

# 大数据计算基础大作业

120L022109 李世轩

2022-12-19

# 目录

<b>1 第一章绪论</b>	<b>4</b>
1.1 大作业题目——Twitter 影响力人物评选系统 . . . . .	4
1.2 问题背景 . . . . .	4
1.3 问题挑战 . . . . .	5
<b>2 方法框架</b>	<b>5</b>
2.1 当前研究现状 . . . . .	5
2.2 本文方法 . . . . .	6
<b>3 算法 0</b>	<b>6</b>
3.1 程序代码 . . . . .	7
3.2 程序结果 . . . . .	9
<b>4 算法 1</b>	<b>9</b>
4.1 程序代码 . . . . .	10
4.2 实验结果 . . . . .	13
<b>5 算法 2</b>	<b>13</b>
5.1 提前处理 . . . . .	14
5.1.1 代码 . . . . .	14
5.1.2 结果 . . . . .	16
5.2 PageRank 算法 . . . . .	16
5.2.1 代码 . . . . .	16
5.2.2 结果 . . . . .	18
<b>6 算法 3</b>	<b>19</b>
6.1 程序代码 . . . . .	19
6.2 实验结果 . . . . .	24
<b>7 结果分析</b>	<b>24</b>
<b>8 MRS 平台实现</b>	<b>25</b>
<b>9 结论</b>	<b>25</b>
<b>10 参考文献</b>	<b>28</b>

目录	3
<b>11 附录</b>	<b>29</b>
11.1 排序程序 . . . . .	29

# 1 第一章绪论

## 1.1 大作业题目——Twitter 影响力人物评选系统

Twitter 是提供当下全球实时事件和热议话题讨论的平台。看新闻成为了许多人茶余饭后的休闲方式。小林最近遇到了一个难题，由于公司最近打算举办一系列的品牌推广活动，作为直接执行人的他需要联系一些有知名度的人物进行商业洽谈。他想到了一个好点子，与 Twitter 上影响力大的用户进行合作，对公司产品进行推广。苦于洽谈人选的他希望得到你的帮助找到商业洽谈人选。

要求：

1. 请你设计三种算法为 Twitter 中的每位用户进行打分，找出其中具有较大影响力的用户作为商业洽谈人选。
2. 针对上述你所设计的算法，使用华为云 MRS 大数据平台实现它们。

## 1.2 问题背景

作为 Web2.0 技术的典型应用，社交网络服务自从诞生之初就得到了迅猛发展，并且渗透到人类日常生活的各个方面，出现了一大批有代表性的社交平台，如国内的微信、腾讯 QQ、新浪微博等，国外的 Facebook、Twitter、Google+ 等。基于上述社交应用，人们不仅可以进行互动、交友、协作等传统的社交行为、还可以在相应功能的平台进行学习、购物、游戏、民意调查、时事获取等网络行为。社交网络时代的到来，深刻变革了网络的使用模式：不再是单一的信息检索与网页浏览，而是基于社交关系构建与维护的信息贡献与分享。在信息时代，人们产生或消费信息的需求逐渐增强，而社交网络成为人们信息获取与互动的重要媒介。

社交网络将广大用户关联到一个大规模的虚拟网络之中，极大地降低了用户之间的交流成本，促进了信息在网络中的快速传播。信息的播离不开用户之间的交互，这些交互行为及其产生的数据为分析社交网络用户行为特点及信息传播规律等问题提供了支撑。社交网络分析就是综合利用多种学科的理论和方法对上述问题进行量化分析，与传统的社会学研究相比，它不仅关注网络中的用户属性，还注重分析用户之间的社交关系和交互行为以及信息在网络中的传播机制。

影响力分析作为社交网络分析的重要内容，是揭示用户行为形成及互动规律的关键技术，为理解网络信息传播和关系结构演化奠定基础，在舆情监控、网络营销、情报分析、个性化推荐、传播预测等多个领域具有重要的社会价值和现实意义，早已成为学术界和工业界的研究热点。

### 1.3 问题挑战

在社会学中，虽然对于影响力的概念并没有提出非常明确的定义，但是大家普遍比较认可的一个说法是：改变他人想法或行为的能力。

参考社会学的定义，并且联系本课题研究的目的之后，我们发现，其实我们需要研究的只是人物的影响力而已。我们可以得出如下相关定义：

人物在某个群体中的影响力：在某个群体中，某人采取某个行为之后，改变他人想法或行为的能力，叫做其在这个群体中的影响力。

社会学中的这种定义方式，大体上来说，还是比较空泛的。问题在于，怎么才算是改变了他人的想法和能力？什么样的东西才算是影响力？

两者影响力如果是个数值，那数值之间的比较意味着什么？如果两个人影响力数值相同，那这种影响力是不是随着不同领域的变化而变化？每个人的影响力是不是也随着时间的变化而变化？这些都是影响力研究本身的难点。

## 2 方法框架

我们对研究问题进一步抽象，我们要解决的就是如何评价一个节点在其社交网络中的重要性，也即是影响力。

### 2.1 当前研究现状

基于网络结构的方法主要通过分析社交数据构建的拓扑结构来衡量用户在网络中的重要性或影响力，其中最经典的是以图论为基础的中心性测量方法，包括度中心性 (Degree centrality)、亲近中心性 (Closeness centrality) 和介数中心性 (Betweenness centrality) 等。度中心性是一种最简的中心性测量，被定义为网络中与节点相连接边数，或与节点相邻节点数。显然，拥有更多连接关系的用户将处于一个重要位置，更容易拥有较高的影响力。在有向图结构中，度中心性还可分为入度中心性和出度中心性两类。亲近中心性是指节点与其它所有节点距离之和的倒数，反映了节点将信息传播到网络中其它节点的速率。介数中心性是指节点处于其它节点之间测地线的频率，表征了对非相邻节点之间信息交互的控制能力。此外，特征向量中心性也可用于测量网络中节点的重要性，并与相邻节点的中心性有关。为了测量不对称网络中节点的中心性，Bonacich 等人提出了  $\alpha$  中心性测量，认为节点中心性不仅与它们之间的连接有关，还依赖于外部因素，并通过  $\alpha$  参数调节内外因素的相对重要性。Kitsak 等人提出了 K-shell 分解法来挖掘高影响力节点，通过在多个数据集上的实验结果表明，相比于高中心性或者高介数节点，K-shell 值大的节点更具有影响力。H 指数一般用于评价研究人员在学术

领域中的生产力和影响力，Romero 等人利用 Twitter 中的 URL 来计算 H 指数，进而识别出高影响力用户。Kwak 等人在 Twitter 中分别利用粉丝数、PageRank 算法以及博文转发数来对用户影响力进行排名，发现前两种方法的结果相似，而与第三种方法的结果存在显著差异，说明用户影响力不同于博文影响力。利用 Flickr 中用户的属性，Almgren 等人设计了一种基于用户结构位置的混合框架来预测影响力用户，并在 Digg 数据集上验证了该方法的有效性。Wang 等人提出了一种在社区中挖掘 top-k 影响力用户的 CGA 算法，其思想是将网络划分为不同社区，然后在每个社区中定位影响力用户。

大多数中心性测量方法没有考虑节点的类型，但在实际应用中，与高影响力节点相连接具有更大价值，如 PageRank 算法 [50] 和 HITS 算法。PageRank 算法给网络中的节点指定一个 PageRank 值，而 HITS 算法则给节点指定一个权威值和一个中心值。后来，研究人员基于这类思想提出了很多有效的个体影响力度量方法。例如，为了克服 PageRank 算法在人际关系网络中的低效问题，Lü 等人提出了一种自适应的无参数的 LeaderRank 算法，该算法在挖掘影响力用户时无需考虑网络类型。Hajian 等人扩展了 PageRank 算法，基于关注、提及、转发、收藏等行为来识别意见领袖，并认为成为高影响力用户需要拥有高影响力粉丝。Jabeur 等人提出的 InfRank 算法和 LeadRank 算法，前者利用传播信息和被高影响力用户转发的能力来度量用户影响力，后者利用被其他用户转发和提及的能力来度量用户领导能力。通过考虑用户社交多样性和信息传播概率，Huang 等人在转发和关注关系图上利用扩展的 PageRank 算法来挖掘用户影响力。Pujol 等人提出了只依赖局部结构信息的 NodeRanking 方法来识别影响力用户，并能适应不同类型的网络或图结构。Majer 等人提出的 UserRank 方法，通过用户的关注关系和发布的博文来计算个体影响力。

## 2.2 本文方法

经过对当前研究的查阅，可以发现现在对用户影响力的研究大多是对 PageRank 算法及 HIT 算法等经典算法的发展与改进，在其中不仅考虑网络的拓扑结构，还更多的考虑到了一些时序信息（用户活跃度等信息）。但是在北大作业的数据集中仅仅给出了网络的拓扑结构。

数据集格式如下（图 1）：

所以在本文中会使用比较初始的算法，如 PageRank 算法，HIT 算法。

## 3 算法 0

因为原论文（大作业给出题目来源）给出的数据集太大（源数据未解压达到 6GB 以上，且解压后一个分包（共 4 个分包）达到 7GB），本人电脑性能不足以支持如此大

## Social graph

- Download

\* Now we offer direct download links: [\[GitHub\]](#)

twitter\_rv.tar.gz.torrent (34KB) or twitter\_rv.zip.torrent (26KB) (# of seeds >= 4)

twitter\_rv.tar.gz, 6,475,352,982 bytes, MD5: c31b4c2d6f3ae325e516e78b499c46f8

twitter\_rv.zip, 4,859,337,443 bytes, MD5: 5f2399aac71c604ac5a100fb6ca7e297

----

twitter\_rv.net, 26,172,280,241 bytes, MD5: 9c0f7983a523edd1b753af68c5acc4bd

- Format

```
USER \t FOLLOWER \n
```

\* USER and FOLLOWER are represented by numeric ID (integer).

\* These numeric IDs are the same as numeric IDs Twitter managed.

\* Therefore, you can access a profile of user 12 via [http://api.twitter.com/1/users/show.xml?user\\_id=12](http://api.twitter.com/1/users/show.xml?user_id=12).

user\_id=12.

\* For details, see [Twitter API Page](#)

- Example

```
12 13
12 14
12 15
16 17
```

\* Users 13, 14 and 15 are followers of user 12.

\* User 17 is a follower of user 16.

图 1: 数据集格式

规模的计算（甚至连将其中一个分包在 sublime 中打开都做不到），这里使用一个简单的算法来生成测试数据。

### 3.1 程序代码

首先引入一个类来表示生成的一行数据：

```
1 public class TwoId {
2     private int userId;
3     private int followerId;
4     private static final int MAX = 10000;
5     private static final int MIN = 100;
6     public TwoId() {
7         Random random = new Random();
8         this.userId = random.nextInt(MAX - MIN + 1) +
9             MIN;
10        this.followerId = random.nextInt(MAX - MIN + 1)
11            + MIN;
12    }
13 }
```

```
12     @Override
13     public String toString() {
14         return userId+"\t"+followerId+"\n";
15     }
16
17     @Override
18     public boolean equals(Object o) {
19         if (this == o) return true;
20         if (o == null || getClass() != o.getClass())
21             return false;
22         TwoId twoId = (TwoId) o;
23         return new EqualsBuilder().append(userId, twoId.
24             userId).append(followerId, twoId.followerId).
25             isEqual();
26     }
27
28     @Override
29     public int hashCode() {
30         return new HashCodeBuilder(17, 37).append(userId
31             ).append(followerId).toHashCode();
32     }
33 }
```

然后循环生成每一行在输出到文件中。(这里暂定生成 1000 行)

```
1 public class Generate {
2     private static final int COUNT = 1000;
3     public static void main(String[] args) {
4         Set<TwoId> set = new HashSet<>();
5         for (int i = 0; i < COUNT; i++) {
6             set.add(new TwoId());
7         }
8         FileWriter fileWriter = null;
9         try {
10             File file = new File("input/graph1.txt");
11             if(file.exists()) {
```



```
12         file.delete();
13         file.createNewFile();
14     }
15     fileWriter = new FileWriter(file);
16     for (TwoId ids :
17         set) {
18         fileWriter.write(ids.toString());
19     }
20 } catch (IOException e) {
21     e.printStackTrace();
22 } finally {
23     if (fileWriter != null) {
24         try {
25             fileWriter.flush();
26             fileWriter.close();
27         } catch (IOException e) {
28             e.printStackTrace();
29         }
30     }
31 }
32
33 }
34 }
```

## 3.2 程序结果

可以看到生成数据如原论文数据集一样（如图 2）

## 4 算法 1

在评价用户影响力的算法中，最直接的便是统计结点本身的某些特性来推断其影响力。

从社交网络中用户影响力分析的需求来看，我们最需要选择的是那些可以引起其他人反应的用户，比如转发和评论。这就决定了那些越是可以把自己的消息送到最多的人面前的用户，影响力越大。

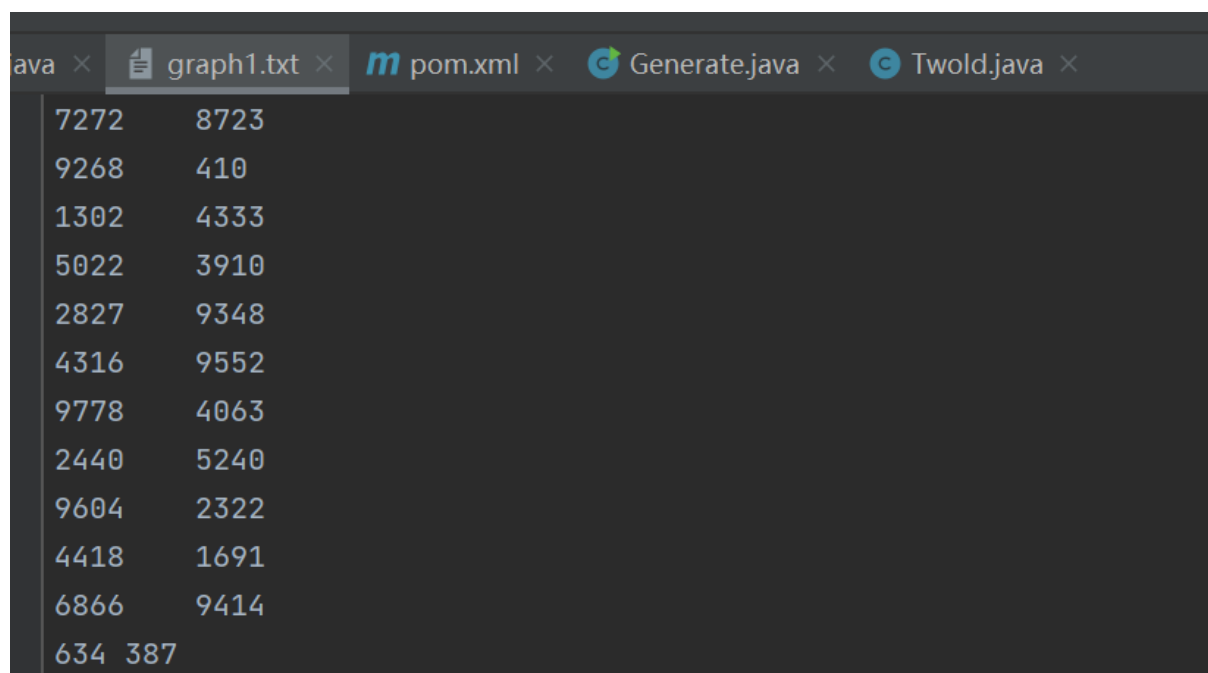


图 2: 生成的测试数据

在这里，我们可以使用社会学者们经常使用的影响力评价算法：最大度算法。

最大度算法的一个最基本的假设是，越是拥有更多跟其他节点链接的节点，越可能让更多人看到自己所发布的信息。因此，一个朴素的想法是，社交网络中度数越大的结点，其影响力也越大。即：

$$Inf(v) = d(v), v \in V \quad (1)$$

## 4.1 程序代码

这里使用 mapreduce 来实现此算法，接下来给出程序代码

```

1 public class AnalysisMapper extends Mapper<LongWritable,
    Text, LongWritable, LongWritable> {
2     private static final Logger LOGGER = Logger.
        getLogger(AnalysisMapper.class);
3     private static LongWritable one = new LongWritable
        (1);
4     private static LongWritable id = new LongWritable();
5
6     @Override

```

```
7     protected void map(LongWritable key, Text value,
                        Mapper<LongWritable, Text, LongWritable,
                        LongWritable>.Context context) throws IOException
                        , InterruptedException {
8         String line = value.toString();
9         //LOGGER.info("读入字符串"+line);
10        String[] words = line.split("\\t");
11        long userId = Long.parseLong(words[0]);
12        long followerId = Long.parseLong(words[1]);
13        id.set(userId);
14        context.write(id, one);
15    }
16 }
```

```
1 public class AnalysisReducer extends Reducer<
    LongWritable, LongWritable, LongWritable,
    LongWritable> {
2     private static LongWritable sumNum = new
        LongWritable();
3
4     @Override
5     protected void reduce(LongWritable key, Iterable<
        LongWritable> values, Reducer<LongWritable,
        LongWritable, LongWritable, LongWritable>.Context
        context) throws IOException,
        InterruptedException {
6         long sum = 0;
7         for (LongWritable value :
8             values) {
9             sum += value.get();
10        }
11        sumNum.set(sum);
12        context.write(key, sumNum);
13    }
14 }
```

接下来展示该算法 Driver 程序

```
1 public class MyDriver {
2     public static class LongWritableDecreasingComparator
        extends
3         LongWritable.Comparator {
4         public int compare(byte[] b1, int s1, int l1,
            byte[] b2, int s2, int l2) {
5             return -super.compare(b1, s1, l1, b2, s2, l2
                );
6         }
7     }
8
9
10    public static void main(String[] args) throws
        IOException, InterruptedException,
        ClassNotFoundException {
11        Configuration conf = new Configuration();
12        Job job1 = Job.getInstance(conf, "Analysis");
13        job1.setJarByClass(MyDriver.class);
14        job1.setMapperClass(AnalysisMapper.class);
15        job1.setReducerClass(AnalysisReducer.class);
16        job1.setMapOutputKeyClass(LongWritable.class);
17        job1.setMapOutputValueClass(LongWritable.class);
18        FileInputFormat.addInputPath(job1, new Path(args
            [0]));
19
20        FileOutputFormat.setOutputPath(job1, new Path(
            args[1]));
21        FileSystem fs = new Path(args[1]).getFileSystem(
            conf);
22        if (fs.exists(new Path(args[1]))) { fs.delete(new
            Path(args[1]), true); }
23
24        if (job1.waitForCompletion(true)) {
25            Job job2 = Job.getInstance(conf, "sort");
```

```

26         job2.setJarByClass(MyDriver.class);
27         job2.setMapperClass(SortMapper.class);
28         job2.setReducerClass(SortReducer.class);
29         job2.setMapOutputKeyClass(LongWritable.class);
30         job2.setMapOutputValueClass(Text.class);
31
32         FileInputFormat.addInputPath(job2, new Path(
33             args[1]));
34         FileOutputFormat.setOutputPath(job2, new Path(
35             args[2]));
36         if (fs.exists(new Path(args[2]))){
37             fs.delete(new Path(args[2]), true);
38         }
39
40         job2.setSortComparatorClass(
41             LongWritableDecreasingComparator.class);
42         System.exit(job2.waitForCompletion(true)
43             ? 0 : 1);
44     }
45 }
46 }

```

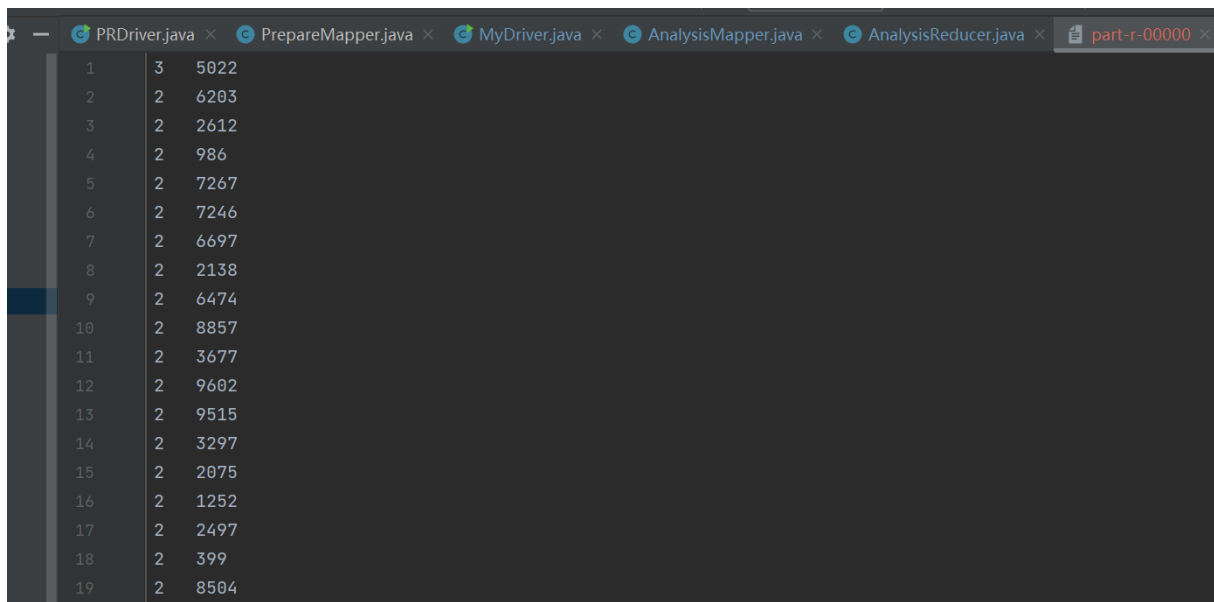
## 4.2 实验结果

实验结果（如图 3）每行两列，第一列为该节点的入度，第二列为节点 ID。另外这里还使用 MapReduce 本身特性进行了降序排序，因其不是重点将在附录中给出，且后续算法中不再进行这样的简单排序。

## 5 算法 2

算法二采用经典的 PageRank 算法。

历史上，PageRank 算法作为计算互联网网页重要度的算法被提出。PageRank 是定义在网页集合上的一个函数，它对每个网页给出一个正实数，表示网页的重要程度，



1	3	5022
2	2	6203
3	2	2612
4	2	986
5	2	7267
6	2	7246
7	2	6697
8	2	2138
9	2	6474
10	2	8857
11	2	3677
12	2	9602
13	2	9515
14	2	3297
15	2	2075
16	2	1252
17	2	2497
18	2	399
19	2	8504

图 3: 最大度算法结果

整体构成一个向量，PageRank 值越高，网页就越重要，在互联网搜索的排序中可能就被排在前面。

假设互联网是一个有向图，在其基础上定义随机游走模型，即一阶马尔可夫链，表示网页浏览者在互联网上随机浏览网页的过程。假设浏览者在每个网页依照连接出去的超链接以等概率跳转到下一个网页，并在网上持续不断进行这样的随机跳转，这个过程形成一阶马尔可夫链。PageRank 表示这个马尔可夫链的平稳分布。每个网页的 PageRank 值就是平稳概率。用户的关注关系组成的社交网络同样是这样以个有向图，所以我们可以套用 PageRank 算法来计算用户影响力。

## 5.1 提前处理

### 5.1.1 代码

因为 PageRank 的迭代算法需要数据处于一定形式。首先需要对原数据进行一些处理将其整理成方便 PageRank 算法处理的形式。

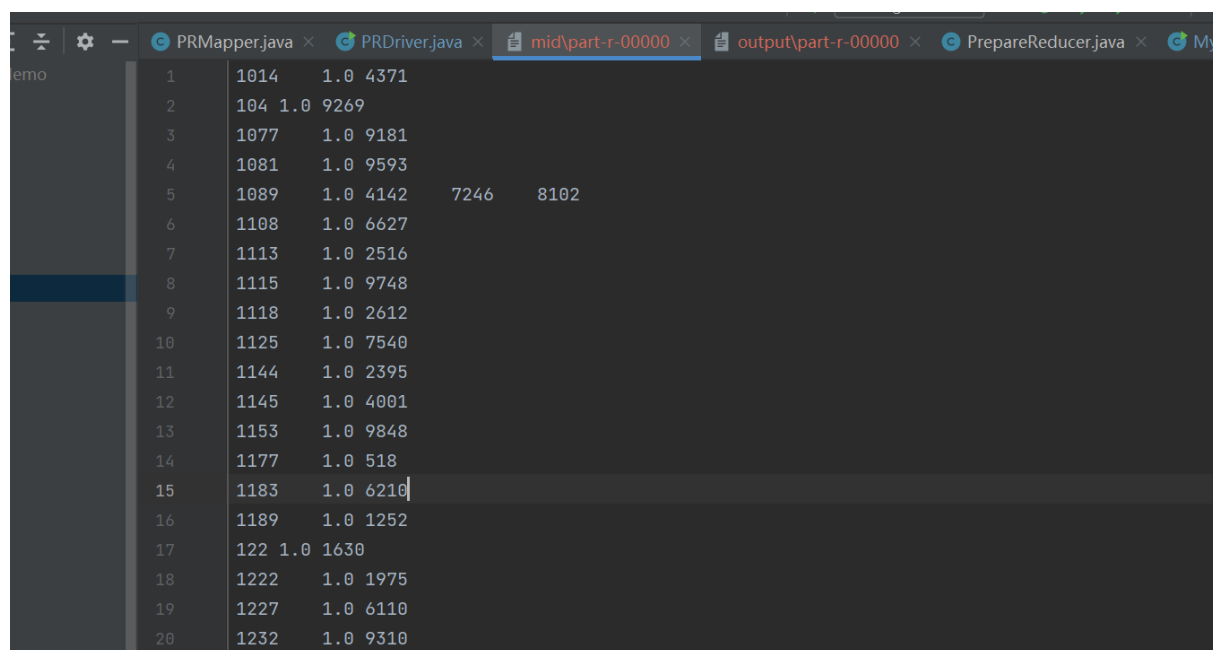
```
1 public class PrepareMapper extends Mapper<Object, Text,  
    Text, Text> {  
2     private static Text user = new Text();  
3     private static Text follower = new Text();  
4  
5     @Override
```

```
6     protected void map(Object key, Text value, Mapper<
      Object, Text, Text, Text>.Context context) throws
      IOException, InterruptedException {
7         StringTokenizer stringTokenizer = new
          StringTokenizer(value.toString());
8         String userId = stringTokenizer.nextToken();
9         user.set(userId);
10        String followerId = stringTokenizer.nextToken();
11        follower.set(followerId);
12        context.write(follower, user);
13    }
14 }
```

```
1 public class PrepareReducer extends Reducer<Text, Text,
      Text, Text> {
2     private static Text v = new Text();
3
4     @Override
5     protected void reduce(Text key, Iterable<Text>
      values, Reducer<Text, Text, Text, Text>.Context
      context) throws IOException, InterruptedException
      {
6         StringBuilder sb = new StringBuilder("1.0");
7         for (Text value :
8             values) {
9             sb.append("\t").append(value.toString());
10        }
11        v.set(sb.toString());
12        context.write(key, v);
13    }
14 }
```

### 5.1.2 结果

数据准备结果如下（如图 4），第一列为用户 ID，第二列（1.0）为初始化 PR 值，第三列及其后面是关注者 ID。



1	1014	1.0	4371		
2	104	1.0	9269		
3	1077	1.0	9181		
4	1081	1.0	9593		
5	1089	1.0	4142	7246	8102
6	1108	1.0	6627		
7	1113	1.0	2516		
8	1115	1.0	9748		
9	1118	1.0	2612		
10	1125	1.0	7540		
11	1144	1.0	2395		
12	1145	1.0	4001		
13	1153	1.0	9848		
14	1177	1.0	518		
15	1183	1.0	6210		
16	1189	1.0	1252		
17	122	1.0	1630		
18	1222	1.0	1975		
19	1227	1.0	6110		
20	1232	1.0	9310		

图 4: 数据准备结果

## 5.2 PageRank 算法

### 5.2.1 代码

```

1 public class PRMapper extends Mapper<Object, Text, Text,
    Text> {
2     private static Text KEY = new Text();
3     private static Text VALUE = new Text();
4
5     @Override
6     protected void map(Object key, Text value, Mapper<
        Object, Text, Text, Text>.Context context) throws
        IOException, InterruptedException {
7         StringTokenizer tokenizer = new StringTokenizer(
            value.toString());
8         String id = tokenizer.nextToken();

```



```

9         double pageRankValue = Double.parseDouble(
            tokenizer.nextToken());
10        int count = tokenizer.countTokens();
11        double averagePR = pageRankValue / count;
12
13        StringBuilder linkIds = new StringBuilder("&");
14        while (tokenizer.hasMoreTokens()) {
15            String linkId = tokenizer.nextToken();
16            KEY.set(linkId);
17            VALUE.set("@ " + averagePR);
18            context.write(KEY, VALUE);
19            linkIds.append(" ").append(linkId);
20        }
21        KEY.set(id);
22        VALUE.set(linkIds.toString());
23        context.write(KEY, VALUE);
24    }
25 }

```

```

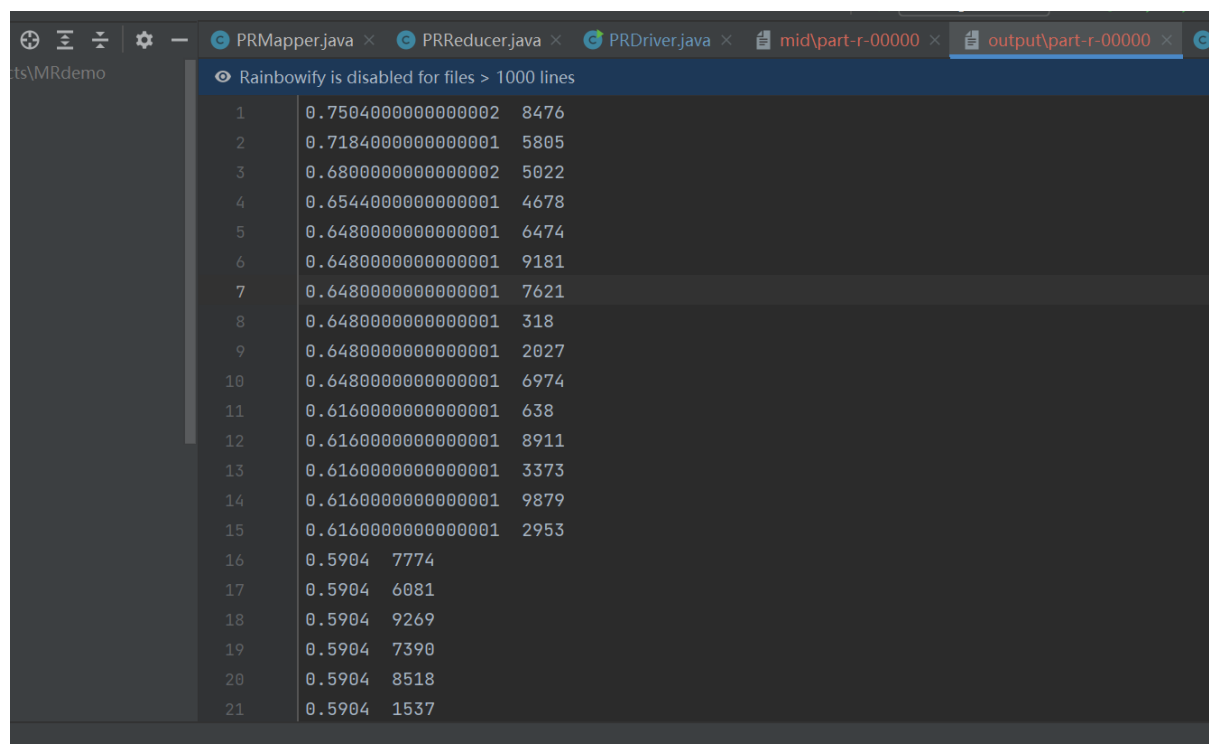
1 public class PRReducer extends Reducer<Text, Text, Text,
    Text> {
2     private static Text VALUE = new Text();
3
4     @Override
5     protected void reduce(Text key, Iterable<Text>
        values, Reducer<Text, Text, Text, Text>.Context
        context) throws IOException, InterruptedException
        {
6         StringBuilder link = new StringBuilder();
7         double pr = 0.0;
8         for (Text val : values) {
9             if (val.toString().charAt(0) == '@') {
10                 pr += Double.parseDouble(val.toString().
                    substring(1));
11             } else if (val.toString().charAt(0) == '&')

```

```
12         {
13             link.append(val.toString().substring(1))
14             ;
15         }
16     }
17     pr = 0.8 * pr + 0.2;
18     VALUE.set(pr + link.toString());
19     context.write(key, VALUE);
20 }
```

### 5.2.2 结果

这里直接展示迭代并排序后的结果



Item ID	PageRank	Count
1	0.7504000000000002	8476
2	0.7184000000000001	5805
3	0.6800000000000002	5022
4	0.6544000000000001	4678
5	0.6480000000000001	6474
6	0.6480000000000001	9181
7	0.6480000000000001	7621
8	0.6480000000000001	318
9	0.6480000000000001	2027
10	0.6480000000000001	6974
11	0.6160000000000001	638
12	0.6160000000000001	8911
13	0.6160000000000001	3373
14	0.6160000000000001	9879
15	0.6160000000000001	2953
16	0.5904	7774
17	0.5904	6081
18	0.5904	9269
19	0.5904	7390
20	0.5904	8518
21	0.5904	1537

图 5: PageRank 结果

## 6 算法 3

算法 3 采用 HITS 算法。

HITS 算法的全称是“基于超链接的主题搜索”(Hyperlink-Induced Topic Search), 该算法由 Jon Kleinberg 于 1999 年提出, 与 PageRank 算法一样, 也是一种用于对网页进行排序的算法。与 PageRank 不同的是, HITS 将网页分成两类, 即: Hub 页面和 Authority 页面。其中 Hub 页面类似于常见的门户网站, 像 hao123 首页之类的, 它提供了大量高质量的网页链接; 而 Authority 页面更像是用户希望访问的网站, 比如搜索的时候我们希望用百度, 购物的时候我们希望进入淘宝和京东等。Hub 页面相当于充当了一个中间枢纽的角色, 对于用户而言, 他们更关注高 Authority 的网页。下面简单介绍一下 HITS 算法的原理和求解过程。

经过笔者的计算, 发现如果继续使用 MapReduce 的方式完成这个算法, 每一次迭代至少需要四到五次 MapReduce 过程, 且其中的数据变化过于繁杂, 所以这里改用 Spark 框架来完成算法。

### 6.1 程序代码

```
1 public class HITS {
2     private static final Logger LOGGER = Logger.
        getLogger(HITS.class);
3     private static final double MIN_DELTA = 0.0001;
4
5     private static final int MAX_ITERATION = 50;
6
7     public static void main(String[] args) {
8         SparkConf sc = new SparkConf().setMaster("local
        [*]").setAppName("PageRank");
9         JavaSparkContext jsc = new JavaSparkContext(sc);
10        jsc.setLogLevel("WARN");
11
12        if (args.length < 2) {
13            LOGGER.error("should assigned parameter of
        page info");
14            System.exit(-1);
15        }
```

```

16     }
17     JavaRDD<String> data = jsc.textFile(args[0]);
18
19     JavaRDD<String> inIds = data.map((Function<
20         String, String>) s -> s.split("\t")[0]);
21     JavaRDD<String> outIds = data.map((Function<
22         String, String>) s -> s.split("\t")[1]);
23     JavaPairRDD<String, Double> hub = inIds.union(
24         outIds)
25         .mapToPair((PairFunction<String, String,
26             Double>) s -> new Tuple2<>(s, 1.0))
27         .distinct();
28     JavaPairRDD<String, Double> authority = hub;
29
30     JavaPairRDD<String, String> inLinks = data.
31         mapToPair((PairFunction<String, String,
32             String>) s -> {
33             String[] strings = s.split("\t");
34             return new Tuple2<>(strings[0], strings[1]);
35         }).cache();
36     JavaPairRDD<String, String> outLinks = data.
37         mapToPair((PairFunction<String, String,
38             String>) s -> {
39             String[] strings = s.split("\t");
40             return new Tuple2<>(strings[1], strings[0]);
41         }).cache();
42     for (int i = 0; i < MAX_ITERATION; i++) {
43         LOGGER.warn("—————第次" + (i + 1) +
44             "迭代—————");
45         // authority 为所有指向它的 hub 值
46         JavaPairRDD<String, Double> tempAuthority =
47             outLinks.join(hub)
48             .mapToPair((PairFunction<Tuple2<
49                 String, Tuple2<String, Double>>,
50                 String, Double>)

```

```

39         stringTuple2Tuple2
            -> new Tuple2<>(  

                stringTuple2Tuple2._2._1,  

                stringTuple2Tuple2._2._2  

            ))  

40         .reduceByKey((Function2<Double,  

            Double, Double>) Double::sum);  

41 tempAuthority = authority.leftOuterJoin(  

    tempAuthority)  

42     .mapToPair((PairFunction<Tuple2<  

        String, Tuple2<Double, Optional<  

        Double>>>, String, Double>)  

        stringTuple2Tuple2  

43         -> new Tuple2<>(  

            stringTuple2Tuple2._1,  

            stringTuple2Tuple2._2._2.  

            or(0.0))));  

44  

45 // 求和 标准化  

46 Double auSum = Math.sqrt(tempAuthority.  

    values()  

47     .map((Function<Double, Double>  

        aDouble -> Math.pow(aDouble, 2.0)  

        )  

48     .reduce((Function2<Double, Double,  

        Double>) Double::sum));  

49 JavaPairRDD<String, Double> newAuthority =  

    tempAuthority  

50     .mapToPair((PairFunction<Tuple2<  

        String, Double>, String, Double>)  

        stringDoubleTuple2  

51         -> new Tuple2<>(  

            stringDoubleTuple2._1,  

            stringDoubleTuple2._2 /  

            auSum));

```

```

52 //hub为所有它指向的 authority 值
53 JavaPairRDD<String, Double> tempHub =
    inLinks.join(authority)
54     .mapToPair((PairFunction<Tuple2<
        String, Tuple2<String, Double>>,
        String, Double>)
        stringTuple2Tuple2
55         -> new Tuple2<>((
            stringTuple2Tuple2._2._1,
            stringTuple2Tuple2._2._2
        )))
56     .reduceByKey((Function2<Double,
        Double, Double>) Double::sum);
57 tempHub = hub.leftOuterJoin(tempHub)
58     .mapToPair((PairFunction<Tuple2<
        String, Tuple2<Double, Optional<
        Double>>>, String, Double>)
        stringTuple2Tuple2
59         -> new Tuple2<>((
            stringTuple2Tuple2._1,
            stringTuple2Tuple2._2._2.
            or(0.0))) );
60 Double hubSum = Math.sqrt(tempHub.values()
61     .map((Function<Double, Double>)
        aDouble -> Math.pow(aDouble, 2.0)
        )
62     .reduce((Function2<Double, Double,
        Double>) Double::sum));
63 JavaPairRDD<String, Double> newHub = tempHub
64     .mapToPair((PairFunction<Tuple2<
        String, Double>, String, Double>)
        stringDoubleTuple2
65         -> new Tuple2<>((
            stringDoubleTuple2._1,
            stringDoubleTuple2._2 /

```

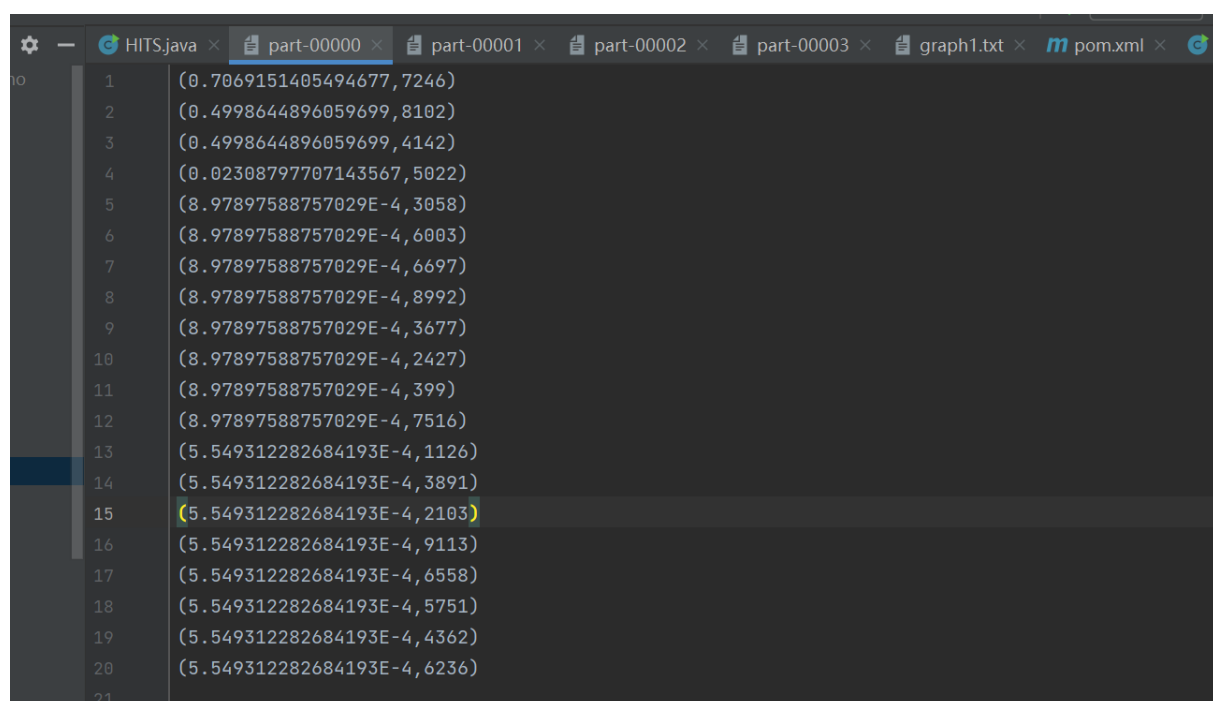
```

        hubSum));
66 // 计算改变量
67 Double change1 = newAuthority.join(authority
    )
68     .map((Function<Tuple2<String, Tuple2
        <Double, Double>>, Double>)
        stringTuple2Tuple2
69         -> Math.abs(
            stringTuple2Tuple2._2._1
            - stringTuple2Tuple2._2.
                _2))
70     .reduce((Function2<Double, Double,
        Double>) Double::sum);
71 Double change2 = newHub.join(hub)
72     .map((Function<Tuple2<String, Tuple2
        <Double, Double>>, Double>)
        stringTuple2Tuple2
73         -> Math.abs(
            stringTuple2Tuple2._2._1
            - stringTuple2Tuple2._2.
                _2))
74     .reduce((Function2<Double, Double,
        Double>) Double::sum);
75 // 更新 authority
76 authority = newAuthority;
77 hub = newHub;
78 if (change2 + change1 < MIN_DELTA) break;
79 }
80 authority.mapToPair(new PairFunction<Tuple2<
    String, Double>, Double, String>() {
81     @Override
82     public Tuple2<Double, String> call(Tuple2<
        String, Double> stringDoubleTuple2)
        throws Exception {
83         return new Tuple2<>(stringDoubleTuple2.

```

```
                _2, stringDoubleTuple2._1);  
84            }  
85        }).sortByKey(false).saveAsTextFile(args[1]);  
86    }  
87 }
```

## 6.2 实验结果



1	(0.7069151405494677,7246)
2	(0.4998644896059699,8102)
3	(0.4998644896059699,4142)
4	(0.02308797707143567,5022)
5	(8.97897588757029E-4,3058)
6	(8.97897588757029E-4,6003)
7	(8.97897588757029E-4,6697)
8	(8.97897588757029E-4,8992)
9	(8.97897588757029E-4,3677)
10	(8.97897588757029E-4,2427)
11	(8.97897588757029E-4,399)
12	(8.97897588757029E-4,7516)
13	(5.549312282684193E-4,1126)
14	(5.549312282684193E-4,3891)
15	(5.549312282684193E-4,2103)
16	(5.549312282684193E-4,9113)
17	(5.549312282684193E-4,6558)
18	(5.549312282684193E-4,5751)
19	(5.549312282684193E-4,4362)
20	(5.549312282684193E-4,6236)
21	

图 6: HITS 算法结果

## 7 结果分析

这里列出各个算法分析结果的前 15 个 userId。这里可以看到标出三个结果中相同的部分并不多（如图 7），个人推测是随机生成的数据并不具有真正社交网络的一些性质，所以导致算法并不好用。



Max Degree		PageRank		HITS	
userId	degree	userId	pr	userId	authority
5022	3	8476	0.7504	7246	0.706915141
6203	2	5805	0.7184	8102	0.49986449
2612	2	5022	0.68	4142	0.49986449
986	2	4678	0.6544	5022	0.023087977
7267	2	6474	0.648	3058	8.98E-04
7246	2	9181	0.648	6003	8.98E-04
6697	2	7621	0.648	6697	8.98E-04
2138	2	318	0.648	8992	8.98E-04
6474	2	2027	0.648	3677	8.98E-04
8857	2	6974	0.648	2427	8.98E-04
3677	2	638	0.616	399	8.98E-04
9602	2	8911	0.616	7516	8.98E-04
9515	2	3373	0.616	1126	5.55E-04
3297	2	9879	0.616	3891	5.55E-04
2075	2	2953	0.616	2103	5.55E-04

## 8 MRS 平台实现

这里是华为 MRS 平台上的运行成功示意图（如图 8），及运行成功后的 OBS 桶（如图 9）。另外由于华为平台给出的命令貌似有 BUG（如图 10），（也可能是我本人使用的框架版本和其默认版本不同）导致 Spark 实现的 HITS 算法无法在上面运行。

## 9 结论

本文使用 MapRduce 框架实现了最大度算法，PageRank 算法，利用 Spark 框架实现了 HITS 算法，并在 MRS 平台上运行这三个算法，利用这些算法对部分 Twitter 数据进行用户影响力评分。

Max Degree		PageRank		HITS	
userId	degree	userId	pr	userId	authority
5022	3	8476	0.7504	7246	0.706915141
6203	2	5805	0.7184	8102	0.49986449
2612	2	5022	0.68	4142	0.49986449
986	2	4678	0.6544	5022	0.023087977
7267	2	6474	0.648	3058	8.98E-04
7246	2	9181	0.648	6003	8.98E-04
6697	2	7621	0.648	6697	8.98E-04
2138	2	318	0.648	8992	8.98E-04
6474	2	2027	0.648	3677	8.98E-04
8857	2	6974	0.648	2427	8.98E-04
3677	2	638	0.616	399	8.98E-04
9602	2	8911	0.616	7516	8.98E-04
9515	2	3373	0.616	1126	5.55E-04
3297	2	9879	0.616	3891	5.55E-04
2075	2	2953	0.616	2103	5.55E-04

图 7: 结果分析

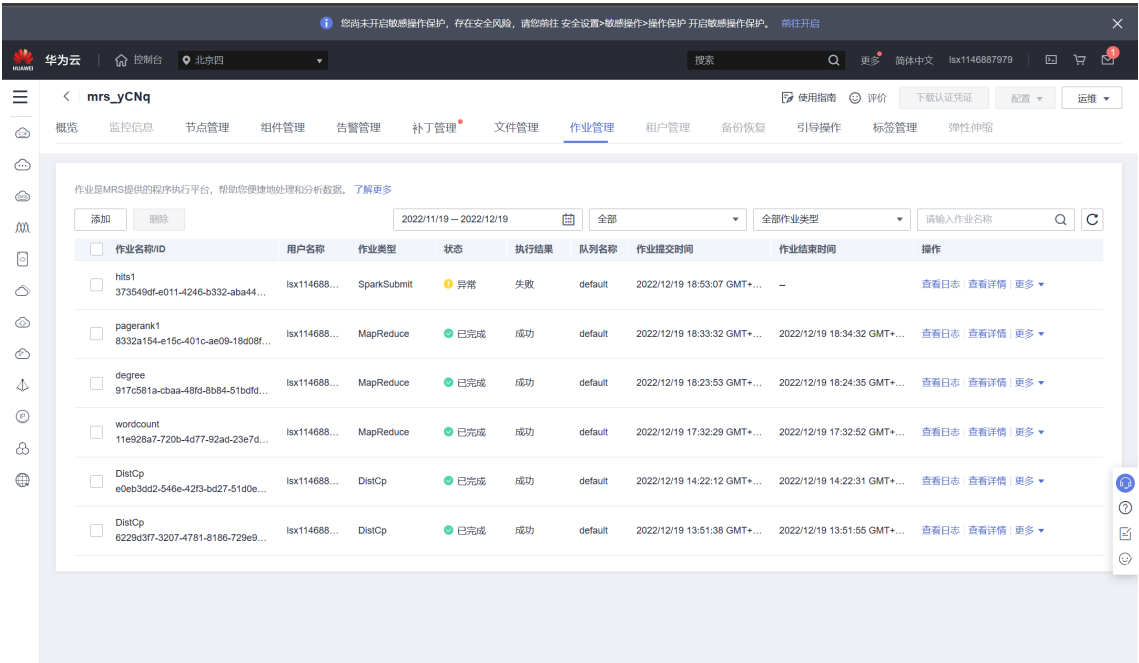


图 8: MRS 运行结果

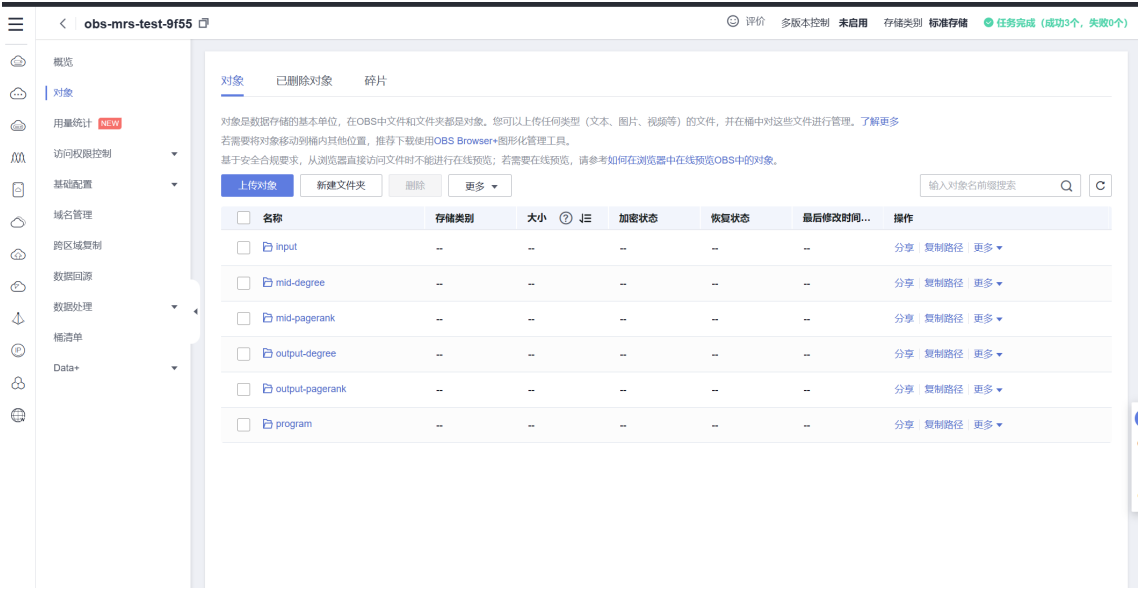


图 9: OBS 桶运行结果

作业类型

SparkSubmit

作业名称

hits1

container-localizer-syslog

stderr

stdout

syslog

```
1 SLF4J: Class path contains multiple SLF4J bindings.
2 SLF4J: Found binding in [jar:file:/srv/BigData/hadoop/data1/nm/localdir/usercache/lsx1146887979/appcache/application_1671427974095_0064/filecache/251/slf4j-1.7.16.jar!/org/slf4j/impl/StaticLoggerBinder.class]
3 SLF4J: Found binding in [jar:file:/opt/share/slf4j-log4j12-1.7.10/slf4j-log4j12-1.7.10.jar!/org/slf4j/impl/StaticLoggerBinder.class]
4 SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
5 SLF4J: Actual binding is of type [org.slf4j.impl.Log4jLoggerFactory]
6 Warning: Master yarn-cluster is deprecated since 2.0. Please use master "yarn" with specified deploy mode instead.
7 Warning: Skip remote jar obs://obs-mrs-test-9f55/program/sparkDemo.jar.
8 java.lang.reflect.InvocationTargetException
9   at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
10  at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
```

### Master yarn-cluster is deprecated since 2.0. Please use master "yarn" with specified deploy mode

技术标签: spark spark-submit yarn

使用下面命令,

```
1 bin/spark-submit --class SimpleApp --master yarn-cluster /home/zzhan/workspace/jobs/sparkdemo_2.11-0.1.jar
```

会产生下面的告警信息:

```
1 Master yarn-cluster is deprecated since 2.0. Please use master "yarn" with specified deploy mode instead.
```

### 使用新命令替换之

```
1 bin/spark-submit --class SimpleApp --master yarn --deploy-mode cluster /home/zzhan/workspace/jobs/sparkdemo_2.11-
```

图 10: 命令 BUG

## 10 参考文献

### 参考文献

- [1] 全拥, 贾焰, 张良, 朱争, 周斌, 方滨兴. 在线社交网络个体影响力算法测试与性能评估 [J]. 通信学报, 2018, 39(10): 1-10.
- [2] 全拥. 面向社交网络的影响力分析关键技术研究 [D]. 国防科技大学, 2019. DOI: 10.27052/d.cnki.gzjgu.2019.000260.
- [3] 王彪. 社交网络中的用户影响力分析 [D]. 哈尔滨工业大学, 2012.
- [4] 吴思竹, 张智雄. 网络中心度计算方法研究综述 [J]. 图书情报工作, 2010, 54(18): 107-110+148.
- [5] MapReduce 之 PageRank 算法概述、设计思路和源码分析 (<https://blog.csdn.net/u010414589/article/details/51404971>) [EB/OL]
- [6] HITS 算法—从原理到实现 (<https://blog.csdn.net/rubinorth/article/details/52231620>) [EB/OL]

## 11 附录

### 11.1 排序程序

这里是利用 MapReduce 框架本身会根据 key 来进行排序的特性来编写的排序程序。

```
1 public class SortMapper extends Mapper<LongWritable ,
    Text, LongWritable , Text> {
2     private static Text id = new Text();
3     private static LongWritable num = new LongWritable()
        ;
4
5     @Override
6     protected void map(LongWritable key, Text value ,
        Mapper<LongWritable , Text, LongWritable , Text>.
        Context context) throws IOException ,
        InterruptedException {
7         String line = value.toString();
8         String[] words = line.split("\\t");
9         id.set(words[0]);
10        num.set(Long.parseLong(words[1]));
11        context.write(num, id);
12    }
13 }
```

```
1 public class SortReducer extends Reducer<LongWritable ,
    Text, LongWritable , Text> {
2     @Override
3     protected void reduce(LongWritable key, Iterable<
        Text> values , Reducer<LongWritable , Text,
        LongWritable , Text>.Context context) throws
        IOException , InterruptedException {
4         for (Text value :
5             values) {
6             context.write(key, value);
```

7	}
8	}
9	}