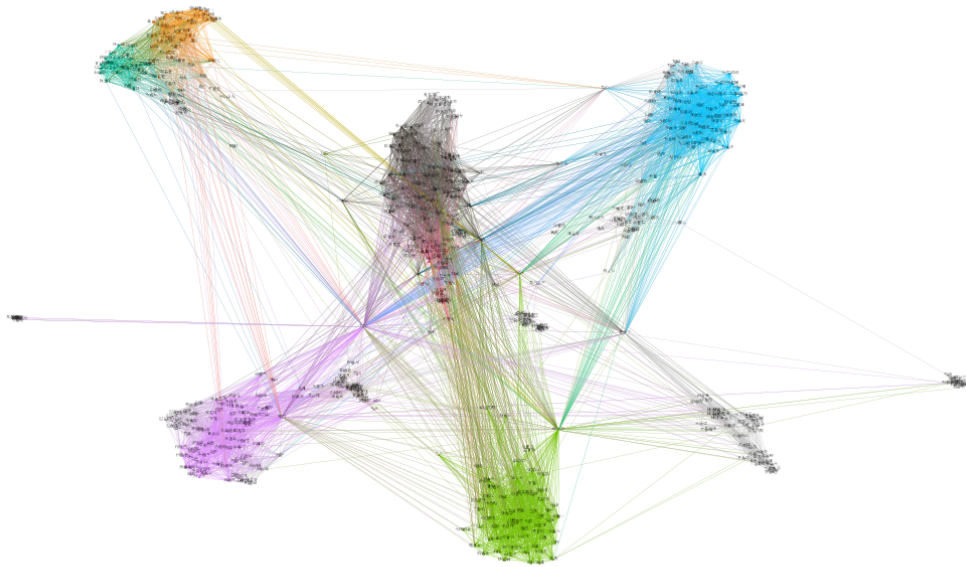


Figure 2: 可视化结果

可以看出，人物名根据标签传播算法，大致分为了 7 到 8 个类簇，每个类簇大致呈现出椭圆状。有趣的是，左上角的两个不同色类簇，联系十分紧密，两个类簇间有很多节点相交，为同一小说划分为不同类簇，但两者又由于处于同一或系列小说而靠得十分接近（相较于其他类簇）。

将上图放大可得到局部簇群可视化结果，可以看到许多人物的名字，每个人物名是一个节点，节点与节点之间的边表示了两两同线的关系，宏观上边的密集程度也体现了类簇之间的联系紧密程度，图片如下：

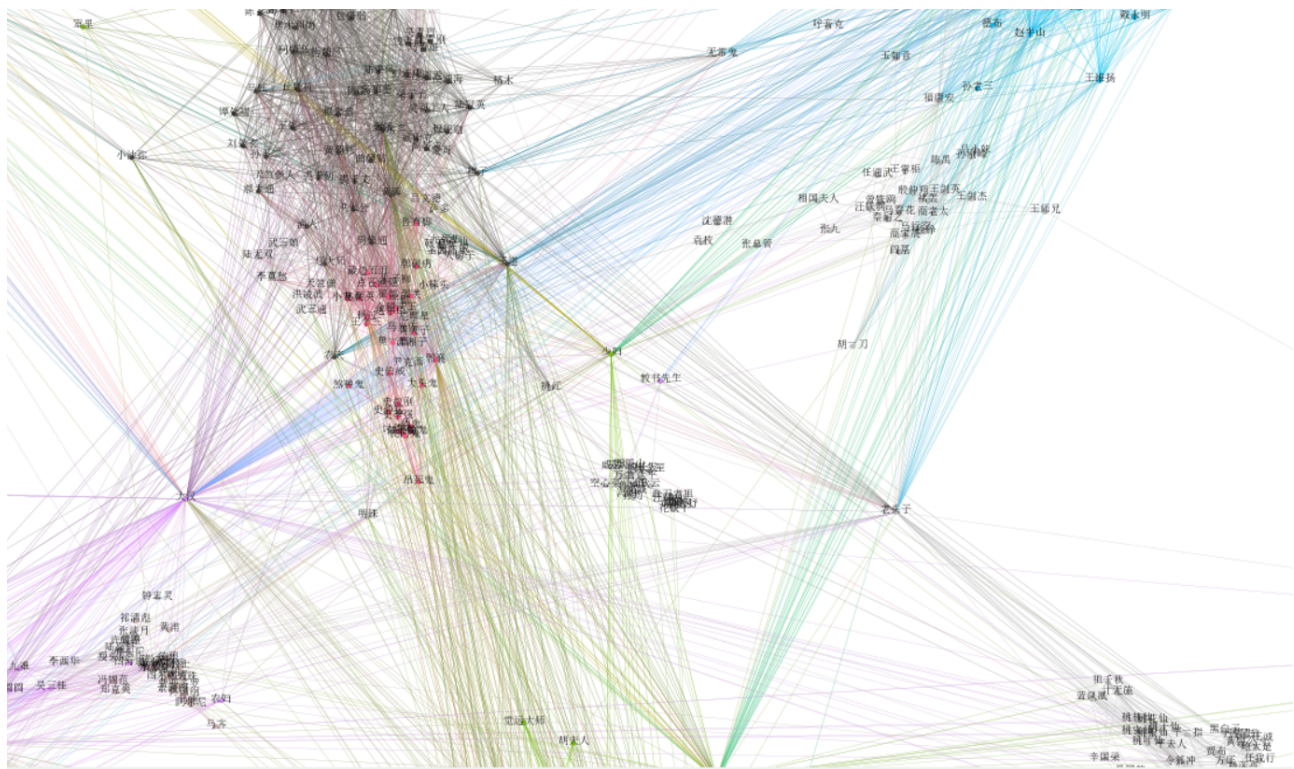


Figure 3: 放大可视化视图

6 程序运行和实验结果说明与分析

6.1 每个 job 的输出结果文件截图

6.1.1 任务 1 数据预处理

```

1 上官 桑飞虹
2 上官 桑飞虹 无常鬼
3 上官 胡斐
4 上官铁生
5 中年武师 孙伏虎 汉子 尉迟连 大汉 杨宾
6 乾隆皇帝 福康安 陈家洛 汤沛
7 任通武 胡斐 福康安
8 任通武 胡斐 福康安
9 何思豪
10 何思豪
11 何思豪 商宝震 徐铮
12 何思豪 少妇
13 何思豪 徐铮
14 何思豪 徐铮
15 何思豪 徐铮 马春花 少妇
16 何思豪 袁紫衣
17 俞朝奉
18 倪不大 倪不小 桑飞虹 福康安 海兰珊 汤沛
19 倪不大 倪不小 福康安 桑飞虹 郭玉堂 胡斐
20 倪不大 倪不小 福康安 胡斐 程灵素 马春花

```

Figure 4: 任务 1 运行结果

6.1.2 任务 2 特征抽取：人物同现统计

1	<丁坚,丹青生>	10
2	<丁坚,令狐冲>	15
3	<丁坚,任我行>	1
4	<丁坚,向问天>	6
5	<丁坚,岳不群>	1
6	<丁坚,左冷禅>	1
7	<丁坚,方生>	1
8	<丁坚,施令威>	8
9	<丁坚,秃笔翁>	8
10	<丁坚,胖子>	1
11	<丁坚,风清扬>	1
12	<丁坚,鲍大楚>	2
13	<丁坚,黄钟公>	2
14	<丁坚,黑白子>	10
15	<万里风,何铁手>	1
16	<万里风,刘培生>	2
17	<万里风,梅剑和>	3
18	<万里风,洞玄>	2
19	<万里风,焦公礼>	3
20	<万里风,程青竹>	1

Figure 5: 任务 2 运行结果

6.1.3 任务 3 特征处理：人物关系图构建与特征归一化

1	丁大全	[汉子,0.05882 杨过,0.05882 小王将军,0.11765 宋五,0.05882 大汉,0.23529 陈大方,0.23529 郭襄,0.05882 王惟
2	丁敏君	[殷素素,0.00437 灭绝师太,0.10917 班淑娴,0.01310 白龟寿,0.02620 简捷,0.00437 纪晓芙,0.13100 胡青牛,0.008
3	万圭	[丁典,0.03385 万震山,0.19077 冯坦,0.01538 凌退思,0.00615 卜垣,0.03692 吴坎,0.07692 周圻,0.01231 哑巴,0
4	万庆澜	[吴国栋,0.00741 书僮,0.01481 周仲英,0.13333 周大奶奶,0.00741 周绮,0.06667 周英杰,0.00741 孟健雄,0.06667
5	上官	[冲虚,0.00543 余鱼同,0.01087 任我行,0.03261 令狐冲,0.03804 东方不败,0.05435 丘处机,0.01087 上官铁生,0.0
6	上官毅山	[余鱼同,0.10870 兆惠,0.02174 周仲英,0.02174 骆冰,0.08696 周绮,0.02174 哈合台,0.02174 孙大善,0.02174
7	上官虹	[陈达海,0.15789 李三,0.26316 史仲俊,0.36842 霍元龙,0.21053]
8	乌旺阿菩	[杨逍,0.03571 范遥,0.21429 赵敏,0.14286 韦一笑,0.03571 鹤笔翁,0.03571 鹿杖客,0.21429 灭绝师太,0.071
9	乌老大	[阿紫,0.01538 丁春秋,0.01923 不平道人,0.10769 乔峰,0.00385 云岛主,0.00385 公冶乾,0.01154 包不同,0.05385
10	乔峰	[吴长风,0.00909 单小山,0.00227 单季山,0.00341 单叔山,0.00227 单伯山,0.00455 包不同,0.01136 刘竹庄,0.001
11	九死生	[人厨子,0.20000 黄蓉,0.20000 张一氓,0.20000 百草仙,0.20000 韩无垢,0.20000]
12	仪和	[莫大,0.00538 不戒和尚,0.00806 东方不败,0.00538 乐厚,0.00269 于嫂,0.03763 令狐冲,0.22312 仪清,0.13172
13	余兆兴	[黎生,0.22222 黄蓉,0.14815 鲁有脚,0.03704 郭靖,0.11111 裘千仞,0.03704 简长老,0.03704 程瑶迦,0.03704 洪
14	佟国纲	[教士,0.05769 林兴珠,0.03846 索额图,0.17308 苏菲亚,0.03846 费要多罗,0.17308 双儿,0.01923 阿二,0.01923
15	侍剑	[说不得,0.02174 丁不三,0.02174 展飞,0.26087 汉子,0.02174 石破天,0.32609 花万紫,0.06522 谢烟客,0.02174
16	俏鬼	[史伯威,0.08333 史仲猛,0.08333 催命鬼,0.08333 丧门鬼,0.08333 笑脸鬼,0.08333 大头鬼,0.25000 吊死鬼,0.166
17	倪浩	[崔秋山,0.11364 孙仲寿,0.12500 大汉,0.01136 哑巴,0.02273 吴平,0.01136 刘芳亮,0.02273 刘培生,0.01136 郑
18	元义方	[司徒伯雷,0.09756 司徒鹤,0.02439 吴三桂,0.19512 吴应彪,0.02439 富春,0.04878 张康年,0.07317 曾柔,0.02439
19	兆惠	[木卓伦,0.08861 李沅芷,0.02110 杨成协,0.01266 樵子,0.00422 武铭,0.00422 汉子,0.00844 汪浩天,0.00422 焦
20	公孙止	[潇湘子,0.01023 丘处机,0.00256 公孙绿萼,0.01023 周伯通,0.01023 天竺僧,0.00767 孙婆婆,0.00256 完颜萍,0.0

Figure 6: 任务 3 运行结果

6.1.4 任务 4 数据分析：基于人物关系图的 PageRank 计算

1	丁大全	3.31E-4
2	丁敏君	8.65E-4
3	万圭	0.001581
4	万庆澜	5.02E-4
5	上官	8.08E-4
6	上官赣山	2.47E-4
7	上官虹	2.93E-4
8	乌旺阿普	1.88E-4
9	乌老大	0.001153
10	乔峰	0.004269
11	九死生	1.28E-4
12	仪和	0.001437
13	余兆兴	1.6E-4
14	佟国纲	3.93E-4
15	侍剑	4.43E-4
16	俏鬼	2.09E-4
17	倪浩	5.98E-4
18	元义方	2.94E-4
19	兆惠	9.55E-4
20	公孙止	9.64E-4

Figure 7: 任务 4 运行结果

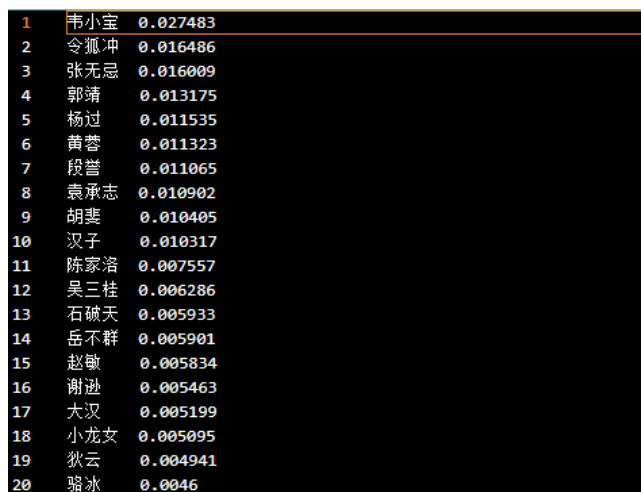
6.1.5 任务 5 数据分析：在人物关系图上的标签传播

(说不得,张中)
(赵敏,张中)
(殷离,张中)
(张松溪,张中)
(范遥,张中)
(殷梨亭,张中)
(谢逊,张中)
(太虚子,张中)
(周芷若,张中)
(闻苍松,张中)
(阳顶天,张中)
(韩千叶,张中)
(小昭,张中)
(宋远桥,张中)
(成昆,张中)
(周颠,张中)
(觉远大师,张中)
(空智,张中)
(殷野王,张中)
(张无忌,张中)
(辛然,张中)
(殷天正,张中)
(空性,张中)
(空闻,张中)
(张三丰,张中)
(杨逍,张中)
(寿南山,张中)
(察罕特穆尔,张中)
(唐洋,张中)

Figure 8: 任务 5 运行结果

6.1.6 任务 6 结果整理

LPA 结果已在任务四进行了整理，无需重复操作。而 PageRank 排序结果如下：



1	韦小宝	0.027483
2	令狐冲	0.016486
3	张无忌	0.016009
4	郭靖	0.013175
5	杨过	0.011535
6	黄蓉	0.011323
7	段誉	0.011065
8	袁承志	0.010902
9	胡斐	0.010405
10	汉子	0.010317
11	陈家洛	0.007557
12	吴三桂	0.006286
13	石破天	0.005933
14	岳不群	0.005901
15	赵敏	0.005834
16	谢逊	0.005463
17	大汉	0.005199
18	小龙女	0.005095
19	狄云	0.004941
20	骆冰	0.0046

Figure 9: 任务 6 PageRank 排序运行结果

7 程序性能优化

使用 Combination 进行计算优化，例如任务二统计同现次数时，对 Mapper 输出的同现组合可以预先求一次和，得到局部的同现次数，在发送给 Reducer 节点，进一步累和，得到全局的同现次数。任务四可以对 Mapper 发出的键值对，在 Combination 中预先对 value 中 rank 值贡献求和，这样最终 Reducer 阶段收到相同 key 的 value 就不会过多。

7.1 参数调优与结果分析

7.1.1 任务二

从任务二的结果中，可以发现：设置的窗口大小不同，人物同现关系也就不一样，当上下文局限于一个段落时，窗口较小，人物同现关系比较局限，在后续的处理例如 LPA 算法中，就会出现许多的孤岛、小簇集。而把窗口设置的过大，又会使得许多没有很强关系的人物之间发生同现关系。导致后续处理例如 PageRank 中一些小人物 rank 值变高，也会使得 LPA 中簇集个数变少，聚类效果模糊，很多人物集合不能很好区分。本实验中使用一段文字作为窗口大小，后续可以提升效果时，可以将窗口大小设置成 3 个段落较为合适。

7.1.2 任务四

对于任务四的结果进行分析，我们可以看出，有许多耳熟能详的人物 rank 居于前列，例如韦小宝、郭靖、令狐冲、袁承志等，这些符合 PageRank 算法所要挖掘的信息，属于金庸小说中的主要角色。但也会出现大汉这样的人物。原因可能是，大汉在任务一分词算法中被划分为了人物名，然后其作为一个被金庸随意所取得名号可能在多本小说中经常出现，所以很容易接收到其他人物甚至重要人物的 rank 值贡献，导致其 rank 值很高。

PageRank 算法中参数 d 控制最终求得 rank 值时随机游走部分的占比，对于本次实验，探索的对象是小说中的人物，人物的 rank 值受同现人物的贡献较大，而受随机游走部分应该较小，对 d 的取值进行讨论，分别取 0.85, 0.88, 0.9，发现 d 取值 0.88 效果较好，不仅控制了随机游走的比例，也不致使随机游走占比过小，两者取了折中。

对于迭代次数，起初使用 20 轮，通过输出结果发现，在十轮以下时，每轮迭代都会使得结果有很大改变，但到十轮以上时，改变越来越小，可以观察到结果正逐步逼近一个不变值。但正如逼近理论，越逼近界限，所需的迭代次数就越大，所需时间和内存是十分不划算的，因此设置 15 为最大迭代轮数，提前收敛结束，虽未完全收敛，但此时相邻迭代结果的变化已然很小。

7.1.3 任务五

对于人物五的结果进行分析，我们可以看出，相同标签的人物都是在同一本书中的人物。但因为人物同现关系窗口选择的大小问题，会出现一些很小的簇或者孤岛。有时同一本书中的人物可能会被划分成几个不同的簇，但是同一个簇里的人物往往在书中是有很强的关联性的，比如《笑傲江湖》中的桃谷六仙和《射雕英雄传》里的江南七怪单独形成了一个小簇。

LPA 在二部图时会出现震荡，当迭代轮数超过 10 轮后就会开始震荡，因此我们只迭代 15 轮。

7.2 HDFS 上的输出路径

7.2.1 任务 1 数据预处理

/user/2019st01/character_extracting

7.2.2 任务 2 特征抽取：人物同现统计

/user/2019st01/cooccurrence_counting

7.2.3 任务 3 特征处理：人物关系图构建不特征归一化

/user/2019st01/relations_freq

7.2.4 任务 4 数据分析：基于人物关系图的 PageRank 计算

/user/2019st01/pagerank

7.2.5 任务 5 数据分析：在人物关系图上的标签传播

/user/2019st01/LPA

7.2.6 任务 6 结果整理

PageRank 结果降序排序：

/user/2019st01/PageRankSort

LPA 结果整理：

/user/2019st01/LPA/collection

7.3 Hadoop 上 WebUI 执行报告

7.3.1 任务 1 数据预处理

Application Overview	
User:	2019st01
Name:	CharacterExtracting
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Fri Jul 19 10:58:27 +0800 2019
Elapsed:	34sec
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:	211266 MB-seconds, 56 vcore-seconds

Figure 10: 任务 1 CharacterExtracting WebUI

7.3.2 任务 2 特征抽取：人物同现统计

Application Overview	
User:	2019st01
Name:	CooccurrenceCounting
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Fri Jul 19 11:01:04 +0800 2019
Elapsed:	25sec
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:	1500588 MB-seconds, 184 vcore-seconds

Figure 11: 任务 2 CooccurrenceCounting WebUI

7.3.3 任务 3 特征处理：人物关系图构建与特征归一化

Application Overview	
User:	2019st01
Name:	RelationBuilding
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Fri Jul 19 11:03:43 +0800 2019
Elapsed:	32sec
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:	845880 MB-seconds, 126 vcore-seconds

Figure 12: 任务 3 RelationBuilding WebUI

7.3.4 任务 4 数据分析：基于人物关系图的 PageRank 计算

Application Overview	
User:	2019st01
Name:	GraghBuilder
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Thu Jul 18 22:52:10 +0800 2019
Elapsed:	19sec
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:	354759 MB-seconds, 59 vcore-seconds

Figure 13: 任务 4 GraghBuilder WebUI

Application Overview	
User:	2019st01
Name:	PageRank
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Thu Jul 18 22:58:53 +0800 2019
Elapsed:	24sec
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:	633369 MB-seconds, 93 vcore-seconds

Figure 14: 任务 4 PageRank WebUI

Application Overview	
User:	2019st01
Name:	GetResult
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Thu Jul 18 22:59:20 +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:	542032 MB-seconds, 81 vcore-seconds

Figure 15: 任务 4 GetResult WebUI

7.3.5 任务 5 数据分析：在人物关系图上的标签传播

Application Overview	
User:	2019st01
Name:	LabelPropagation
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Thu Jul 18 21:15:27 +0800 2019
Elapsed:	19sec
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:	441359 MB-seconds, 69 vcore-seconds

Figure 16: 任务 4 GetResult WebUI

7.3.6 任务 6 结果整理

PageRank 结果排序的 WebUI 截图：

Application Overview	
User:	2019st01
Name:	PageRankSort
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Thu Jul 18 23:22:08 +0800 2019
Elapsed:	20sec
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:	343433 MB-seconds, 58 vcore-seconds

Figure 17: 任务 6 PageRank 结果排序的 WebUI 截图

LPA 结果整理的 WebUI 截图：

Application Overview	
User:	2019st01
Name:	GlobalSort
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Thu Jul 18 21:15:47 +0800 2019
Elapsed:	18sec
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:	732611 MB-seconds, 99 vcore-seconds

Figure 18: 任务 6 LPA 结果整理的 WebUI 截图

8 Hadoop 与 Spark 对比

8.1 编程语言

Scala 是函数式编程语言，需要通过匿名函数的形式传递函数的内容，代码简洁，但是程序员实现较为复杂。而 Java 是对程序员易理解的编程语言，对于复杂的算法实现，依然可以采用优雅的思维。

8.2 编程框架

Hadoop 编程框架易于程序员理解，将并行程序设计抽象成 Map 和 Reduce，使得程序员不用在意底层实现，只需在意 Map 和 Reduce 中处理的键值对即可。但是正是因为 Hadoop 分为 Mapper 和 Reducer 两个固定的阶段，通过文件系统在节点之间传递数据，磁盘计算大量的 IO，导致性能较低，对很多非批处理的大数据问题存在局限性。并且 Hadoop 的计算模式固定，只有 map 和 reduce 两个阶段，对于某些问题的处理不够灵活。而 Spark 编程框架中，以内存计算为核心，速度上明显快于 Hadoop 编程框架。另外，在 Hadoop 中，跨作业数据共享只能通过磁盘文件进行共享，而 Spark 编程框架中，RDD 是基于内存的弹性分布式数据集，通过对 RDD 的一系列操作完成计算任务，可以大大提高性能。同时一组 RDD 形成可执行的有向无环图 DAG，构成灵活的计算流图，覆盖多种计算模式。同时 Spark 打破了 Hadoop 固定的 map 和 reduce 框架，可以灵活地在任务中多次地运用 map 和 reduce 操作。

8.3 程序性能分析

对于任务四和任务五，需要循环迭代计算最终的结果，对程序的运行时间情况进行比较。发现在 Hadoop 中迭代十五轮大约需要 5 分钟的时间，而 Spark 中迭代十五轮只需要不到 10 秒的时间，差距巨大。原因在于 Hadoop 每一轮迭代都是新建一个 MapReduce 的 job 进行配置和启动，而 Spark 多为内存级别操作，读内存进行更新和修改，静态数据直接维护在内存中，例如 PageRank 中的图结构，Hadoop 需要在 Map 和 Reduce 之间发送每个节点的出度列表，而 Spark 只需维护在静态内存中，需要时访问即可。

9 实验总结

本次大实验，我们感受到了一个完整的大数据处理任务是什么样的过程。在完成实验的过程中，小组三人团结合作，分工明确，学习知识之余也体会到到结队完成项目的经历。是一次意义重大的团队项目经历。

深理解了 Hadoop 框架编程，高度抽象的并行程序设计，使得调试程序变得方便，这样程序员只需要专注于高层的内容而无需关注系统底层的细节。

学习了基于 Spark 框架的编程，深入理解了两者的联系与区别，对 Spark 这类计算模型有了一些新的认识，并且体会到了函数式语言的特性，不止限于书本上介绍的那样，亲身实验过，对其易用性和高效快速性有了深入体会。

10 参考文献与致谢

《深入理解大数据——大数据处理与编程实践》——黄宜华

<https://blog.csdn.net/a532672728/article/details/72477591>

<https://blog.csdn.net/u013378306/article/details/52550805>

<http://journals.aps.org/pre/abstract/10.1103/PhysRevE.76.036106>

https://en.wikipedia.org/wiki/Label_Propagation_Algorithm

诚挚感谢提供集群服务器资源的助教和老师!