

# 大数据实验课程设计——《西游记》人物关系挖掘

小组编号：10

组员：

姓名	学号
陈智骐	191220017
罗思明	191220075
张桓	191220156

## 大数据实验课程设计——《西游记》人物关系挖掘

任务描述

小组分工

设计思路和实现

Phase1：数据预处理

1.1 任务描述

1.2 算法原理

1.2.1 分词工具ansj\_seg

1.2.2 Distributed cache的使用

1.3 实现细节

1.3.1 Phase1Mapper类

1.3.2 Phase1Reducer类

Phase2：人物同现统计

2.1 任务描述

2.2 算法原理

2.3 实现细节

2.3.1 Map

2.3.2 自定义Key

2.3.3 Reduce

Phase3：人物关系图构建与特征归一化

3.1 任务描述

3.2 算法原理

3.2.1 Mapper

3.2.2 Reducer

Phase4：基于人物关系图的 PageRank 计算

4.1 任务描述

4.2 算法原理

4.3 实现细节

4.3.1 GraphBuilder类

4.3.2 PageRankIter类

4.3.3 PageRankViewer类

Phase5：标签传播

5.1 任务描述

5.2 算法选择

5.3 算法原理

5.3.1 算法描述

5.3.2 并行化优化

5.3.3 Mapper

#### 5.3.4 Reducer

### 5.4 实现细节

#### 5.4.1 标记数据的处理

#### 5.4.2 初始化

#### 5.4.3 输出结果

#### 5.4.4 标签的设置

数据可视化处理

数据转换

源代码编译方式

Jar包运行方式

实验结果与运行截图

Phase1: 数据预处理

Phase2: 人物同现统计

Phase3: 人物关系图构建与特征归一化

Phase4: PageRank

Phase5: 标签传播

Phase6: 可视化

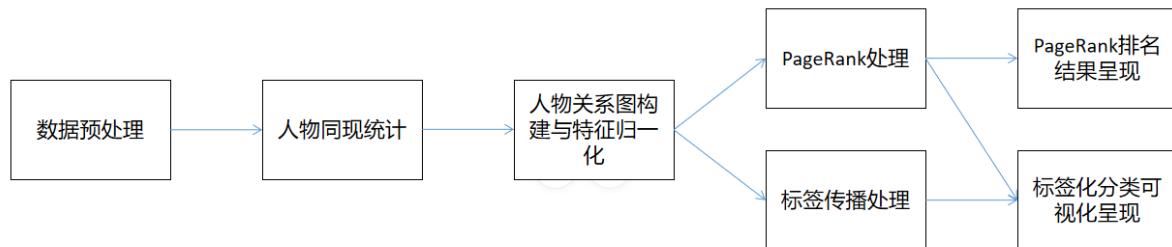
性能分析

## 任务描述

通过一个综合数据分析案例：“西游释厄传——西游记中的人物关系挖掘”，来学习和掌握 MapReduce 程序设计。通过本课程设计的学习，可以体会如何使用 MapReduce 完成一个综合性的数据挖掘任务，包括全流程的数据预处理、数据分析、数据后处理等；实验涉及的主要内容有：

- 在 Hadoop 中使用第三方的 JAR 包来辅助分析
- 掌握简单的MapReduce算法设计，包括单词同现算法、数据整理与归一算法以及数据排序等
- 掌握带有迭代特性的MapReduce算法设计，包括PageRank算法与标签传播算法等。

本组实验的实验流程如下：



## 小组分工

- 陈智骐：Phase3人物关系图构建与特征归一化、Phase5标签传播算法
- 罗思明：Phase2人物同现统计、Phase6可视化数据转换
- 张桓：Phase1数据预处理、Phase4 PageRank算法

实验报告与实验ppt共同撰写完成

## 设计思路和实现

# Phase1：数据预处理

## 1.1 任务描述

从原始的西游记小说的文本中，抽取出与人物互动相关的数据。需要屏蔽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。

- 数据输入：西游记系列小说文集（未分词）；西游记系列小说中的人名列表
- 数据输出：分词后保留人名

比如输入：(西游记中的某一段内容) 将近天门，金星高叫道：“那天门天将，大小吏兵，放开路者。此乃下界仙人，我奉玉帝圣旨，宣他来也。”这增长天王与众天丁俱才敛兵退避。猴王始信其言。同金星缓步入里观看…

输出: 金星 玉帝 增长天王 猴王 金星

## 1.2 算法原理

### 1.2.1 分词工具ansj\_seg

本任务需要将原小说文本中的人名提取出来，我们已经有了西游记的人名列表 `xiyouji_name_list.txt`，因此工作就在于匹配原文中的出现的人名并输出。这里使用开源的 Ansj\_seg 进行分词，为了准确识别出人名，可以利用 Ansj\_seg 的 **自定义词典** 功能，首先将人名列表里的所有人物都加入到用户自定义词典中，然后再去匹配。

### 1.2.2 Distributed cache 的使用

每个Mapper会获得小说的部分文本，因此需要给每个Mapper都设置好自定义词典，比你根据词典内容在分到的部分文本中匹配人名。于是就考虑使用分布式缓存，在Mapper的 `setup()` 函数中，拉取分布式缓存中的人名列表文件，并建立自定义词典。

## 1.3 实现细节

### 1.3.1 Phase1Mapper类

这个类的任务是在 `setup` 阶段拉取分布式缓存的人名列表，并将其中的人名添加到自定义词典中，然后用词典匹配文本里出现的人名并且输出。由于实验数据中每一段都放在了一行里，因此只需将每一行中的人名找出，拼接成 `金星 玉帝 增长天王 猴王 金星` 的形式输出即可，基本实现如下：

```
public class Phase1Mapper extends Mapper {  
    private HashSet<String> nameSet = new HashSet<String>(); // 将姓名存入hash表  
    protected void setup(Context context) {  
        // 从分布式缓存中获取人名列表文件  
        URI[] cacheFiles = context.getCacheFiles();  
        ...  
        // 读取人名加入到自定义词典中  
        String line = null;  
        while (null != (line = bufferedReader.readLine())) {  
            nameSet.add(line);  
            DicLibrary.insert(DicLibrary.DEFAULT, line);  
        }  
    }  
}
```

```

protected void map(LongWritable key, Text value, Context context) {
    // 使用ansj_seg对段落分析提取人名
    String line = value.toString();
    Result result = DicAnalysis.parse(line);
    List<Term> names = result.getTerms();
    // 本段落出现的人名写入一行
    StringBuilder stringBuilder = new StringBuilder();
    for (Term name : names) {
        if (nameSet.contains(name.getName())) { // 是想要的人名
            stringBuilder.append(name.getName() + " ");
        }
    }
}
}

```

### 1.3.2 Phase1Reducer类

由于 `Phase1Mapper` 类已经提取出了每一段的所有人名并拼接完成，Reducer端直接输出即可，无需额外处理。

## Phase2：人物同现统计

### 2.1 任务描述

利用任务1的输出，统计人物的同现关系：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，表明两人之间的关系越密切；而任务1的统计结果中，**一个段落中的人名列表作为一个行单位输出到文件中**，因此，只需要逐行解析任务1的结果文件并进行统计和格式化的输出即可。

此外，对于一个主要人物的不同指代（例如悟空和猴王），需要进行额外处理将其统一起来，不需要再进行针对别名的额外统计。

例如：

输入：

唐僧 悟空 猴王 八戒

悟空 八戒

输出：

<唐僧, 悟空> 1

<唐僧, 八戒> 1

<悟空, 唐僧> 1

<悟空, 八戒> 2

<八戒, 唐僧> 1

<八戒, 悟空> 2

## 2.2 算法原理

这一阶段的工作并不算复杂，有几个主要设计：

- mapreduce任务设置中，设置输入格式为 `TextInputFormat`，map阶段针对文件的每一行进行解析；
- 为了统一别名，在map的 `setup` 阶段，预先编辑一个**字典**，参考人名列表文件，将**别名作为键、统一名为值写入字典**；当处理文件时，检测到某一个字典键时，就取出其对应的字典值，如此一来就实现了名统一；
- 由于会存在一个人物多次出现的情况，统计一段中的人名时，首先将它们放入一个**集合**中，可以实现去重；一个段落统计完毕后，再对集合中的元素两两确定同现关系；
- 同现关系要求自定义键值，其内部存储了同现关系对应的两个人物名，并实现了 `WritableComparable` 接口。

## 2.3 实现细节

### 2.3.1 Map

```
public class Phase2Mapper extends Mapper<LongWritable, Text, Phase2Key, IntWritable> {
    Map<String, String> name_tab = new HashMap<>(); //别名和统一的映射字典

    @Override
    protected void setup(Mapper<LongWritable, Text, Phase2Key, IntWritable>.Context context) throws IOException, InterruptedException {
        super.setup(context);

        //将<别名, 统一名>写进字典, 其他人物同
        name_tab.put("唐三藏", "唐僧");
        name_tab.put("陈玄奘", "唐僧");
        name_tab.put("玄奘", "唐僧");
        ...
    }

    @Override
    protected void map(LongWritable key, Text value, Mapper<LongWritable, Text, Phase2Key, IntWritable>.Context context) throws IOException, InterruptedException {
        String[] str_list = value.toString().split(" ");

        Set<String> name_set = new HashSet<>();
        for(String name: str_list) {
            if(name_tab.containsKey(name))
                //如果是别名, 那么将统一名加入集合中
                name_set.add(name_tab.get(name));
            else
                name_set.add(name);
        }

        //遍历集合中的名字, 两两之间向外输出同现关系
        int n = name_set.size();
        String[] name_list = name_set.toArray(new String[n]);
        for(int i = 0; i < n; i++) {
            for(int j = i + 1; j < n; j++) {
```

```

        context.write(new Phase2Key(new Text(name_list[i]), new
Text(name_list[j])),
                     new IntWritable(1));
        context.write(new Phase2Key(new Text(name_list[j]), new
Text(name_list[i])),
                     new IntWritable(1));
    }
}
}
}

```

### 2.3.2 自定义Key

```

public class Phase2Key implements WritableComparable {
    //属性：同现关系对应的两个人物名
    private Text name_1;
    private Text name_2;
}

```

### 2.3.3 Reduce

```

public class Phase2Reducer extends Reducer<Phase2Key, IntWritable, Text, Text> {
    @Override
    protected void reduce(Phase2Key key, Iterable<IntWritable> values,
    Reducer<Phase2Key, IntWritable, Text, Text>.Context context) throws IOException,
    InterruptedException {
        int sum = 0;
        //简单地将每一个键对应的值进行相加
        while(values.iterator().hasNext())
            sum += values.iterator().next().get();
        context.write(new Text(key.toString()), new Text(string.valueOf(sum)));
    }
}

```

## Phase3：人物关系图构建与特征归一化

### 3.1 任务描述

这个阶段以phase2的输出作为输入，根据共现关系，生成人物之间的关系图，以邻接表形式表示。其中如果两个人之间具有共现关系，则有一条边；而共现次数为边的权重。最后，为了后面的分析方便，需要对共现次数进行归一化处理，共现次数转化为共现概率。

### 3.2 算法原理

### 3.2.1 Mapper

Mapper的输入为phase2的输入：key为人物共现对 `(name1, name2)`，value为共现次数 `v`。由于 phase2的输出是对称的，在phase3中可以看作是 `name1 到 name2` 有一条权为 `v` 的有向边。

在Mapper中，将输入key中的 `name1` 为输出的key，其value为 `(name2, v)` 二元组。这样，相同的 `name1` 对应的value会交给一个reduce处理，表示一个节点 `name1` 的所有出边。

### 3.2.2 Reducer

Reducer的输入为 `name1, [(name2, v)]`，`name1` 为正在处理的节点，`[(name2, v)]` 为该节点的出边的列表，这也就是我们所要的邻接表的一行。此外，对于每一行，还需要将边权归一化。这只需要先求出当前节点的共现次数总数 `n`，再将每个边权都除以 `n` 即可。

## Phase4：基于人物关系图的 PageRank 计算

### 4.1 任务描述

根据任务三构建的人物关系图，计算每个任务的PageRank值，从而定量分析小说的主角。

- 数据输入：任务三构建的任务关系图，即

唐僧 [悟空, 0.5 | 八戒, 0.5]  
悟空 [唐僧, 0.33333 | 八戒, 0.66666]

- 数据输出：每个人物的PageRank值，按照PageRank值降序排列

### 4.2 算法原理

PageRank是一种在搜索引擎中根据网页之间相互的链接关系计算网页排名的技术，每个网页的 PageRank 值越高，则该网页在搜索结果中的位置越靠前。在本实验中我们用该算法来分析人物关系网，计算每个人物的PageRank值来定量分析谁是主角。

通常PageRank算法有两种模型：简化模型和随机浏览模型。其中随机浏览模型是由于网络链接图中可能出现某个结点没有出度或者没有入度而导致排名泄露和排名下沉的问题而提出的，但在本实验中，人物关系是双向的，因此绝对不会出现排名泄露和排名下沉的问题，因此使用简化模型更加合适。

简化模型的公式如下，对于任意网页  $P_i$ ，它的PageRank值  $R(P_i)$  为：

$$R(P_i) = \sum_{P_j \in B_i} \frac{R(P_j)}{L_j}$$

- 其中  $B_i$  为所有链接到网页  $P_i$  的网页集合， $L_j$  为网页  $P_j$  的对外链接数量。

而在本实验的任务三中构建的归一化的人物图的形式为 `八戒 [唐僧, 0.33333 | 悟空, 0.66666]`，也就是说经过归一化每个人物出边的权重都已经计算好了，即八戒给唐僧的PageRank值0.33333的贡献，给悟空0.66666的贡献。因此本实验人物  $i$  的PageRank值  $R(P_i)$  计算公式为：

$$R(P_i) = \sum_{P_j \in B_i} R(P_j) \times W_j(P_i)$$

- 其中  $B_i$  为指向人物  $P_i$  的人物列表，而  $W_j(P_i)$  表示人物  $P_j$  对人物  $P_i$  的贡献权重。

## 4.3 实现细节

在MapReduce上实现PageRank算法分成3步：

- 步骤一：GraphBuilder，建立网页之间的超链接图
- 步骤二：PageRankIter，迭代计算各个网页的PageRank值
- 步骤三：RankViewer，按PageRank值从大到小输出

下面依次介绍这三个步骤，分别对应三个类。

### 4.3.1 GraphBuilder类

这个类是为了创建邻接图和赋予初始PR值1.0，由于任务三中已经构建了归一化的人物关系图，因此这个类只需要初始化每个人物初始PR值为1.0即可。样例格式为：

- 数据输入：任务三的输出，即：  
唐僧 [悟空, 0.5 | 八戒, 0.5]
- 数据输出：初始化每个人物的PR值为1.0，即：  
唐僧 1.0\$[悟空, 0.5 | 八戒, 0.5]

为了区分PR值和邻接链表，在二者之间加入了\$符号进行分割。显然这个类中只需要一个Mapper处理每一行输入即可，并不需要Reducer。

### 4.3.2 PageRankIter类

这个类用于迭代计算PR值，直到PR值收敛或迭代预定次数，我们最终选择以命令行参数的方式来规定迭代次数作为迭代终止条件。在该类中需要Mapper和Reducer两个阶段才能完成迭代过程：

- PageRankIterMapper类

这个类对于GraphBuilder类的输出，`唐僧 1.0$[悟空, 0.5 | 八戒, 0.5]` 即对每条记录

`<name, PR$link_list>`，产生两种键值对输出：

- 贡献值

即遍历 `link_list` 中的每个 `(outName, weight)` 输出 `<outName, PR*weight>`，

其中 `outName` 表示当前人物 `name` 指向的人物，`PR*weight` 表示当前人物对 `outName` 的PR贡献值。

- 图的邻接关系

为了位置图的邻接关系用于后续的迭代，对每个 `<name, PR$link_list>` 还需要输出 `<name, link_list>`。`link_list` 的形式为 `[outName1, weight1 | outName2, weight2 ...]`。

- PageRankIterReducer类

这个类对从PageRankIterMapper类得到的两种键值对进行分析处理，由于上一步输出的键值对中键都为人名，因此相同人名的键值对将输出到同一个PageRankIterReducer下。

因此PageRankIterReduce需要将得到的多个 `<outName, PR*weight>` 的所有 `PR*weight` 相加得到 `cur_PR`，作为本人物下一轮迭代的PR值，同时保存邻接链表信息 `link_list`，将二者拼接得到键值对 `<name, cur_PR$link_list>` 输出。

### 4.3.3 PageRankViewer类

为了实现结果按照PR值降序排列，我们利用的是MapReduce自带的Shuffle功能，因此就必须将PR值放到键中。即对上一步迭代后得到的 `<name, cur_PR$link_list>` 提取生成 `<cur_PR, name>` 来进行排序。

而MapReduce默认使用升序排列，因此需要自定义数据结构 `DecDoublewritable` 类作为 `cur_PR` 的类型，

这个类继承自 `Doublewritable`，重载了其 `compareTo` 方法，代码如下：

```
public static class DecDoublewritable extends Doublewritable {  
    ...  
    @Override  
    public int compareTo(Doublewritable o) {  
        return -super.compareTo(o);  
    }  
}
```

## Phase5：标签传播

### 5.1 任务描述

在phase3得到了任务共现关系图之后，可以在此之上实现标签传播算法，通过在图的顶点上打标签，进行图顶点的聚类分析，实现《西游记》中人物关系的挖掘。

### 5.2 算法选择

我们查阅资料后发现，标签传播其实是一类算法，其中有不同具体的实现版本：有半监督的、也有无监督的，这些算法各有优劣，适合于不同种类的聚类任务。在分析了我们的实验使用场景后，我们选择了下述的这种基于概率矩阵的软标签半监督版本的标签传播算法。

这个算法与其它的标签传播算法相比的特点是：

- 它是半监督的，可以人为设置少量初始的标签，结果有比较好的可解释性，利于实现本实验中的启发式的人物关系挖掘
- 它基于矩阵运算，具有很好的并行性，可以利用MapReduce框架提供很好的优化效果

### 5.3 算法原理

#### 5.3.1 算法描述

设总共有  $C$  个类，和  $N$  个样本，其中有  $L$  个初始标记过的样本，定义一个  $L \times C$  矩阵  $Y_L$ ，它的第  $i$  行表示样本  $i$  标签的指示向量，若样本  $i$  的类别为  $j$ ，则  $Y_L(i, j) = 1$ ，否则  $Y_L(i, j) = 0$ 。类似的，对于剩下的  $U$  个未标记样本，定义  $U \times C$  的矩阵  $Y_U$ ， $Y_U(i, j)$  表示这个样本  $i$  归为类  $j$  的可能性，为一个  $[0, 1]$  的数，这样在标签传播的过程中，每个样本的归类就不是固定的，这称为 **软标签** (soft label)。最后，将  $Y_L$  和  $Y_U$  合并，得到一个所有样本的软标签矩阵：

$$F = \begin{pmatrix} Y_L \\ Y_U \end{pmatrix}$$

同时，在phase3，我们得到了一个人物共现矩阵，而其中的边权进行了归一化，可以看作是每个节点的出边的概率矩阵 $P$ 。按照标签传播算法的表述，每个节点会根据这个概率向它的邻接节点传播它的标签；而每个节点新一轮的标签值为它收到的标签值的和；另外，标记的样本的标签始终不受影响。而标签传播的过程可以用 $F$ 左乘概率矩阵 $P$ 得到，由此就得到了标签传播的迭代算法：

```

do
   $F \leftarrow P F$ 
   $F_L \leftarrow Y_L$ 
until  $F_L$ 收敛 或 到达迭代上限

```

最后，对于任一样本 $i$ ，它聚类的结果为 $j$ ，使得 $F(i, j) = \max_{k \in [1..N]} \{F(i, k)\}$ .

### 5.3.2 并行化优化

由于这个标签传播算法是基于矩阵计算的，因此可以将其设计成并行算法，并可以方便地使用MapReduce框架实现。

由矩阵乘法的定义，每一轮迭代得到的 $F(i, j)$ 为 $P(i, -)$ 和 $F(-, j)$ 的点积，这样每个元素的计算都是独立了，可以并行计算。每次计算涉及 $P$ 和上一次迭代的 $F$ 两个数据，由于 $P$ 是在整个算法中固定的，而 $F$ 会随着每次迭代改变，因此这里我们设计将 $P$ 作为每个Mapper拥有一份的数据，而将 $F$ 矩阵按列切分，分配给不同的Mapper节点并行处理。

另外，由于矩阵乘法中对应 $F$ 的运算分量是列向量，因此 $F$ 矩阵在迭代过程中表示为按列存的邻接表。即它的签名为`(标签, [(样本, 值)])`。

### 5.3.3 Mapper

在`setup`函数中，读取phase3的输出文件，构造矩阵 $P$ 。

Mapper的输入为 $F$ 矩阵的一列 $j$ ，`map`函数将这一列与 $P$ 的每一行 $i$ 相乘，得到新一轮的 $F(i, j)$ 输出，输出格式key为列 $j$ （即标签），value为行 $i$ （即样本名）和 $F(i, j)$ 的值的对。这个输出格式使得相同的列可以分到一个reduce，从而保持新一轮的 $F$ 矩阵也是按列存储的，以便下一轮的迭代。

### 5.3.4 Reducer

Reducer的输入是同一列 $j$ 的所有 $(i, F(i, j))$ 的列表，这就对应着 $F$ 矩阵按列存储的邻接表，在reduce中完成序列化输出即可。

## 5.4 实现细节

### 5.4.1 标记数据的处理

一个最重要的细节是对于标记数据的处理。其实这个算法中，对于标记的数据和未标记的数据的处理逻辑是基本一致的，唯一的区别是标记数据的值在迭代时始终不变。因此，我们只需要在每次迭代后保证有标记的数据保持不变即可。

在Mapper初始化时，再保存一个记录标记数据的表 $M : L \rightarrow C$ ，由于半监督学习场景下标记数据量是很小的，因此不会引入性能瓶颈。然后，在迭代计算时，每次计算 $F(i, j)$ 数据时，都先在 $M$ 中查找：

```

if  $M(i) = \text{null}$  :
     $F(i, j) \leftarrow$  迭代的新值
else if  $M(i) = j$  :
     $F(i, j) \leftarrow 1$ 
else :
     $F(i, j) \leftarrow 0$ 

```

第一种情况对应非标记数据，为计算得到的新值；第二、三种情况对应标记数据，保持原值不变。

### 5.4.2 初始化

这个算法有两个输入 $P$ 和 $F$ ，其中 $P$ 在Mapper的 `setup` 中从phase3的输出中读入；而 $F$ 则为前一次的迭代结果，作为 `map` 函数的输入。因此我们还需要一个初始的 $F_0$ . 在 $F_0$ 中，标记的数据就使用它的原值构建，而对于未标记的数据，算法描述中称它的值可以为任意值，对此我们设计了两种策略：

1. 将未标记的数据每个的类别初始化为其自身
2. 将未标记的数据每个随机分配一个已标记的类别

在实现中，这两种策略的表现有所不同。事实上，1对应的分类数 $C$ 等于样本数 $N$ ；而2的分类数 $C$ 等于标签数 $L$ 。这样如果使用1，一些边缘的角色可能不属于任何一个标记的类别而自成一类；而使用2可以强制要求每个角色都归为我们所标记的几个类别之一。我们讨论后认为这两种实现都有道理，结果也具有可解释性，于是保留了对应实现。

### 5.4.3 输出结果

最后还需要将最后一次迭代的 $F$ 矩阵处理成可读的结果，这还需要两步：

1. 首先要将按列存储的 $F$ 再按行整理出每个样本（每行）最大的 $F(i, j)$ 对应的类。
2. 然后，再将相同聚类的样本整理到一起。

为此，我们又定义了两个MapReduce任务。

对于1，设计Mapper将以邻接表格式储存的每一列数据，还原成key为 $i$ （样本），value为 $(j, F(i, j))$ 对的形式，以行 $i$ 做合并。到了Reducer，就统计每个行 $i$ 中最大的 $F(i, j)$ 对应的 $j$ 作为它的聚类结果输出，输出key为标签，value为这个样本。

对于2，它的输入是1的输出，设计Mapper为ID函数，Reducer中自然得到了key为标签的所有样本，序列化输出即可。

### 5.4.4 标签的设置

最后，在执行这个半监督算法之前，我们还要设置一些初始标签。这里我们设计了几种策略：

- 师徒四人
- 师徒四人+白龙马
- 所有主要人物，即师徒四人+白龙马+如来佛祖+观音菩萨+玉皇大帝

我们将这些设计的标签都进行了运行测试、比较输出，以取得较好的人物关系挖掘结果。

# 数据可视化处理

## 数据转换

在执行完标签传播算法之后，为了使分类结果有更加直观的表达，可以对实验数据进行处理和转换，使其进行可视化呈现。

在本次实验中利用**Gephi**软件进行可视化处理，为了达到可视化的效果，需要提供两个csv文件：

- 节点文件，提供每一个节点的id、name（人物名，用于节点标识）、tag（标签传播处理后的标签，用于颜色分类）和pr（PageRank值，用于节点大小排序）；
- 边文件（在本次实验中为有向边），提供每一条边的id、source\_id（出节点id）、target\_id（入节点id）和weight（权重）。

由此设计节点和边的数据结构：

```
//节点
public static class Point{
    private int id;
    private String name;
    private int tag;
    private double pr;
    ...
}

//边
public static class Edge{
    private int id;
    private int source_id;
    private int target_id;
    private double weight;
    ...
}
```

而需要用到的结果数据格式：

- PageRank后结果（未进行排序处理前），即为`dataFile`：

**Name \t PageRank\$[Name\_0, weight\_0|Name\_1, weight\_1|...]**

- 标签传播处理后结果，即为`tagFile`：

**Tag \t Name\_0 Name\_1 ...**

因此编写程序进行程序转换：

- 首先读取`dataFile`，对每一行的`Name`进行id映射，填入字典中（方便后续根据同现关系中的两个人物名确定边的出节点id和入节点id），同时向节点列表中添加新的Point对象，初始化其id、name和pr属性；
- 然后读取`tagFile`，在每一个tag下，对于读取到的每一个人物名，通过字典取出对应值（即id），寻找到在节点列表中位置，初始化tag属性：节点列表视为初始化完成；
- 再读取`dataFile`，对于一个人物的每一个邻居`Name_i weight_i`，初始化一个Edge对象，填入id、source、target（通过名字和id的映射字典获得值）以及weight：边列表视为初始化完成；
- 将节点列表和边列表分别写入csv中，导入Gephi中即可完成可视化。

```
Map<String, Integer> name_map = new HashMap<>(); //人物名和id的映射字典
List<Point> pointList = new ArrayList<>(); //点列表
List<Edge> edgeList = new ArrayList<>(); //边列表
```

```

String lineText = "";
BufferedReader br = new BufferedReader(new FileReader("dataFile"));
while((lineText = br.readLine()) != null) {
    //lineText结构: Name \t PageRank$[Name_0, weight_0|Name_1, weight_1|...]
    int id = name_map.size();
    String name = lineText.split("\t")[0];
    double pr = Double.parseDouble(lineText.split("\t")[1].split("\\"$")[0]);

    name_map.put(name, id);                                //添加映射<name, id>
    pointList.add(new Point(id, name, pr));                //添加新节点, 初始化id, name, pr
属性
}
br.close();

lineText = "";
br = new BufferedReader(new FileReader("tagFile"));
while((lineText = br.readLine()) != null) {
    //lineText结构: Tag \t Name_0 Name_1 ...
    int tag = Integer.parseInt(lineText.split("\t")[0]);

    String[] name_list = lineText.split("\t")[1].split(" ");
    for(String name:name_list) {
        //通过name_map获取id(即点列表的索引), 并完善节点属性: 设置tag
        pointList.get(name_map.get(name)).setTag(tag);
    }
}
br.close();

lineText = "";
br = new BufferedReader(new FileReader("dataFile"));
while((lineText = br.readLine()) != null) {
    //lineText结构: Name \t PageRank$[Name_0, weight_0|Name_1, weight_1|...]
    //获取邻居列表neighbor_list
    String prestr = lineText.split("\t")[1].split("\\"$")[1];
    preStr = preStr.substring(1, preStr.length() - 1);
    String[] neighbor_list = preStr.split("\\"|");
    for(String neighbor:neighbor_list) {
        int id = edgeList.size();
        int source_id = name_map.get(lineText.split("\t")[0]);
        int target_id = name_map.get(neighbor.split(", ")[0]);
        double weight = Double.parseDouble(neighbor.split(", ")[1]);

        edgeList.add(new Edge(id, source_id, target_id, weight));      //添加新边
    }
}
br.close();

//写入point.csv: 属性按照id name tag pr的顺序写入, 中间用逗号隔开
...
//写入edge.csv: 属性按照id source_id target_id weight的顺序写入, 中间用逗号隔开
...

```

## 源代码编译方式

首先在IDEA的项目结构中添加hadoop相关的依赖项；由于在数据预处理阶段需要用到外部的分词工具，因此需要额外添加依赖包 `ansj_seg-5.1.6.jar` 以及 `nlp-lang-1.7.1.jar`，添加这些依赖后可以编译运行。

## Jar包运行方式

本次实验有两种最终输出：PageRank排名和人物标签化分类，需要不同的运行指令：

- 得到PageRank排名结果：

```
hadoop jar final.jar MainDriver /data/2022s/relation/xiyouji_name_list.txt  
/data/2022s/relation/xiyouji pagerank 20
```

第一个参数指示运行的jar包为 `final.jar`，第二个参数指示主类为 `MainDriver`，第三、第四个参数指示需要用到的初始数据（人名列表和分章的小说内容），第五个参数 `pagerank` 指示最终得出的是 PageRank 排名结果，第六个参数 `20` 指示 PageRank 的迭代次数。

- 得到人物标签化分类结果：

```
hadoop jar final.jar MainDriver /data/2022s/relation/xiyouji_name_list.txt  
/data/2022s/relation/xiyouji labelprop 20
```

前四个参数与前述无异，第五个参数 `labelprop` 指示最终得出的是人物标签化分类结果，第六个参数 `20` 指示标签化传播算法的迭代次数。

这两种指令的一系列运行结果都会输出到 `/user/2021sh10/xiyouji` 目录下。

## 实验结果与运行截图

### Phase1：数据预处理

Phase1的输出结果在 `/user/2021sh10/xiyouji/Phase1Output/part-r-00000` 中，每一行代表一个段落中出现的人名列表，前后以空格隔开：

File - /user/2021sh10/xiyouji/Phase1Out...

Page 1 of 15 ▶◀▶▶◀

```
七衣仙女  
七衣仙女 王母 齐天大圣 王母  
三千揭谛 迦叶 阿傩 如来 如来 孙行者 三千揭谛  
三官 阿弥陀佛 唐僧 虎力大仙 八戒  
三星  
三星 悟空  
三星 美猴王  
三星 镇元子  
三星 镇元子 唐僧 八戒 寿星 寿星 八戒 福星 八戒 八戒 三星 三星 禄星 孙行者 寿星 八戒 福星 八戒 八戒  
三星 镇元子 镇元子 三星 唐僧 八戒 沙僧  
三清  
三清 猴王  
三清 羊力大仙 八戒 沙僧  
三清 虎力大仙  
不坏尊王 永住全刚 如来 如来 佛祖 孙悟空 如来 四大全刚 如来 如来 恒空
```

运行截图：

Logged in as: sh10

# hadoop Application application\_1656400433955\_2787

**Cluster**  
 About  
 Nodes  
 Node Labels  
 Applications  
 NEW  
 NEW\_SAVING  
 SUBMITTED  
 ACCEPTED  
 RUNNING  
 FINISHED  
 FAILED  
 KILLED  
  
**Scheduler**  
  
 Tools

**Kill Application**

User: 2021sh10		Application Overview
Name: final.jar		
Application Type: MAPREDUCE		
Application Tags:		
YarnApplicationState: FINISHED		
Queue: root.2021s		
FinalStatus Reported by AM: SUCCEEDED		
Started: Tue Jul 05 14:48:26 +0800 2022		
Elapsed: 1mins, 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: 4857743 MB-seconds, 608 vcore-seconds

Show 20 ▾ entries	Attempt ID	Started	Node	Logs	Blacklisted Nodes
	appattempt_1656400433955_2787_000001	Tue Jul 5 14:48:26 +0800 2022	http://slave016:8042	Logs	N/A

## Phase2: 人物同现统计

Phase2的输出结果在 /user/2021sh10/xiyouji/Phase2Output/part-r-00000 中，每一行格式为 <Name\_0, Name\_1> count，指示Name\_0和Name\_1的同现次数：

File - /user/2021sh10/xiyouji/Phase2Out...

Page 1 of 52 ▶◀▶▶◀

```

<--秤金, 唐僧> 2
<--秤金, 如来佛祖> 1
<--秤金, 沙僧> 2
<--秤金, 灵感大王> 3
<--秤金, 猪八戒> 7
<--秤金, 陈关保> 4
<--秤金, 陈清> 4
<--秤金, 陈澄> 4
<七政, 二十八宿> 1
<七政, 可韩丈人真君> 1
<七政, 太阳> 1
<七政, 太阴> 1
<七政, 孙悟空> 1
<七政, 玉皇大帝> 1

```

Kill Application

User:	2021sh10
Name:	Phase2
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
Queue:	root.2021s
FinalStatus Reported by AM:	SUCCEEDED
Started:	Tue Jul 05 14:49:49 +0800 2022
Elapsed:	17sec
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:	122410 MB-seconds, 31 vcore-seconds

Show 20 entries Search:

Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1656400433955_2788_000001	Tue Jul 5 14:49:49 +0800 2022	http://slave02:8042	Logs	N/A

运行截图：

## Phase3：人物关系图构建与特征归一化

Phase3的输出结果在/`/user/2021sh10/xiyouji/Phase3Output/part-r-00000`中，每一行格式为`Name\t[Name_0, weight_0|Name_1, weight_1|...]`，`Name_i`为Name的邻居，`weight_i`为归一化处理后的同现频率，一个Name对应的所有`weight_i`之和为1。

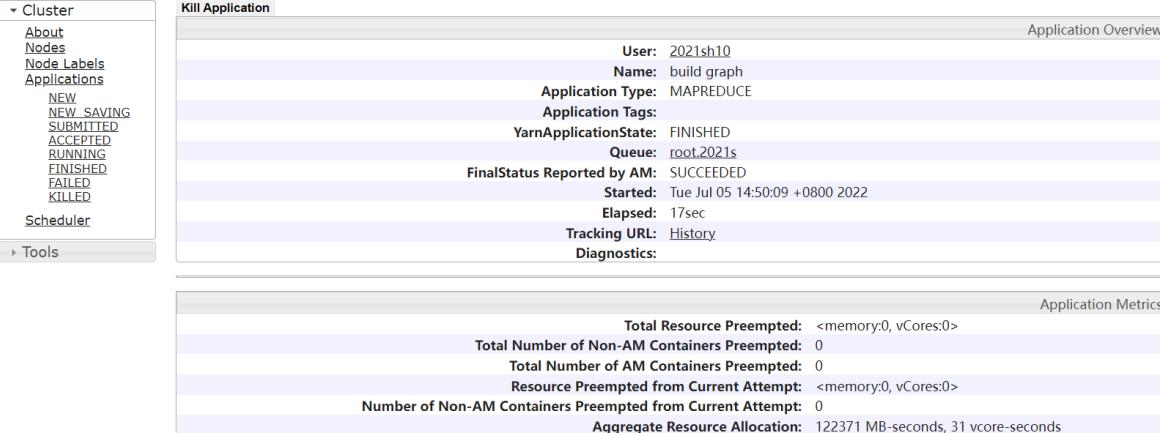
File - /user/2021sh10/xiyouji/Phase3Out...

Page 1 of 43 ▶◀▶▶◀

一秤金 [唐僧, 0.074074 如来佛祖, 0.037037 沙僧, 0.074074 灵感大王, 0.111111 猪八戒, 0.259259 陈关保, 0.148148 陈清, 0.148148 陈澄, 0.148148]
七政 [二十八宿, 0.142857 可韩丈人真君, 0.142857 太阳, 0.142857 太阴, 0.142857 孙悟空, 0.142857 玉皇大帝, 0.142857 诸天, 0.142857]
七衣仙女 [三清, 0.041667 中央黄极, 0.041667 黄角大仙, 0.041667 黄衣仙女, 0.041667 青衣仙女, 0.041667 绿衣仙女, 0.041667 红衣仙女, 0.041667 紫衣仙女, 0.041667 素衣仙女, 0.041667 皂衣仙女, 0.041667 王母, 0.083333 玉皇大帝, 0.041667 孙悟空, 0.083333 土地, 0.041667 四帝, 0.041667 南方南极观音, 0.041667 十洲三岛仙翁, 0.041667 北方北极玄灵, 0.041667 力士, 0.041667 五方五老, 0.041667 五斗星君, 0.041667 东方崇恩圣帝, 0.041667]
万圣公主 [九头驸马, 0.111111 灞波儿奔, 0.111111 牛魔王, 0.111111 奔波儿灞, 0.111111 黑鱼精, 0.111111 鲇鱼怪, 0.111111 万圣龙王, 0.111111 神通广大, 0.111111 孙悟空, 0.111111]
万圣龙王 [唐僧, 0.111111 九头驸马, 0.166667 万圣公主, 0.055556 黑鱼精, 0.055556 鲇鱼怪, 0.055556 猪八戒, 0.111111 神通广大, 0.055556 王母, 0.055556 牛魔王, 0.055556 灞波儿奔, 0.055556 沙僧, 0.055556 孙悟空, 0.111111 奔波儿灞, 0.055556]
万岁狐王 [玉面公主, 0.166667 神通广大, 0.166667 猪八戒, 0.166667 牛魔王, 0.166667 罗刹女, 0.166667 土地, 0.166667]

运行截图：

# hadoop Application application\_1656400433955\_2789



The screenshot shows the Hadoop Application Overview page for application\_1656400433955\_2789. Key details include:

- User: 2021sh10
- Name: build graph
- Application Type: MAPREDUCE
- Application Tags: FINISHED
- YarnApplicationState: FINISHED
- Queue: root.2021s
- FinalStatus Reported by AM: SUCCEEDED
- Started: Tue Jul 05 14:50:09 +0800 2022
- Elapsed: 17sec
- Tracking URL: History
- Diagnostics:

Application Metrics section shows:

- Total Resource Preempted: <memory:0, vCores:0>
- Total Number of Non-AM Containers Preempted: 0
- Total Number of AM Containers Preempted: 0
- Resource Preempted from Current Attempt: <memory:0, vCores:0>
- Number of Non-AM Containers Preempted from Current Attempt: 0
- Aggregate Resource Allocation: 122371 MB-seconds, 31 vcore-seconds

Logs table:

Show 20 entries	Attempt ID	Started	Node	Logs	Blacklisted Nodes
	appattempt_1656400433955_2789_000001	Tue Jul 5 14:50:09 +0800 2022	http://slave012:8042	Logs	N/A

## Phase4: PageRank

Phase4的输出结果在/`user/2021sh10/xiyouji/Phase4Output`中，迭代中间结果在该目录下的`Data1/part-r-00000`至`Data20/part-r-00000`中，每一行格式为`Name \t PageRank$[Name_0, weight_0|Name_1, weight_1|...]`，PageRank即为该人物的当前pagerank值；排名后的结果在该目录下的`FinalRank/part-r-00000`中，每一行格式为`Name \t PageRank`，按照PageRank值从大到小进行排序。

File - /user/2021sh10/xiyouji/Phase4Out...

Page 1 of 45 ▶◀▶▶◀

```

一秤金 0.7978846858182135|[唐僧, 0.074074|如来佛祖, 0.037037|沙僧, 0.074074|灵感大王, 0.111111|猪八戒, 0.259259|陈关保, 0.148148|陈清, 0.148148|陈澄, 0.148148]
七政 0.20676716972962478|[二十八宿, 0.142857|可韩丈人真君, 0.142857|太阳, 0.142857|太阴, 0.142857|孙悟空, 0.142857|玉皇大帝, 0.142857|诸天, 0.142857]
七衣仙女 0.7076942107077825|[三清, 0.041667|中央黄极, 0.041667|黄角大仙, 0.041667|黄衣仙女, 0.041667|青衣仙女, 0.041667|绿衣仙女, 0.041667|红衣仙女, 0.041667|紫衣仙女, 0.041667|素衣仙女, 0.041667|皂衣仙女, 0.041667|王母, 0.083333|玉皇大帝, 0.041667|孙悟空, 0.083333|土地, 0.041667|四帝, 0.041667|南方南极观音, 0.041667|十洲三岛仙翁, 0.041667|北方北极玄灵, 0.041667|力士, 0.041667|五方五老, 0.041667|五斗星君, 0.041667|东方崇恩圣帝, 0.041667]
万圣公主 0.26606667190319816|[九头驸马, 0.111111|灞波儿奔, 0.111111|牛魔王, 0.111111|奔波儿灞, 0.111111|黑鱼精, 0.111111|鲇鱼怪, 0.111111|万圣龙王, 0.111111|神通广大, 0.111111|孙悟空, 0.111111]
万圣龙王 0.5324849412920823|[唐僧, 0.111111|九头驸马, 0.166667|万圣公主, 0.055556|黑鱼精, 0.055556|鲇鱼怪, 0.055556|猪八戒, 0.111111|神通广大, 0.055556|王母, 0.055556|牛魔王, 0.055556|灞波儿奔, 0.055556|沙僧, 0.055556|孙悟空, 0.111111|奔波儿灞, 0.055556]
万岁狐王 0.17766481837544773|[玉面公主, 0.166667|神通广大, 0.166667|猪八戒, 0.166667|牛魔王, 0.166667|罗刹女, 0.166667|土地, 0.166667]

```

```

猪八戒 46.749193802235
唐僧 41.86890947087324
孙悟空 40.70300311070657
沙僧 36.34408215774396
玉皇大帝 12.051919758225438
观音菩萨 9.946319988935974
如来佛祖 8.05385470270588
土地 7.7866412386711295
白龙马 6.773227344321923
神通广大 6.42628719694803
哪吒 4.676967301666322
雷公 4.39852151042744
五方揭谛 4.160328912594081
二十八宿 3.7737376014817654
诸天 3.4277652079352428

```

运行截图：

PageRank预处理：

**Hadoop Application application\_1656400433955\_2790**

**Kill Application**

User:	2021sh10
Name:	Task4-1: buildGraph
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
Queue:	root.2021s
FinalStatus Reported by AM:	SUCCEEDED
Started:	Tue Jul 05 14:50:29 +0800 2022
Elapsed:	17sec
Tracking URL:	<a href="#">History</a>
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:	119646 MB-seconds, 31 vcore-seconds

Show 20 entries

Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1656400433955_2790_000001	Tue Jul 5 14:50:29 +0800 2022	<a href="#">http://slave010:8042</a>	Logs	N/A

PageRank迭代：

**Hadoop Application application\_1656400433955\_2812**

**Kill Application**

User:	2021sh10
Name:	Task4-2: pageRankIter
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
Queue:	root.2021s
FinalStatus Reported by AM:	SUCCEEDED
Started:	Tue Jul 05 14:57:27 +0800 2022
Elapsed:	16sec
Tracking URL:	<a href="#">History</a>
Diagnostics:	

**Application Metrics**

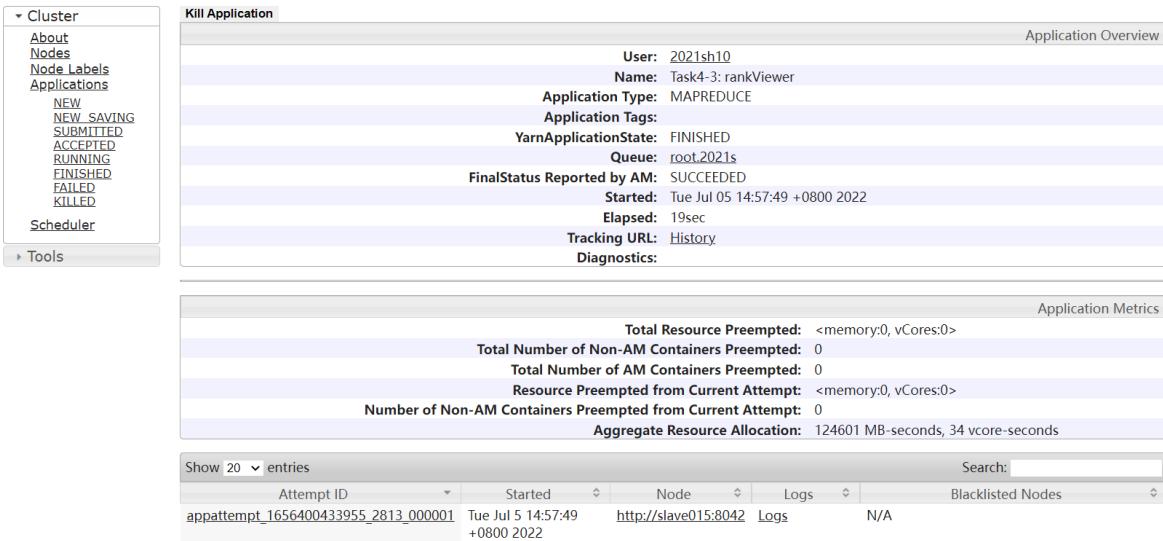
Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	113097 MB-seconds, 30 vcore-seconds

Show 20 entries

Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1656400433955_2812_000001	Tue Jul 5 14:57:27 +0800 2022	<a href="#">http://slave003:8042</a>	Logs	N/A

PageRank排序输出：

## Application application\_1656400433955\_2813



Application Overview

User: 2021sh10  
 Name: Task4-3: rankViewer  
 Application Type: MAPREDUCE  
 Application Tags: FINISHED  
 YarnApplicationState: FINISHED  
 Queue: root.2021s  
 FinalStatus Reported by AM: SUCCEEDED  
 Started: Tue Jul 05 14:57:49 +0800 2022  
 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:	124601 MB-seconds, 34 vcore-seconds

Show 20 entries Search:

Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1656400433955_2813_000001	Tue Jul 5 14:57:49 +0800 2022	http://slave015:8042	Logs	N/A

## Phase5：标签传播

Phase5的输出结果在/`user/2021sh10/xiyouji/Phase5Output`中，迭代中间在该目录下的`Data1/part-r-00000`至`Data20/part-r-00000`中，每一行格式为`TagId\t[Name_0, weight_0|Name_1, weight_1|...]`，`weight_i`表示`Name_i`分配到该`TagId`的概率；迭代完成后的输出结果在该目录下的`result/part-r-00000`中，每一行格式为`TagId \t Name`，即指示每一个`Name`最终分配到的`TagId`；最终输出结果在该目录下的`final_result/part-r-00000`，每一行格式为`TagId \t Name_0 Name_1 ...`，即指示每一个`TagId`对应的人名列表。

File - /user/2021sh10/xiyouji/Phase5Out... Page 1 of 22 ▶◀▶▶◀

1 [身本忧, 0.004348 猴精, 0.197535 打扫力士, 0.096866 接引归真佛, 0.133354 四值功曹, 0.221785 高太公, 0.029743 十洲三岛仙翁, 0.098050 杜如晦, 0.277869 运水力士, 0.096866 水母娘娘, 0.382033 贤善首佛, 0.199998 角木蛟, 0.192029 卞城王, 0.216024 虎精, 0.250000 诸天, 0.209316 程咬金, 0.277869 有来有去, 0.156357 牛蟠, 0.256127 宝象国王, 0.081867 巡海夜叉, 0.467189 风伯, 0.205701 黑鱼精, 0.141254 推云童子, 0.125359 千里眼, 0.108696 玉皇大帝, 0.000000 云里雾, 0.288665 辟寒大王, 0.182343 镇山太保, 0.267453 善游步佛, 0.133354 大鹏金翅雕, 0.094022 慧炬照佛, 0.133354 黑水河河神, 0.061006 复海大圣, 0.110682 风婆婆, 0.367524 鮀提督, 0.167147 狮奴, 0.206220 文殊菩萨, 0.062092 宝月光佛, 0.133354 火焰五光佛, 0.000000 蜘蛛精, 0.254721 翊圣真君, 0.000000 葛仙翁, 0.275708 梅山六兄弟, 0.057083 二十八宿, 0.176173 银头揭谛, 0.193283 松树, 0.177629 八宝金身, 0.078539 哪吒, 0.137271 妙音声佛, 0.953083 转轮王, 0.216024 腊梅, 0.179620 马三宝, 0.277869 鬼谷子, 0.000000 李渊, 0.213898 金吒, 0.045738 十二元辰, 0.184486 皂衣仙女, 0.098050 城隍, 0.197111 四大天师, 0.161649 丘弘济, 0.252240 楚江王, 0.228413 轸水蛇, 0.193283 如来佛祖, 0.000000 红百万, 0.233445 黄风怪, 0.234157 桧树, 0.182636 秦广王, 0.212035 春娇, 0.057140 薛仁贵, 0.277272 斗木獬, 0.193793 宝幢光王佛, 0.000000 王小二, 0.500000 金海光佛, 0.953083 五岳, 0.170431 李淳风, 0.277272 灵感大王, 0.159772 黄角大仙, 0.098050 善财龙女, 0.115803 南方南极观音, 0.098050 黄花观观主, 0.242833 横行蟹士, 0.167147 善财童子, 0.226178 永住金刚, 0.063198 驱神大圣, 0.107105 广目天王, 0.279217 陈清, 0.184882 锄树士, 0.000000 ...
---

```

3 一秤金
2 七政
2 七衣仙女
2 万圣公主
2 万圣龙王
3 万岁狐王
2 三千揭谛
1 三官
2 三星
2 三清
6 上清灵宝天尊
2 不坏尊王
1 与世同君
1 世静光佛
8 丘弘济

```

```

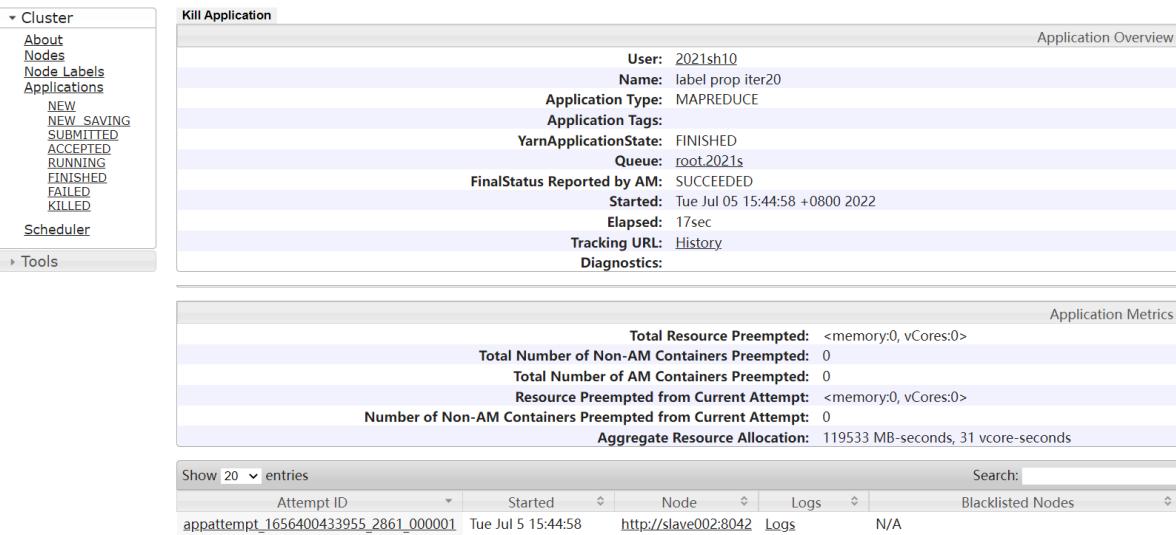
1 李淳风 李彪 光蕊 宝幢王佛 李右衙 药师琉璃光王佛 王珪 四大将 荡魔天尊 唐僧 广目天王 巡海夜叉 明月 大通光佛 大慧
力王佛 胡敬德 无量寿佛 金海光佛 唐太宗 金鱼 斑衣鳜婆 玉华王 玉兔 老虎精 猴精 慧幢胜王佛 慈力王佛 许敬宗 三官 四值功
曹 才光佛 姬娥仙子 高士廉 独角兕大王 野牛精 铜台府刺史 铁扇公主 特处士 观世灯佛 红孩儿 释迦牟尼佛 妙音声佛 燃灯上
古佛 熊罴精 寿星 熊山君 袁天罡 陈萼 蝎子精 寇铭 蜜蜂 马三宝 寇洪 清风 假唐僧 清净喜佛 程咬金 寇梁 海德光明佛 秦琼 真
将军 秦叔宝 法明 福星 凤仙郡郡侯上官氏 禄星 阿弥陀佛 水猿大圣 水母娘娘 与世同君 房玄龄 毗蓝婆菩萨 太阴 毗卢尸佛 虞
世南 相良 殷开山 段志贤 风婆婆 过去未来现在佛 虎精 白鹿 羿二郎 世静光佛 魏征 国师王菩萨 薛仁贵 白象 镇元子 杜如晦
东华帝君 圣婴大王 萧瑀 弥勒尊佛 李翠莲 李玉英 傅奕
2 东方崇恩圣帝 托塔天王 卞城王 危月燕 参水猿 房日兔 可韩丈人真君 意见欲 恶乌龟 东方朔 心月狐 御马监监丞 哪吒 广
力菩萨 善游步佛 平等王 平天大圣 布雾郎君 巴山虎 巨灵神 善财童子 镇都司 尾火虎 东海龙王 善财龙女 小钻风 室火猪 宝象
国王 四大天王 宝光佛 宋帝王 守山大神 孙悟空 孔雀大明王 金毛犼 金头揭谛 四大金刚 娄金狗 四帝 四海龙王 女土蝠 奔波儿
灞 奎木狼 四渎 太阳 金圣皇后 太上老君 东西星斗 天官 地藏王菩萨 大鹏金翅雕 大慈光佛 复海大圣 大势至菩萨 大力金刚 多
闻天王 夜游神 金刚大士 中央黄极 九头驸马 都市王 那罗延佛 九尾狐狸 通风大圣 通臂猿猴 九曜星 二十八宿 黑大王 龙女 万
圣公主 转轮王 身本忧 云程万里鹏 五岳 鲇鱼怪 五斗星君 赤尻马猴 黄风怪 五方五老 五方揭谛 诸天 五瘟 计都星 鼻火
猴 角木蛟 五百阿罗 西海龙王 西天极乐诸菩萨 井木犴 黄角大仙 鱼肚将 鼻嗅爱 穴金龙 蛟魔王 黄衣仙女 虚日鼠 虎将 虎先锋
丘官王 善财童子 乾池海金童善财童子 药王孙 由全羊 万圣龙王 二千揭谛 壬当甲 亥十化 丑平刚 丙十昌 丙辰奴 留心蛇 俗利中

```

运行截图：

迭代：

## Application application\_1656400433955\_2861



The screenshot shows the Hadoop Application Overview page for application\_1656400433955\_2861. The left sidebar includes Cluster (About, Nodes, Node Labels, Applications: NEW, NEW\_SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED), Scheduler, and Tools. The main content area has tabs for Application Overview and Application Metrics.

**Application Overview:**

- User: 2021sh10
- Name: label prop iter20
- Application Type: MAPREDUCE
- Application Tags:
- YarnApplicationState: FINISHED
- Queue: root.2021s
- FinalStatus Reported by AM: SUCCEEDED
- Started: Tue Jul 05 15:44:58 +0800 2022
- Elapsed: 17sec
- 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: 119533 MB-seconds, 31 vcore-seconds

Show 20 entries	Search:			
Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1656400433955_2861_000001	Tue Jul 5 15:44:58 +0800 2022	http://slave002:8042	Logs	N/A

迭代完成，收集结果：

## hadoop Application application\_1656400433955\_2863

Cluster

- About
- Nodes
- Node Labels
- Applications
  - NEW
  - NEW\_SAVING
  - SUBMITTED
  - ACCEPTED
  - RUNNING
  - FINISHED
  - FAILED
  - KILLED
- Scheduler

Tools

Kill Application

Application Overview	
User:	2021sh10
Name:	collect result
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
Queue:	root.2021s
FinalStatus Reported by AM:	SUCCEEDED
Started:	Tue Jul 05 15:45:19 +0800 2022
Elapsed:	16sec
Tracking URL:	<a href="#">History</a>
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:	117742 MB-seconds, 31 vcore-seconds

Show 20 entries Search:

Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1656400433955_2863_000001	Tue Jul 5 15:45:19 +0800 2022	<a href="http://slave009:8042">http://slave009:8042</a>	Logs	N/A

输出最终结果：

## hadoop Application application\_1656400433955\_2865

Cluster

- About
- Nodes
- Node Labels
- Applications
  - NEW
  - NEW\_SAVING
  - SUBMITTED
  - ACCEPTED
  - RUNNING
  - FINISHED
  - FAILED
  - KILLED
- Scheduler

Tools

Kill Application

Application Overview	
User:	2021sh10
Name:	combine result
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
Queue:	root.2021s
FinalStatus Reported by AM:	SUCCEEDED
Started:	Tue Jul 05 15:45:38 +0800 2022
Elapsed:	16sec
Tracking URL:	<a href="#">History</a>
Diagnostics:	

Application Metrics

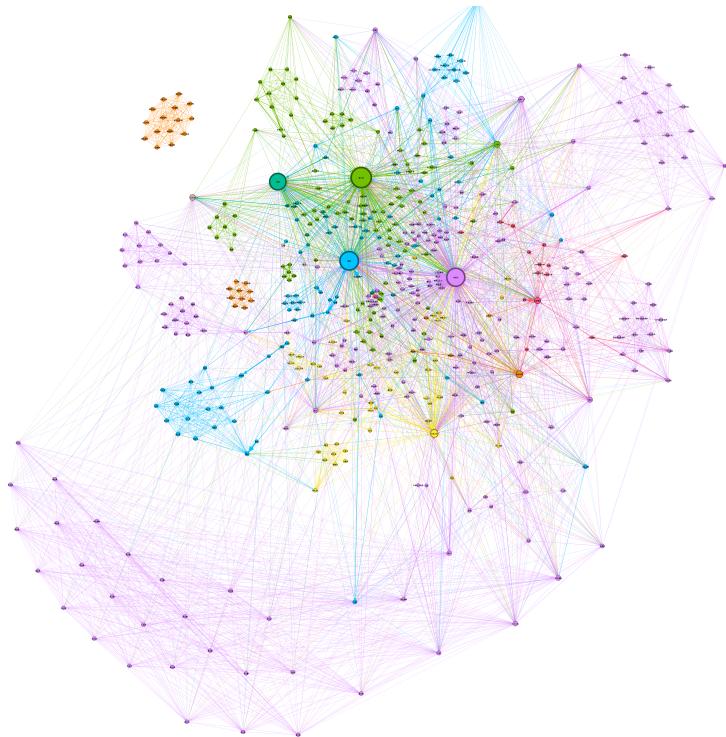
Total Resource Preempted:	<memory:0, vCores:0>
Total Number of Non-AM Containers Preempted:	0
Total Number of AM Containers Preempted:	0
Resource Preempted from Current Attempt:	<memory:0, vCores:0>
Number of Non-AM Containers Preempted from Current Attempt:	0
Aggregate Resource Allocation:	116472 MB-seconds, 31 vcore-seconds

Show 20 entries Search:

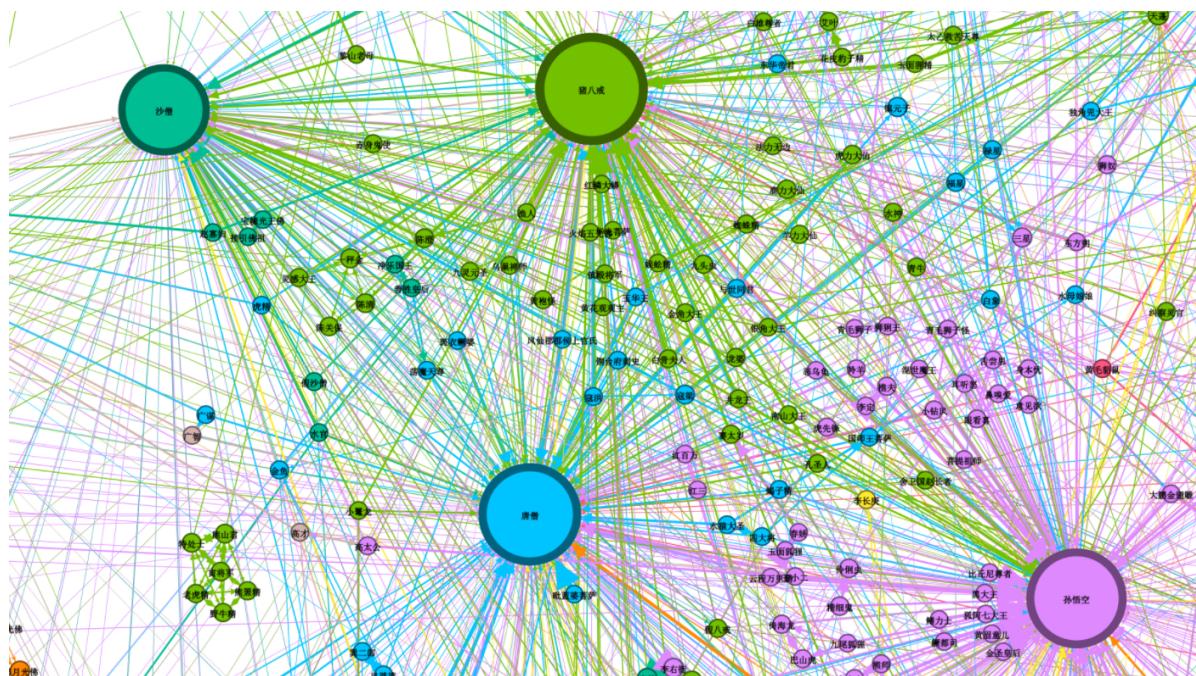
Attempt ID	Started	Node	Logs	Blacklisted Nodes
appattempt_1656400433955_2865_000001	Tue Jul 5 15:45:38 +0800 2022	<a href="http://slave015:8042">http://slave015:8042</a>	Logs	N/A

## Phase6：可视化

利用 Gephi 软件完成可视化处理，由于本实验采用的标签传播算法是半监督算法，因此多次改变初始的标签划分——测试，以得到更直观的可视化结果；最终确定初始化标签划分为 [所有主要人物，即师徒四人+白龙马+如来佛祖+观音菩萨+玉皇大帝]，并且设置按照tag划分节点颜色、按照pr值进行节点大小排序、边携带点颜色，得出的可视化结果如下：



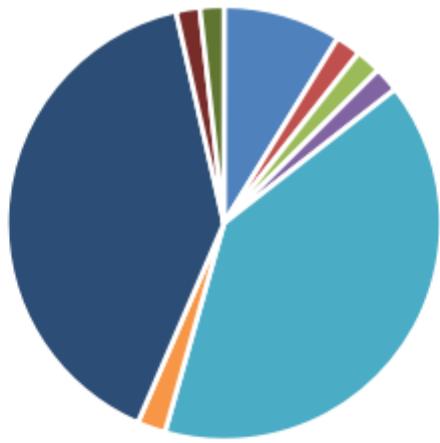
主体的局部视图：



可以观察到，沙僧、猪八戒、唐僧、孙悟空这四位角色的节点突出，并且关系网络复杂，与西游记主角的身份符合。

## 性能分析

程序在集群上运行在各部分所花费的时间分布如下：



■ Phase1 ■ Phase2 ■ Phase3 ■ Phase4-1 ■ Phase4-2  
■ Phase4-3 ■ Phase5-1 ■ Phase5-2 ■ Phase5-3

注意到花费时间最长的是两个迭代过程：即PageRank迭代和标签传播迭代（均迭代了20轮）；除此之外Phase1对于全篇文本的预处理花费时间也较长，原因是引入分词处理以及人名比对；其它阶段时间分布平均，说明处理规模类似，没有明显的性能瓶颈。