# 大数据

# 综合处理实验报告

实验课题 -----《金庸的江湖》

131220044 查鹏（组长）

131220034 许金强

**目录**

1. 实验目标...........................................................................................3
2. 算法原理...........................................................................................3
3. 实验流程...........................................................................................9
4. 实验优化.........................................................................................18
5. 程序运行的截图.............................................................................20
6. 小组成员的分工.............................................................................30
7. 附录................................................................................................30

## 实验目标

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课程设计。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。

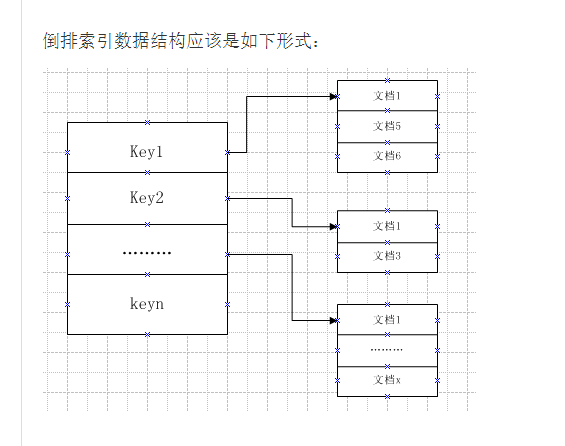
## 算法原理

**2.1 倒排索引算法**

倒排索引源于实际应用中需要根据属性的值来查找记录。这种索引表中的每一项都包括一个属性值和具有该属性值的各记录的地址。由于不是由记录来确定属性值，而是由属性值来确定记录的位置，因而称为倒排索引(inverted index)。带有倒排索引的文件我们称为倒排索引文件，简称倒排文件(inverted file)。

倒排文件（倒排索引），索引对象是文档或者文档集合中的单词等，用来存储这些单词在一个文档或者一组文档中的存储位置，是对文档或者文档集合的一种最常用的索引机制。

搜索引擎的关键步骤就是建立倒排索引，倒排索引一般表示为一个关键词，然后是它的频度（出现的次数），位置（出现在哪一篇文章或网页中，及有关的日期，作者等信息），它相当于为互联网上几千亿页网页做了一个索引，好比一本书的目录、标签一般。读者想看哪一个主题相关的章节，直接根据目录即可找到相关的页面。不必再从书的第一页到最后一页，一页一页的查找。



如上图，建立各个key与所处文档之间的联系，就可以对每一个key都能找到它对应的文档，这就是倒排索引的含义。

**2.2 单词共现算法**

单词共现算法是MapReduce可以用来高效解决的一大类问题的抽象化描述。在自然语言处理以及简历语料库上也有着重要的应用。其目的是在海量语料库中发现在固定窗口内单词a和单词b共同出现的频率，从而构建单词共现矩阵，这样的矩阵可以是对称的，也可以是不对称的，这要看具体的应用。

 这种抽象化的任务的有效解决在实际生活中有着很多的应用。例如电子商家希望发现不同物品被同时购买的情况以便有效安排货物的摆放位置；同时对信息检索领域同义词词典的构建以及文本挖掘等都有着重要的实际应用价值。

设有一个英文句：we are not what we want to be but at least we are not what we used to be.

设共现窗口定义为连续出现的两个单词，则表中给出了上局英文的共现矩阵。

示例英文语句的共现矩阵

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | we | are | not | what | we | want | to | be | but | at | least | used |
| we |  | 2 |  |  |  | 1 |  |  |  |  |  | 1 |
| are | 2 |  | 2 |  |  |  |  |  |  |  |  |  |
| not |  | 2 |  | 2 |  |  |  |  |  |  |  |  |
| what |  |  | 2 |  |  |  |  |  |  |  |  |  |
| want | 1 |  |  |  |  |  | 1 |  |  |  |  |  |
| to |  |  |  |  |  | 1 |  | 1 |  |  |  | 1 |
| be |  |  |  |  |  |  | 1 |  |  |  |  |  |
| but |  |  |  |  |  |  |  |  |  | 1 |  |  |
| at |  |  |  |  |  |  |  |  | 1 |  |  |  |
| least |  |  |  |  |  |  |  |  |  |  |  |  |
| used | 1 |  |  |  |  |  | 1 |  |  |  |  |  |

**2.3 PageRank算法**

PageRank，网页排名，又称网页级别、Google左侧排名或佩奇排名，是一种由根据[网页](http://baike.baidu.com/view/828.htm" \t "http://baike.baidu.com/_blank)之间相互的[超链接](http://baike.baidu.com/view/743.htm" \t "http://baike.baidu.com/_blank)计算的技术，而作为网页排名的要素之一，以[Google](http://baike.baidu.com/view/105.htm" \t "http://baike.baidu.com/_blank)公司创办人[拉里·佩奇](http://baike.baidu.com/view/251897.htm" \t "http://baike.baidu.com/_blank)（Larry Page）之姓来命名。Google用它来体现网页的相关性和重要性，在[搜索引擎优化](http://baike.baidu.com/subview/7147/5109949.htm" \t "http://baike.baidu.com/_blank)操作中是经常被用来评估网页优化的成效因素之一。Google的创始人拉里·佩奇和[谢尔盖·布林](http://baike.baidu.com/view/615669.htm" \t "http://baike.baidu.com/_blank)于1998年在[斯坦福大学](http://baike.baidu.com/view/13725.htm" \t "http://baike.baidu.com/_blank)发明了这项技术。

PageRank通过网络浩瀚的超链接关系来确定一个页面的等级。Google把从A页面到B页面的链接解释为A页面给B页面投票，Google根据投票来源（甚至来源的来源，即链接到A页面的页面）和投票目标的等级来决定新的等级。简单的说，一个高等级的页面可以使其他低等级页面的等级提升。

PageRank让链接来"投票"

一个页面的“得票数”由所有链向它的页面的重要性来决定，到一个页面的超链接相当于对该页投一票。一个页面的PageRank是由所有链向它的页面（“链入页面”）的重要性经过递归算法得到的。一个有较多链入的页面会有较高的等级，相反如果一个页面没有任何链入页面，那么它没有等级。  
　　2005年初，Google为网页链接推出一项新属性[nofollow](http://baike.baidu.com/view/1584081.htm" \t "http://baike.baidu.com/_blank)，使得网站管理员和网站作者可以做出一些Google不计票的链接，也就是说这些链接不算作"投票"。[nofollow](http://baike.baidu.com/view/1584081.htm" \t "http://baike.baidu.com/_blank)的设置可以抵制评论垃圾。

假设一个由4个页面组成的小团体：A，B，C和D。如果所有页面都链向A，那么A的PR（PageRank）值将是B，C及D的Pagerank总和。



继续假设B也有链接到C，并且D也有链接到包括A的3个页面。一个页面不能投票2次。所以B给每个页面半票。以同样的逻辑，D投出的票只有三分之一算到了A的PageRank上。



换句话说，根据链出总数平分一个页面的PR值。



最后，所有这些被换算为一个百分比再乘上一个系数。由于“没有向外链接的页面”传递出去的PageRank会是0，所以，Google通过数学系统给了每个页面一个最小值：

IMG_259

说明：在Sergey Brin和Lawrence Page的1998年原文中给每一个页面设定的最小值是1-d，而不是这里的

(1-d)/N。 所以一个页面的PageRank是由其他页面的PageRank计算得到。Google不断的重复计算每个页面的PageRank。如果给每个页面一个随机PageRank值（非0），那么经过不断的重复计算，这些页面的PR值会趋向于稳定，也就是收敛的状态。这就是搜索引擎使用它的原因。

**2.4 标签传播算法（LPA）**

标签传播算法（LPA）是由Zhu等人于2002年提出，它是一种基于图的半监督学习方法，其基本思路是用已标记节点的标签信息去预测未标记节点的标签信息。利用样本间的关系建立关系完全图模型，在完全图中，节点包括已标注和未标注数据，其边表示两个节点的相似度，节点的标签按相似度传递给其他节点。标签数据就像是一个源头，可以对无标签数据进行标注，节点的相似度越大，标签越容易传播。由于该算法简单易实现，算法执行时间短，复杂度低且分类效果好，引起了国内外学者的关注，并将其广泛地应用到多媒体信息分类、虚拟社区挖掘等领域中。

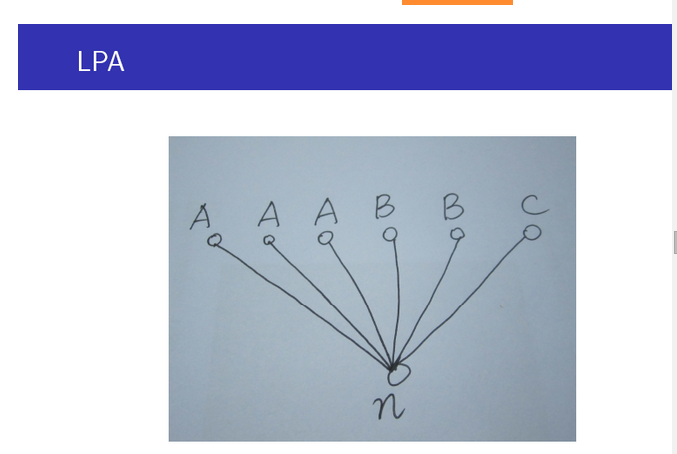
根据LPA算法基本理论，每个节点的标签按相似度传播给相邻节点，在节点传播的每一步，每个节点根据相邻节点的标签来更新自己的标签，与该节点相似度越大，其相邻节点对其标注的影响权值越大，相似节点的标签越趋于一致，其标签就越容易传播。在标签传播过程中，保持已标注数据的标签不变，使其像一个源头把标签传向未标注数据。最终，当迭代过程结束时，相似节点的概率分布也趋于相似，可以划分到同一个类别中，从而完成标签传播过程。

算法过程

第一步: 为所有节点指定一个唯一的标签；

第二步: 逐轮刷新所有节点的标签，直到达到收敛要求为止。对于每一轮刷新，节点标签刷新的规则如下:

        对于某一个节点，考察其所有邻居节点的标签，并进行统计，将出现个数最多的那个标签赋给当前节点。当个数最多的标签不唯一时，随机选一个。



## 如上图（一个图的局部），对于节点n，与它相邻的一共有六个节点，它们的标签分别为A、A、A、B、B、C，根据标签传播的理论，此时n的标签应该为与它相邻的节点中标签最多的那个标签，如果最多的标签不止一个，随机取一个，所以节点n的标签应该为A。其它的节点也是如此这般，最终整个图的标签趋于稳定。

## 三、实验流程

## 实验总的流程图

**分词**

**人物同现分析**

**人物关系构建与特征归一化**

**PageRank值的计算**

**标签传播算法**

## 3.1 数据预处理

本任务的主要工作是从金庸小说文本中的每一段，抽取出出现的人名，舍弃其它内容，为后面的人物同现分析做准备。

Setup:**输入**：金庸武侠小说人名

**输出**：stopWords 用户自定义词典

Map:**输入**：key 文本内容偏移量 value:文本内容的一行

**输出**：key 人名+”，”+人名。。。 value:””

Reduce:**输入**：key 人名+”，”+人名。。。 value:””

**输出**：key 人名+”，”+人名。。。 value:””

具体思路：

setup中先把输入文件（金庸武侠小说中的人名停词表）中的所有内容存储到stopWords中（作为用户自定义词典）；

map阶段读取武侠小说中的每一行，用pdf提供的Ansj\_seg分词工具分词，得到分到的词表，然后遍历这个词表，判断词语的词性是否是名字（这个用到分词工具里的词语属性，名字的属性是nr），对每一个是名字的词语再判断是否存在于之前的停词表中，如果存在，说明这是一个金庸小说中的人物名字，输出，否则继续遍历。

reduce阶段不做特殊操作，直接输出，作为下一个任务输入。

## 3.2 特征抽取：人物同现统计

本任务的主要完成基于单词同现算法的人物同现统计。在人物同现分析中，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。

Map：**输入**：key 任务一的输出文本偏移量 value 人名A，人名B，。。。。。。

**输出**：key 人名A，人名B value 1

。。。。。。

Reduce:**输入**：key 人名A，人名B value 1,1,1。。。。。。

**输出**：key 人名A，人名B value sum（输入的1的总数）

具体思路：

map阶段直接读取一行中的所有出现的人名，然后看看两两组合的情况。map阶段输出key（人名A,人名B）,value 1。

reduce阶段由于经过combiner，相同key的输入已经全部集中到一起了，所以只要把相同key的输入的value相加就行了，比如（人名A,人名B）1,1,1 直接变成 （人名A,人名B）3输出。

**3.3 特征处理：人物关系构建与特征归一化**

为了使后面的方便分析，还需要对共现次数进行归一化处理：将共现次数转换为共现概率。

Map：**输入**：任务二的输出文本偏移量 value 人名A，人名B sum（人名A、B共现次数）

**输出**：key 人名A value 人名B，sum（人名A、B共现次数）

SumCombiner:**输入**：key 人名A value 人名B，sum（人名A、B共现次数）|人名C，sum（人名A、C共现次数）|。。。。。。

**输出**：key 人名A value 人名B，sum（人名A、B共现次数）|人名C，sum（人名A、C共现次数）|。。。。。。|SUM（之前所有sum的总和）

Reduce:**输入**：key 人名A value 人名B，sum（人名A、B共现次数）|人名C， sum（人名A、C共现次数）|。。。。。。|SUM（之前所有sum的总和）

**输出**：key 人名A value 人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

具体思路：

map阶段对输入进行处理，因为要知道概率，必须知道总数，比如（A,B）3，我们必须知道所有（A，...）...，把所有总数给求出来，然后才能知道概率，所以，我们以A作为key，（B，3）作为value输出，sumCombiner阶段，同样所有以A为key的输出全部集中到一起了，我们直接计算总数，最后输出key为A，value为B，3|...|sum。

reduce阶段，取value的最后一项就是总数，然后value前面的每一项的个数除以总数就是概率，输出key是A，value是 B，0.333|...|。

**3.4 数据分析：基于人物关系图的PageRank计算**

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

第一阶段：

map: **输入**:key 任务三的输出文本偏移量 value 人名A 人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**：key 人名B value 人名A，pagerank\*sum（人名A、B共现次数）/SUM

。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

Reduce: **输入**：key 人名B value 人名A，pagerank\*sum（人名A、B共现次数）/SUM

。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**：key value pagerank（新的） links

第二阶段：

map: **输入**:第一阶段输出文本偏移量 value 人名A pagerank 人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**：key 人名B value 人名A，pagerank\*sum（人名A、B共现次数）/SUM

。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

Reduce: **输入**：key 人名B value 人名A，pagerank\*sum（人名A、B共现次数）/SUM

。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**：key value pagerank（新的） links

具体思路：

第一阶段：map阶段，输入是value：A B，0.333|...|，处理该value，定义一个初始pagerank 1.0，对每一个value输出为key是B，value是 A，0.333\*1.0，最后还要输出key是A，value是 #B，0.333|...|，（这个是为了之后的pageRank做准备的）。

reduce阶段定义一个links，对每一个value，先判断是否是以#开头，如果是，links为value从#开始之后的，就是B,0.333|...|，如果不是#开头的，计算pagerank为 所有key为B的输入的值的总和（key为B，value为A，0.333\*1.0等等）最后，pagerank = 0.15 + 0.85 \* pagerank，输出 key，value是 pagerank links。

第二阶段，输入为 value为A 0.25 B,0.333|...|，注意这里和第一阶段的差别，多了一个0.25 pagerank值，这就是经过第一阶段的预处理得到的输入。map阶段处理value，同上，只是这里不要定义一个pagerank 1.0了，而是调用输入的 0.25，输出 B A，0.25\*0.333|...|，

reduce阶段同上，最后输出 key pagerank links。

**3.5 数据分析：在人物关系图上的标签传播（选做）**

标签传播（Label Propagation）是一种半监督的图分析算法，他能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（Community Detection）。

第一阶段：

Map：**输入**：key 任务三的输出文本偏移量 value 人名A 人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**： key 人名B value 人名A(A的初始标签)

key 人名C value 人名A

。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

Reduce：**输入**：key 人名B value 人名A，人名C，。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**：key value label links

第二阶段：

Map：**输入**：key 第一阶段的输出文本偏移量 value 人名A label\_A 人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**： key 人名B value label\_A

key 人名C value label\_A

。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

Reduce：**输入**：key 人名B value label1，label2，。。。。。。

key 人名A value #人名B，sum（人名A、B共现次数）/SUM|人名C，sum（人名A、C共现次数）/SUM|。。。。。。

**输出**：key value label links

具体思路：

第一阶段：map阶段：输入为value:

A B,0.222|C,0.333|...|，对每一个value，输出key是B，value是A（作为B的初始标记候选），等等，最后再像pagerank那样输出key是A，value是#B,0.222|....|。

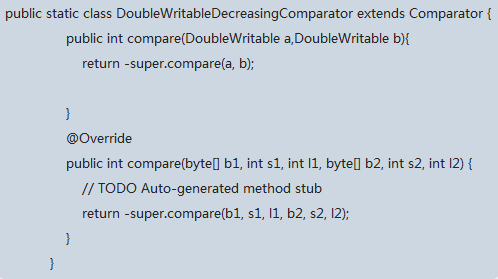
Reduce阶段定义links和label，对map的每一个输入，先判断是否以#开头，如果是，就把links赋值为B,0.222|...|，如果不是，（就是形如B A，C，D...）比如是A，判断一下以B为key，A有没有出现过，如果没有，就记A,1，否则就把A的值加一（用到了map.put(),map.get()）,这样遍历完一个key之后，只要找到其中的值最大的就行了，最大的一样大随机取一个，最后输出key，value：label links。

第二阶段：同上，不过这个时候输入就比第一阶段多一个label了，比如为value：A C B,0.222|C,0.333|...|。map阶段：输入为value: A C B,0.222|C,0.333|...|，对每一个value，输出key是B，value是C（这里就不是初始标记，而是输入中的label，即所有的value都输出这个C label）等等，最后再像pagerank那样输出key是A，value是#B,0.222|....|。，0.333|...|。

reduce阶段同上，最后输出 key label links。

**3.6 分析结果整理（选做）**

这个结果整理对应的是rankSort.java和LPASort.java，由于MapReduce的机制问题（根据主键值自动排序），输出结果自动排序，所以分析结果整理就自动完成，很简单，不过默认的是从小到大排序，这里稍微修改一下，变成从大到小排序就行了。

（具体实现：重载了map的key输出排序函数compare函数：

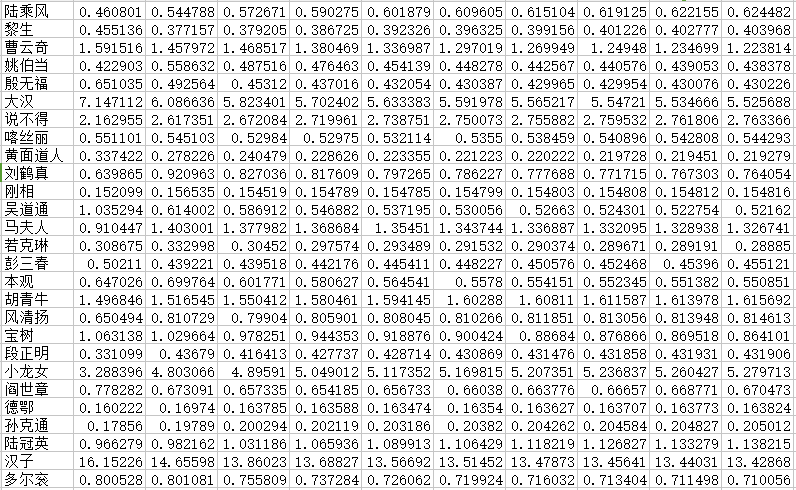
最后结果输出为负的）

1. **实验优化**
2. 人名分词、人物共现分析使用了combiner，这样就导致它们的reduce可以几乎不用做什么别的事了，简化了代码。
3. 我们的pagerank和标签传播部分分了两个部分，有pre\_pagerank和preLPA，作用是为了方便迭代，在原来的key和value基础上加上了pagerank初始值以及label初始值。

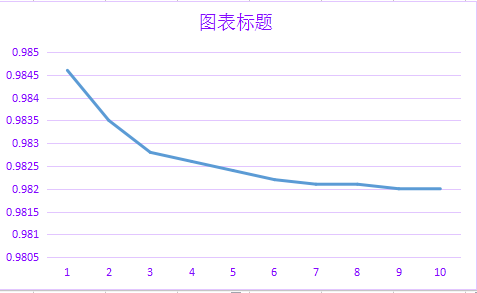
3、标签传播过程中，当两个名字的标签一样时，一开始是随机选一个，后来经过优化，改成了选两个里面pagerank值更大的一个，这样就会导致最后的标签更容易使主角了，使得最后结果更加精确。

4、关于确定迭代是否终止，另外写了一个小程序，判断每一次迭代得到的所有人的pagerank值的平均值之间的差，最后发现当迭代到一定程度之后，差值稳定在一个值上下，就判断迭代终止，这样大大减少了运算量。

下图就是每一个名字的pagerank在每一次迭代之后的值的变化情况：

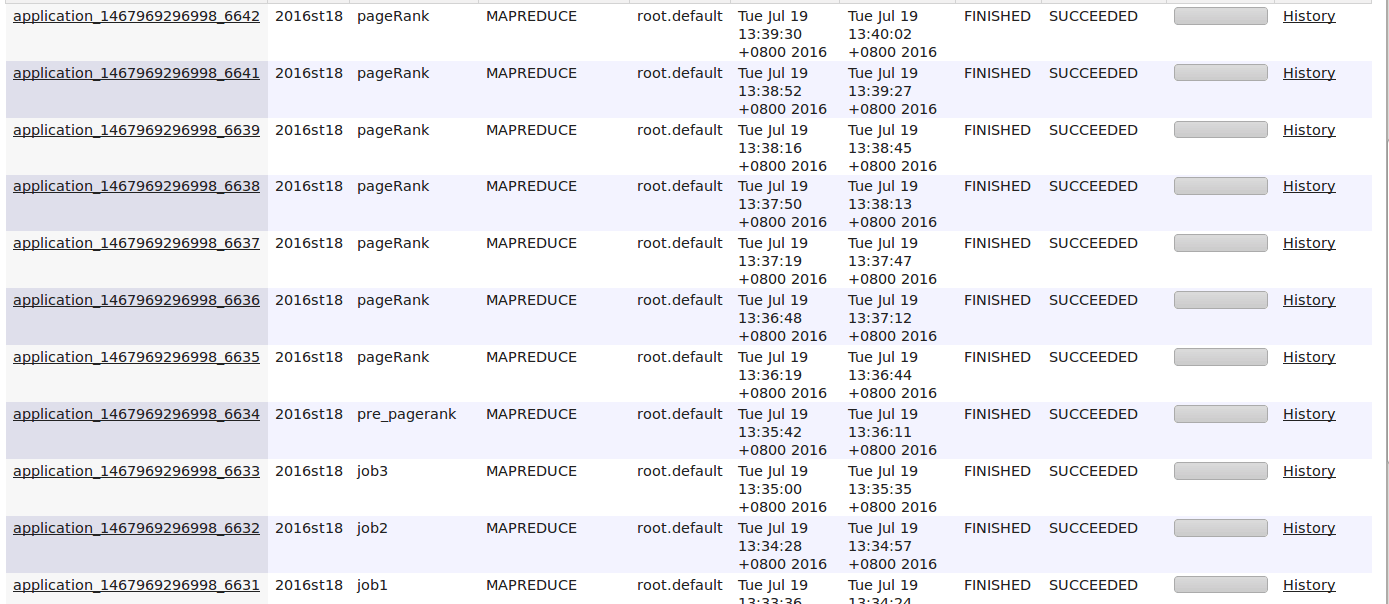


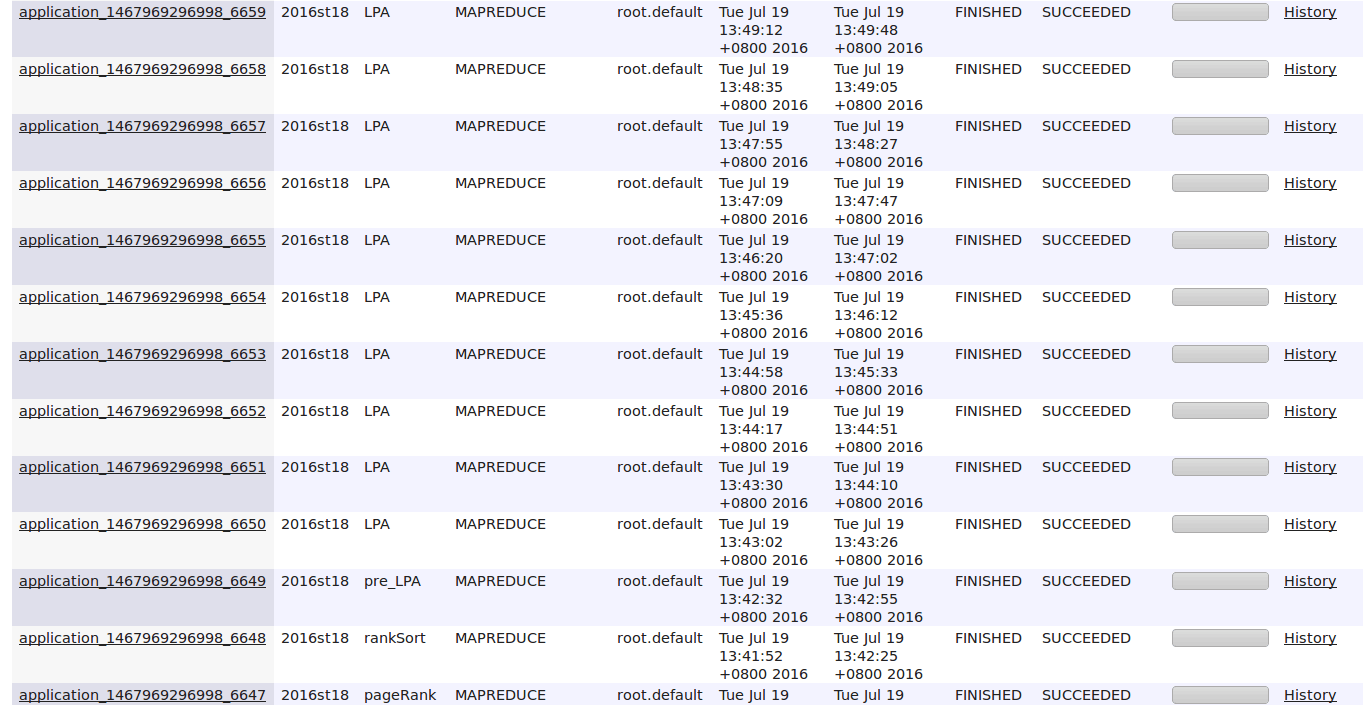
下图是每次迭代，所有人的pagerank值的平均值的变化情况，可以看出，在迭代9次之后，平均值趋于稳定于0.9820，迭代终止。

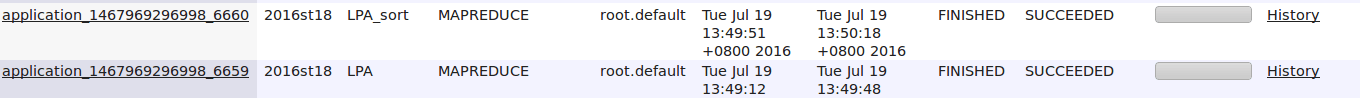


**五、程序运行的截图**

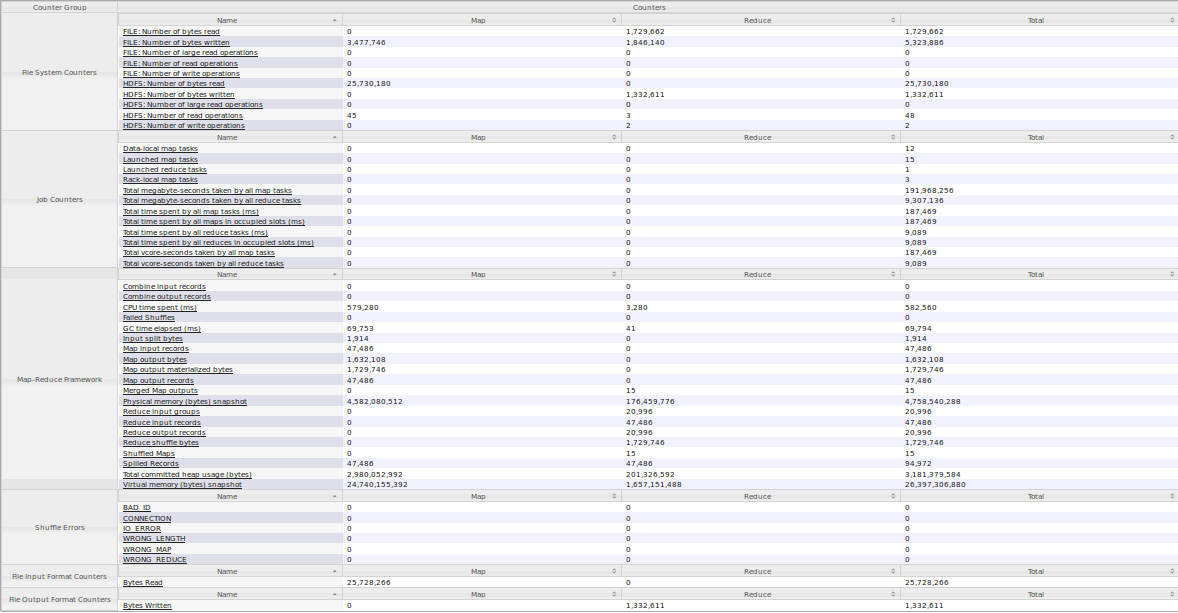
系集群上的运行截图

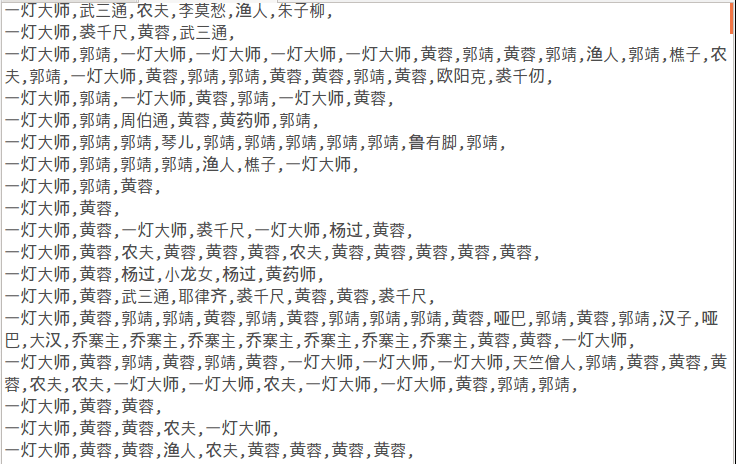




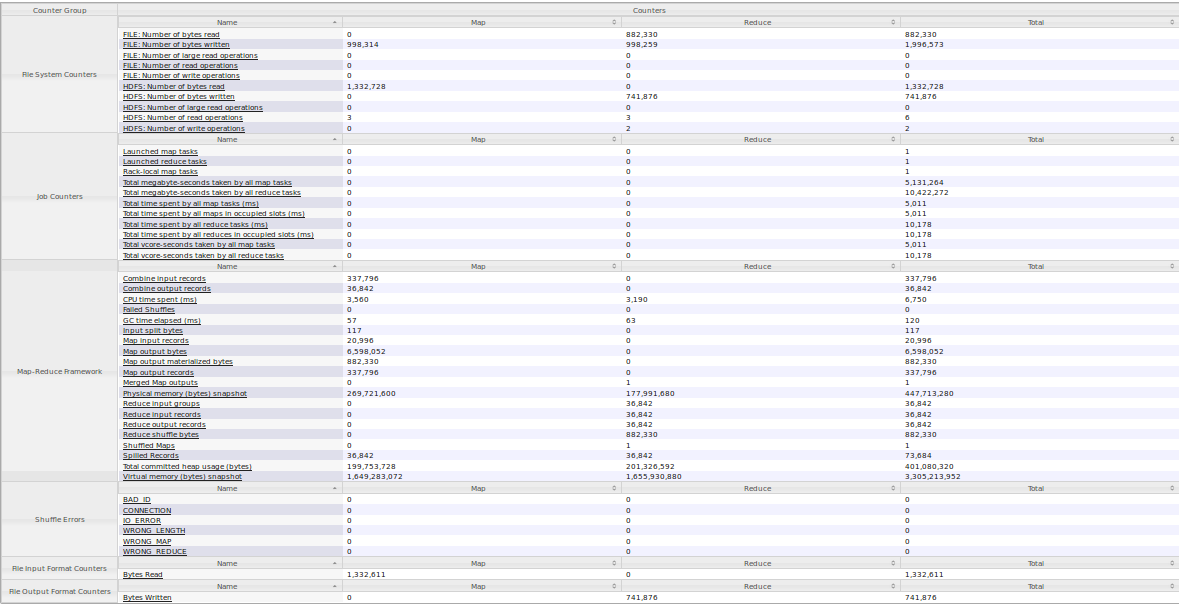


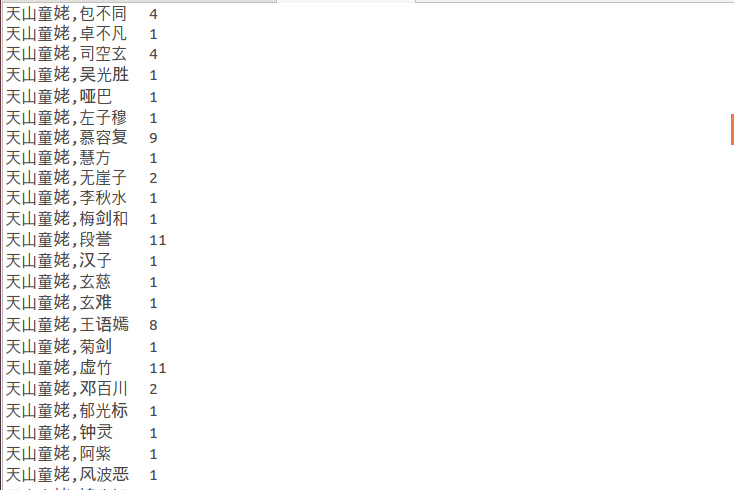
人名分词的截图（总共运行了48s）



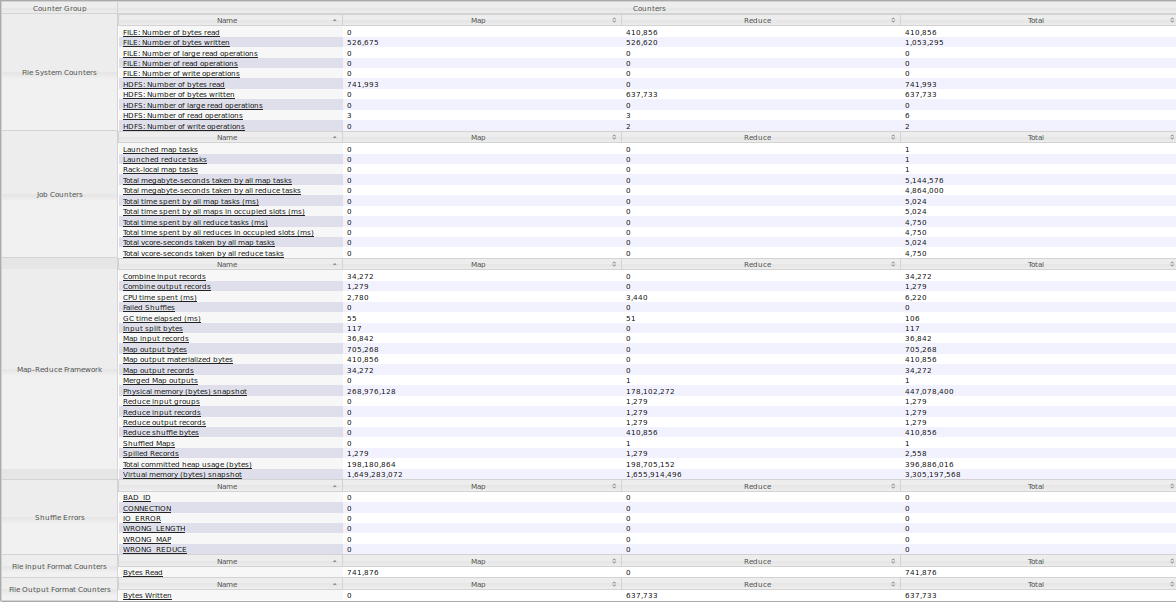


人物同现统计的截图（总共运行了29s）



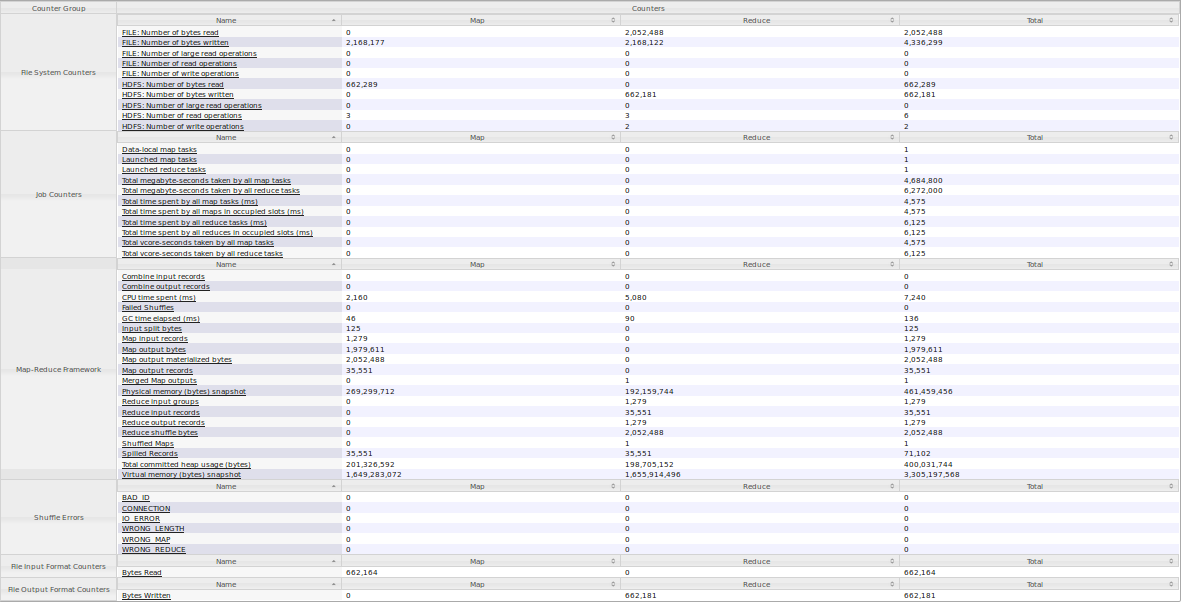


人物关系图构建与特征归一化的截图（总共运行了35s）

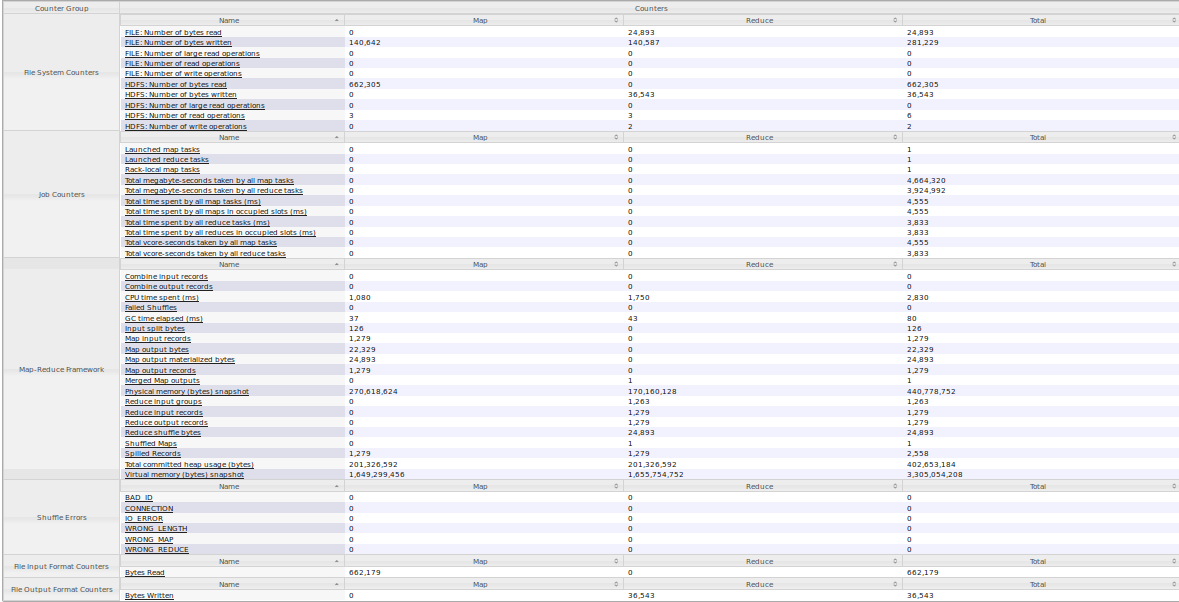




基于人物关系图的pagerank计算的截图（总共运行了29+366 = 395s）

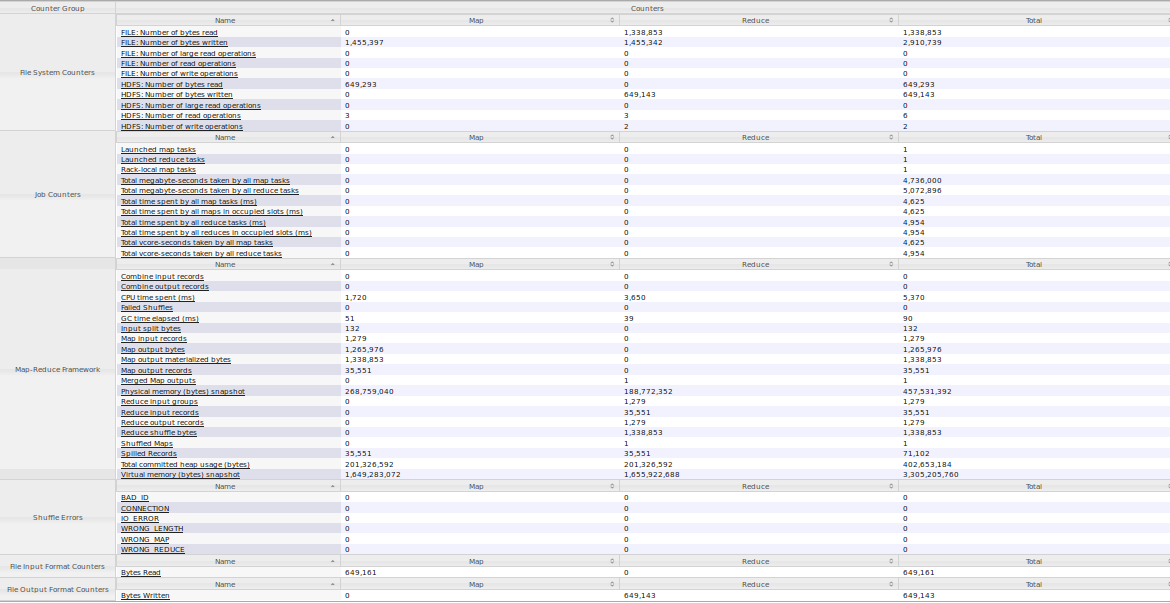


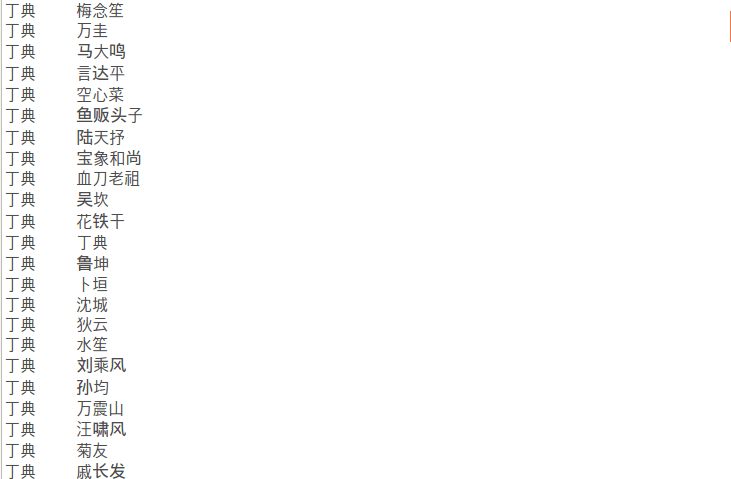
pagerank的排序截图



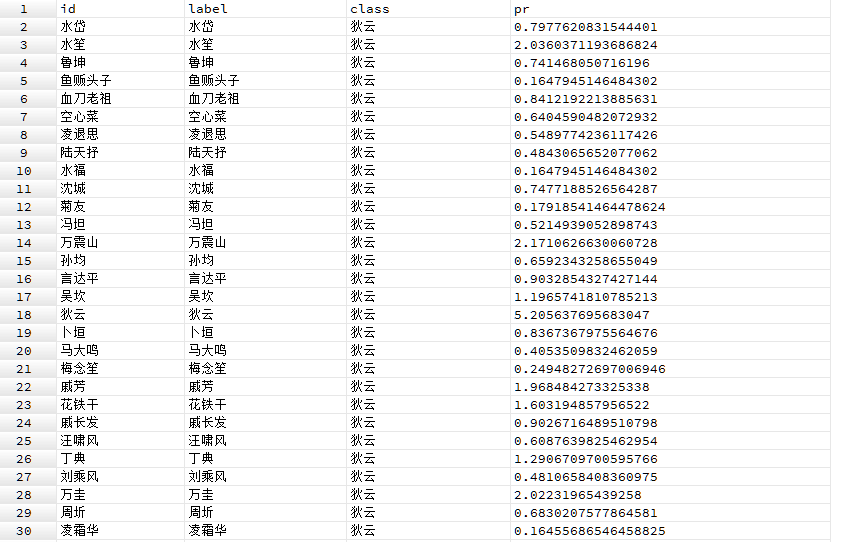


在人物关系图上的标签传播的截图（总共运行了23+376 = 399s）

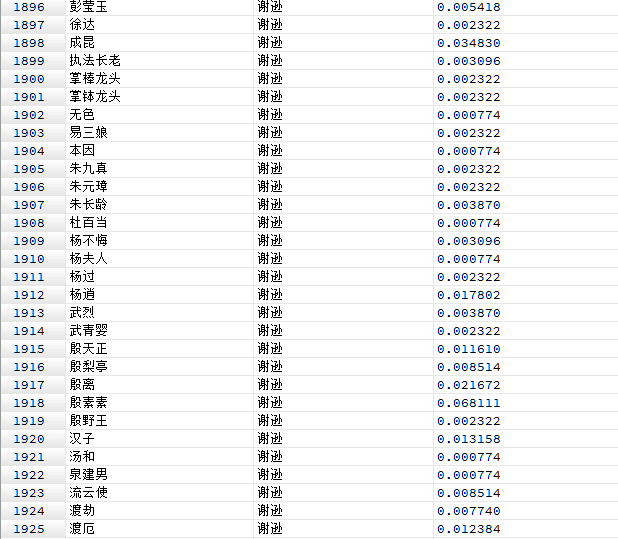




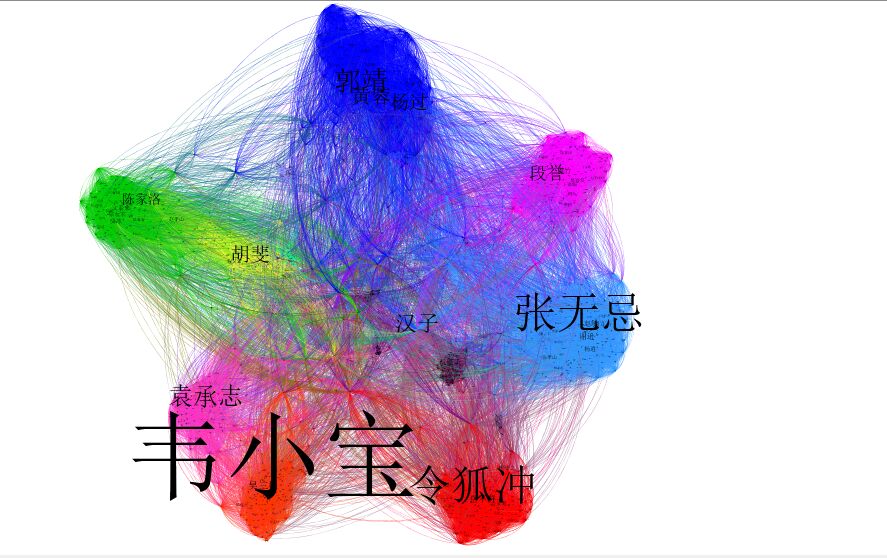
Gephi中的点文件截图



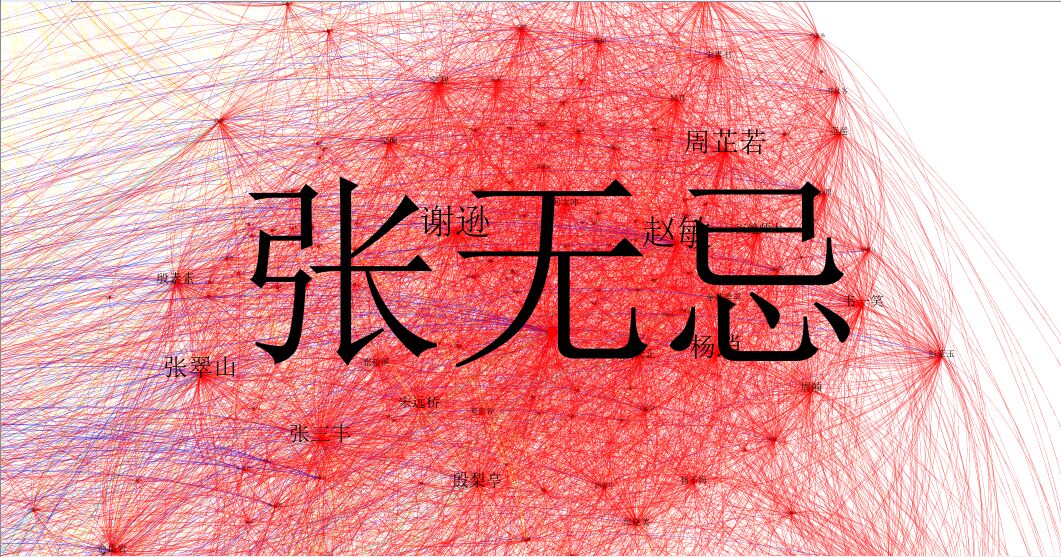
Gephi中的边文件截图



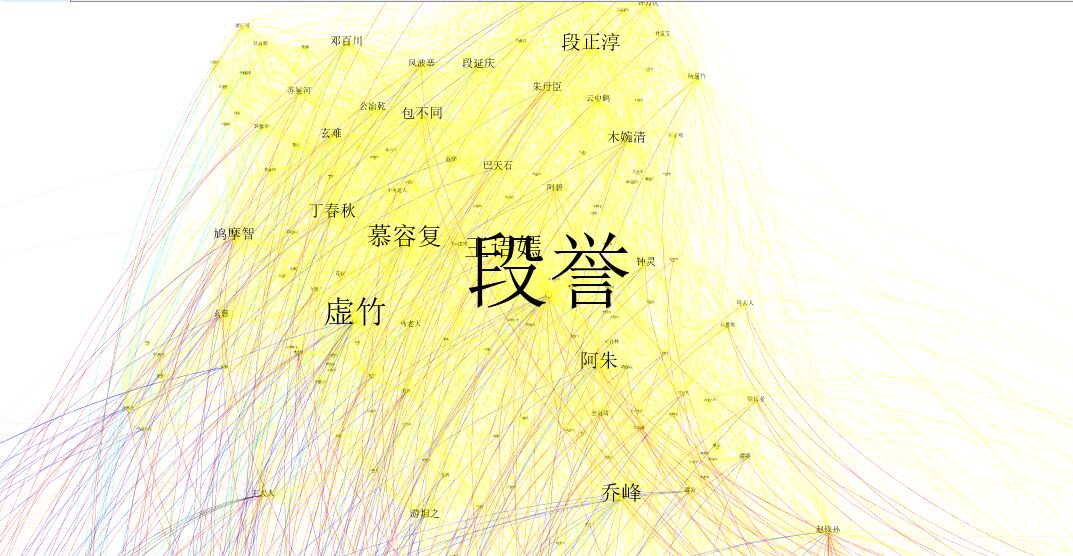
在人物关系图上的标签传播的总的结果截图

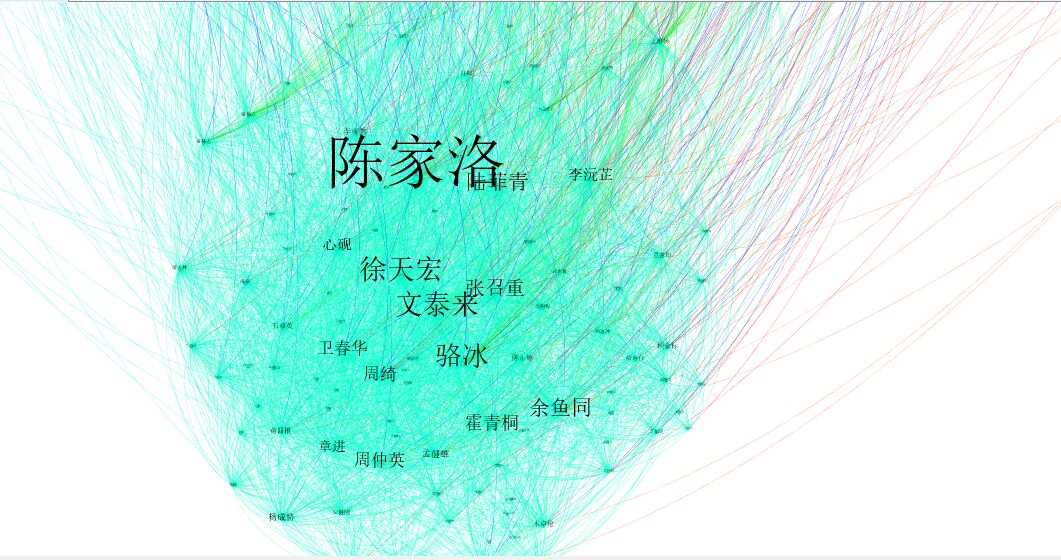


在人物关系图上的标签传播的结果细节截图









**六、小组成员分工**

|  |  |
| --- | --- |
| 查鹏（组长） | 主要负责代码编写：jobOne，jobTwo，jobThree，pagerank和标签传播代码的编写 |
| 许金强 | 主要负责开题报告的编写，gephi绘图、实验结果数据分析和实验报告的编写 |

1. **附录**

在统计任务同现次数过程中我们曾经犯了一个错误，就是比如一行出现的人名是A、B、B、C、D，两两组合，只是判断不能自己和自己组合，并没有考虑重复出现这种情况，导致诸如（A,B）出现两次，这也直接影响了后面PageRank值的计算，导致最后结果不正确，后来我们用java特有机制set，首先，把输入的一行人名直接存储在set里，这里set会自动去除重复的，然后对set里的人名进行两两配对，这样就解决了之前的问题。