实验3报告

1. 实验设计说明

本次实验也依旧是全部完成了必做和选做部分。

1. 主要设计思路：

整个分析过程被分成了3个mapReduce任务。

第一个MapReduce负责将输入的有向图转为无向图。具体到题目要求中，需要有向图中节点a和b分别有指向各自的边才能认定为一个无向图边；

第二个MapReduce负责列出需要查询的边。为了避免重复，规定检测出的三角形的三个顶点abc满足a<b<c。例如存在边ab和ac(序号满足a<b<c)，那么就输出一条待检测的边bc；

第三个MapReduce过程进行最后的计算。

2. 算法设计：

第一个Map-Reduce中，map读入一条边之后，将其小编号在前的形式作为key，边的朝向作为value。例如a<-b的value就是1，a->b的value是2。

Reduce中，key就是当前要检测的无向边，value中有多少项就代表这条边(忽略方向)被找到了多少次。不过次数并不重要，重点在于是否value=0和value=1的都出现过，都有则将这条边输出。

这部分主要是Map比较难。Map部分代码如下：

@Override  
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 StringTokenizer itr = new StringTokenizer(value.toString());  
 String a = itr.nextToken();  
 String b = itr.nextToken();  
 //最终存储时，a一定是小编号，b一定是大编号。  
 //flag是节点序标记，取值=1表示大指向小，=0表示小指向大  
 String flag = "0";  
 if (!a.equals(b)) {  
 if (a.compareTo(b) > 0) {  
 flag = "1";  
 String tmp = a;  
 a = b;  
 b = tmp;  
 }  
 context.write(new Text(a + '\t' + b), new Text(flag));  
 }  
}

第二个MapReduce中，需要列出要查询的边。这里Mapper什么都不干，只是读入边，key为边的较小编号顶点，value为边的较大编号顶点；reduce中则使用一个二重循环，遍历当前key的所有邻居，并将可能构成三角形的需检测边输出到context。

另外，这边的reduce输出由于同时需要保留无向图的边的信息又要保留需要查询的边的信息，为了避免将两者弄混，定义key值为边(例如ab之间的边使用”a+b”表示，中间用+号隔开)，value为“+”表示原无向图中的边，为“-”表示需要查询的边。

Reduce部分代码如下：

@Override  
protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {  
 ArrayList<String> dest = new ArrayList<>();  
 Text curKey = new Text();  
 Text curValue = new Text();  
 curValue.set("+");  
 for(Text value:values){  
 dest.add(value.toString());  
 curKey.set(key.toString()+"+"+value.toString());  
 context.write(curKey,curValue);  
 }  
 curValue.set("-");  
 int size = dest.size();  
 for(int i=0;i<size;++i){  
 for(int j=i+1;j<size;++j){  
 curKey.set(dest.get(i)+"+"+dest.get(j));  
 context.write(curKey,curValue);  
 }  
 }  
}

第三个MapReduce中进行最后的统计工作。

具体地，第二步输出的边都是唯一的(小序号在前)，所以mapper什么都不做只是读入输入，reduce中的key值即为当前的边，value中有“+“代表这条边实际存在，有多少个“-”就代表这条边可以组成多少个不重复的三角形。

Reducer代码如下：

public static class Reduce3 extends Reducer<Text, Text,Text,Text>{  
 private int result = 0;  
 @Override  
 public void cleanup(Context context)throws IOException, InterruptedException{  
 context.write(new Text("result: "),new Text(""+result));  
 }  
 @Override  
 protected void reduce(Text key, Iterable<Text> values, Reducer<Text, Text, Text, Text>.Context context) throws IOException, InterruptedException {  
 int cnt = 0;  
 boolean flag = false;  
 for(Text value:values){  
 if(value.toString().equalsIgnoreCase("+")){  
 flag = true;  
 }else{  
 cnt++;  
 }  
 }  
 if(flag) result+=cnt;  
 }  
}

3. 选做部分的算法设计：

由于选做中只是修改了从有向图到无向图的转换，所以只需要修改第一部分的MapReduce即可。

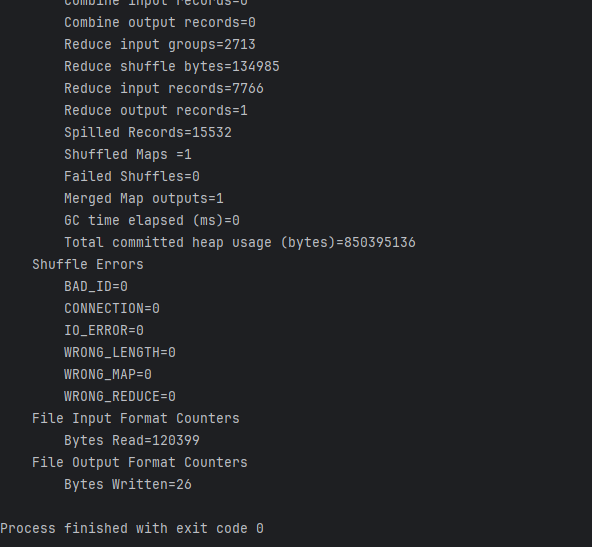
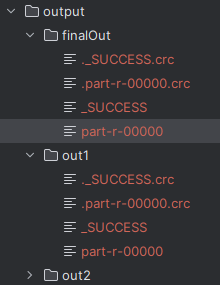
具体地，去除map中的flag部分对于边的朝向的标识，只去除自环的情况；在reduce中直接将key值输出即可。

Map部分的代码：

protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {  
 StringTokenizer itr = new StringTokenizer(value.toString());  
 String a = itr.nextToken();  
 String b = itr.nextToken();  
 //最终存储时，a一定是小编号，b一定是大编号。  
 //flag是节点序标记，取值=1表示大指向小，=0表示小指向大  
 if (!a.equals(b)) {  
 if (a.compareTo(b) > 0) {  
 String tmp = a;  
 a = b;  
 b = tmp;  
 }  
 context.write(new Text(a + '\t' + b), new Text("0"));  
 }  
}

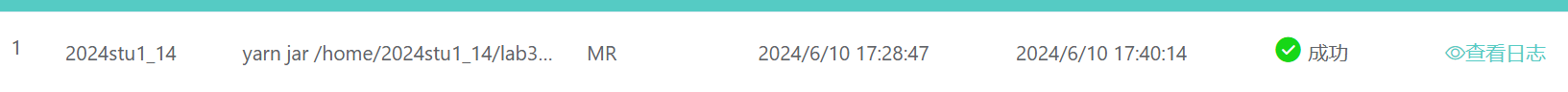
**2. 程序运行和实验结果说明和分析**

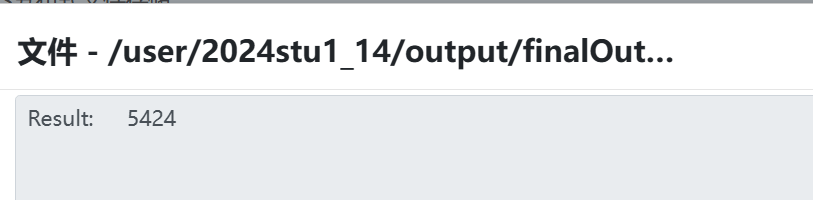
本地运行截图：（必做部分）

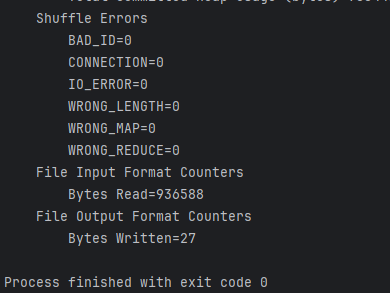
最终使用 web-Google-small.txt 统计出的三角形个数为：**5424**个。

平台运行截图：（必做部分）





选做部分，本地运行截图：



最终使用 web-Google-small.txt 统计出的三角形个数为：**14890**个。

平台执行截图：

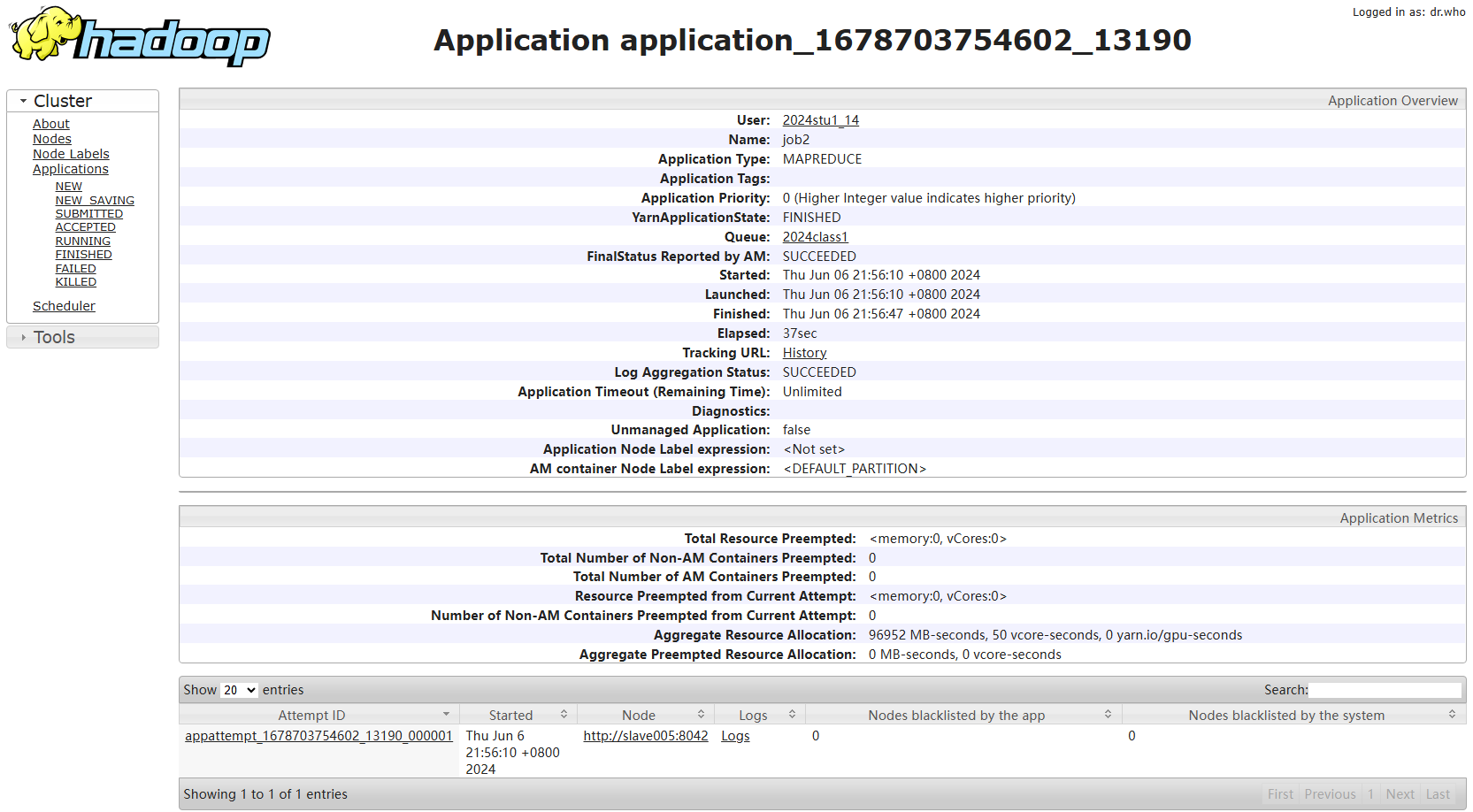


3. 不足与可改进之处

Map2和Map3都没有实际的用处，只是为Reduce服务，平白多出来了两个步骤可能导致开销变大。通过合适的算法设计可能能除去这两个步骤。

4. WebUI 执行报告内容

使用small数据集进行必做部分的实验：（由于分成了三个job，这里只展示第一个）



选做部分的实验：

