图的三角形计数

一、小组信息：

2016st19：张瑞 MF1533072、董茂健 MG1533007、杨杰 MG1533075、金熠波 131220168

二、实验环境

Ubuntu 12.04.2、Java 1.8.0_102、Maven 3.3.9、IntelliJ 15.06

三、设计思路

（1）IF(A->B) OR (B->A) THEN A-B

Dereplication：用于去重边以及筛选可能的三角形组（已看到两条边）

Map：

输入<object, Text>，输出：<Text, IntWritable>

从源文件读出所有非重边并按 u#v，u < v 的形式输出

Reduce：

输入<Text, IntWritable>，输出：< Text, Text>

当发现出现 u#v，且出现 u#w 时，输出<v#w, need>表示需要此边

且输出<u#v, found>，<u#w, found>表示存在边

TriCount：

Map：

输入<object, Text>，输出：<Text, Text>

从文件中读出存在的边和需要的边

Reduce：

输入<Text, Text >，输出：< Text, Text>

如既发现某一条边存在，又发现对某一条边有需求，则需求数相加就是依赖这条边的三角形个数，输出每条边存在情况下的需求数

CollectCount：

Map：

输入<object, Text>，输出：<Text, IntWritable>

读入每条边存在情况下的需求数

Reduce：

输入< Text, IntWritable >，输出：<Text, IntWritable>

对所有需求数相加即为总和

（2）IF(A->B) AND (B->A) THEN A-B

Dereplication：用于去重边以及筛选可能的三角形组（已看到两条边）

Map：

输入<object, Text>，输出：<Text, IntWritable>

从源文件读出所有非重边<u, v>

并输出存在此边<u#v, found>和需要其另一条边<v#u, need>的信息

Reduce：

输入<Text, IntWritable>，输出：< Text, Text>

只考虑 u#v（u < v）的情况，另一种情况可不用考虑，因为此情况与之重复，若<u#v, found>和<u#v, need>同时存在则输出该边<u#v>存在

FindPair：

Map：

输入<object, Text>，输出：<Text, IntWritable>

读出所有存在的边<u#v>，满足（u < v）

Reduce：

输入<Text, IntWritable>，输出：< Text, Text>

当发现出现 u#v，且出现 u#w 时，输出<v#w, need>表示需要此边

且输出<u#v, found>，<u#w, found>表示存在边

TriCount：

Map：

输入<object, Text>，输出：<Text, Text>

从文件中读出存在的边和需要的边

Reduce：

输入<Text, Text >，输出：< Text, Text>

如既发现某一条边存在，又发现对某一条边有需求，则需求数相加就是依赖这条边的三角形个数，输出每条边存在情况下的需求数

CollectCount：

Map：

输入<object, Text>，输出：<Text, IntWritable>

读入每条边存在情况下的需求数。

Reduce：

输入< Text, IntWritable >，输出：<Text, IntWritable>

对所有需求数相加即为总和

四、实现细节（算法实现及各类作用）

（1）IF(A->B) OR (B->A) THEN A-B

Dereplication：

Map：读出边并按序输出，忽略重边

```
25      public void map(Object key, Text value, Context context)
26              throws IOException, InterruptedException {
27          StringTokenizer itr = new StringTokenizer(value.toString());
28          while(itr.hasMoreTokens()){
29              String u = itr.nextToken().toString();
30              String v = itr.nextToken().toString();
31              if(!u.equals(v)){
32                  if(u.compareTo(v) < 0) word.set(u + "#" + v);
33                  else word.set(v + "#" + u);
34                  context.write(word, one);
35              }
36          }
37      }
```

Combiner：对同一边只输出一次

```
40  public static class DereplicationCombiner
41          extends Reducer<Text, IntWritable, Text, IntWritable>{
42      private final static IntWritable one = new IntWritable(1);
43
44      public void reduce(Text key, Iterable<IntWritable> values, Context context)
45              throws IOException, InterruptedException {
46          context.write(key, one);
47      }
48  }
```

Partitioner：将边中序号小的一方作为 Partition 划分的依据

```java
public static class DereplicationNewPartitioner
        extends HashPartitioner<Text, IntWritable>{
    public int getPartition(Text key, IntWritable value, int numReduceTasks){
        String term;
        term = key.toString().split("#")[0];
        return super.getPartition(new Text(term), value, numReduceTasks);
    }
}
```

Reduce：只处理 u < v 的边，将所有 u 的对点加入列表。遍历列表，输出已有边<u, v>，<u, w>并输出需求边<v, w>

```java
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException{
    String[] split = key.toString().split("#");
    if(split[0].compareTo(split[1]) > 0) return;
    word.set(split[0]);
    if((!CurrentItem.equals(word)) && (!CurrentItem.equals("*"))){
        myOutPut(context);
        edgeSet = new ArrayList<String>();
    }
    CurrentItem = new Text(word);
    edgeSet.add(split[1]);
}
```

```java
@Override
protected void cleanup(Context context) throws IOException, InterruptedException{
    myOutPut(context);
}
```

```java
private static void myOutPut(Context reducerContext) throws IOException, InterruptedException {
    for(int i = 0; i < edgeSet.size(); ++i){
        String u = edgeSet.get(i);
        reducerContext.write(new Text(CurrentItem + "#" + u), found);
        for(int j = (i + 1); j < edgeSet.size(); ++j){
            String v = edgeSet.get(j);
            if(u.compareTo(v) > 0) continue;
            reducerContext.write(new Text(u + "#" + v), need);
        }
    }
}
```

TriCount：

Map：读入对边进行需求和已存在的分类

```java
public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while(itr.hasMoreTokens()){
        String k = itr.nextToken().toString();
        String v = itr.nextToken().toString();
        word.set(k);
        if(v.equals(found)) context.write(word, new Text("#"));
        else context.write(word, new Text("1"));
    }
}
```

Combiner：对需求进行类和；对存在进行归并，只输出一份

```
38      public static class TriCountCombiner
39              extends Reducer<Text, Text, Text, Text>{
40          public void reduce(Text key, Iterable<Text> values, Context context)
41                  throws IOException, InterruptedException {
42              int sum = 0;
43              boolean isExist = false;
44              for (Text val : values) {
45                  if(val.toString().equals("#")) isExist = true;
46                  else sum += Integer.valueOf(val.toString());
47              }
48              if(isExist) context.write(key, new Text("#"));
49              if(sum > 0) context.write(key, new Text(Integer.toString(sum)));
50          }
51      }
```

Partitioner：无特殊处理

```
53      public static class TriCountNewPartitioner
54              extends HashPartitioner<Text, Text>{
55          public int getPartition(Text key, Text value, int numReduceTasks){
56              return super.getPartition(key, value, numReduceTasks);
57          }
58      }
```

Reduce：统计需求的数目，当出现存在边，在此边遍历完毕后加入全局计数，并输出全局计数。

```
68          public void reduce(Text key, Iterable<Text> values, Context context)
69                  throws IOException, InterruptedException{
70              word.set(key);
71              if((!CurrentItem.equals(word)) && (!CurrentItem.equals("*"))){
72                  myTreat();
73                  bufRes = 0;
74                  isExist = false;
75              }
76              CurrentItem = new Text(word);
77              for(Text v : values){
78                  if(v.toString().equals("#")) isExist = true;
79                  else bufRes += Integer.valueOf(v.toString());
80              }
81          }
```

```
83          @Override
84          protected void cleanup(Context context) throws IOException, InterruptedException{
85              myTreat();
86              context.write(new Text("Total:"), new Text(Integer.toString(result)));
87          }
```

```
89          private static void myTreat() { if(isExist) result += bufRes; }
92      }
```

CollectCount：

Map：读入上一个 job 输出的各个三角形总数，并发射包含计数的键值对

```
20          public void map(Object key, Text value, Context context)
21                  throws IOException, InterruptedException {
22              StringTokenizer itr = new StringTokenizer(value.toString());
23              while(itr.hasMoreTokens()){
24                  String k = itr.nextToken().toString();
25                  String v = itr.nextToken().toString();
26                  context.write(new Text("1"), new IntWritable(Integer.valueOf(v)));
27              }
28          }
```

Combiner：对局部键值对进行累加

```
31      public static class CollectCountCombiner
32              extends Reducer<Text, IntWritable, Text, IntWritable>{
33          public void reduce(Text key, Iterable<IntWritable> values, Context context)
34                  throws IOException, InterruptedException {
35              int sum = 0;
36              for (IntWritable val : values) {
37                  sum += val.get();
38              }
39              context.write(new Text("1"), new IntWritable(sum));
40          }
41      }
```

Reduce：将所有键值对进行累和，输出最终结果

```
47          public void reduce(Text key, Iterable<IntWritable> values, Context context)
48                  throws IOException, InterruptedException{
49              for(IntWritable v : values){
50                  result += v.get();
51              }
52          }
```

```
54          @Override
55          protected void cleanup(Context context) throws IOException, InterruptedException{
56              context.write(new Text("Total Triangle is:"), new Text(Integer.toString(result)));
57          }
```

Driver：将不同的 job 串起来，设置输入文件夹目录

```
5       public static void main(String[] args) throws Exception{
6           String inputPath = args[0];
7           String outPath = args[1] + "/buf/Dereplication";
8           Dereplication.main(inputPath, outPath);
9           inputPath = outPath;
10          outPath = args[1] + "/buf/TriCount";
11          TriCount.main(inputPath, outPath);
12          inputPath = outPath;
13          outPath = args[1] + "/result";
14          CollectCount.main(inputPath, outPath);
15      }
```

（2）IF(A->B) AND (B->A) THEN A-B

此处只列出不同的实现

Dereplication：

Map：读入一条边，生成其存在边和其相对应的另一条反向需求边

```
27          public void map(Object key, Text value, Context context)
28                  throws IOException, InterruptedException {
29              StringTokenizer itr = new StringTokenizer(value.toString());
30              while(itr.hasMoreTokens()){
31                  String u = itr.nextToken().toString();
32                  String v = itr.nextToken().toString();
33                  if(!u.equals(v)){
34                      word1.set(u + "#" + v);
35                      word2.set(v + "#" + u);
36                      context.write(word1, one);
37                      context.write(word2, zero);
38                  }
39              }
40          }
```

Combiner：统计局部一条边，将所有重复的出现边、需求边归一

```
43      public static class DereplicationCombiner
44              extends Reducer<Text, IntWritable, Text, IntWritable>{
45          public void reduce(Text key, Iterable<IntWritable> values, Context context)
46                  throws IOException, InterruptedException {
47              boolean one = false, zero =false;
48              for(IntWritable v : values){
49                  if(one && zero) break;
50                  if((!one) && (v.get() == 1)){
51                      one = true;
52                      context.write(key, new IntWritable(1));
53                  }
54                  if((!zero) && (v.get() == 0)){
55                      zero = true;
56                      context.write(key, new IntWritable(0));
57                  }
58              }
59          }
60      }
```

Partitioner：无特殊处理

Reduce：当一条边<u, v>（按序）既有存在边又有需求边，即输出一条无向边

```
75          public void reduce(Text key, Iterable<IntWritable> values, Context context)
76                  throws IOException, InterruptedException{
77              String[] split = key.toString().split("#");
78              if(split[0].compareTo(split[1]) < 0){
79                  word.set(key);
80                  if((!CurrentItem.equals(word)) && (!CurrentItem.equals("*"))){
81                      myOutPut(context);
82                      positive = negative = false;
83                  }
84                  CurrentItem = new Text(word);
85                  for(IntWritable v : values){
86                      if((positive) && (negative)) break;
87                      if((!positive) && (v.get() == 1)) positive = true;
88                      if((!negative) && (v.get() == 0)) negative = true;
89                  }
90              }
91          }
```

```
93          @Override
94          protected void cleanup(Context context) throws IOException, InterruptedException{
95              myOutPut(context);
96          }
97
98          private static void myOutPut(Context reducerContext) throws IOException, InterruptedException {
99              if((positive) && (negative)){
100                 reducerContext.write(CurrentItem, new Text("1"));
101             }
102         }
103     }
```

FindPair：

Map：读入一条无向边<u, v>按序

```
24          public void map(Object key, Text value, Context context)
25                  throws IOException, InterruptedException {
26              StringTokenizer itr = new StringTokenizer(value.toString());
27              while(itr.hasMoreTokens()){
28                  String u = itr.nextToken().toString();
29                  String v = itr.nextToken().toString();
30                  word.set(u + "#" + v);
31                  context.write(word, one);
32              }
33          }
```

Combiner：无特殊处理

Partitioner：将序小的点作为 Partition 划分

```java
public static class FindPairNewPartitioner
        extends HashPartitioner<Text, IntWritable>{
    public int getPartition(Text key, IntWritable value, int numReduceTasks){
        String term;
        term = key.toString().split("#")[0];
        return super.getPartition(new Text(term), value, numReduceTasks);
    }
}
```

Reduce：与 Or 逻辑下 Dereplication 中 Reduce 的功能一致。只处理 u < v 的边，将所有 u 的对点加入列表。遍历列表，输出已有边<u, v>，<u, w>并输出需求边<v, w>

```java
public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException{
    String[] split = key.toString().split("#");
    if(split[0].compareTo(split[1]) > 0) return;
    word.set(split[0]);
    if((!CurrentItem.equals(word)) && (!CurrentItem.equals("*"))){
        myOutPut(context);
        edgeSet = new ArrayList<String>();
    }
    CurrentItem = new Text(word);
    edgeSet.add(split[1]);
}
```

```java
@Override
protected void cleanup(Context context) throws IOException, InterruptedException{
    myOutPut(context);
}
```

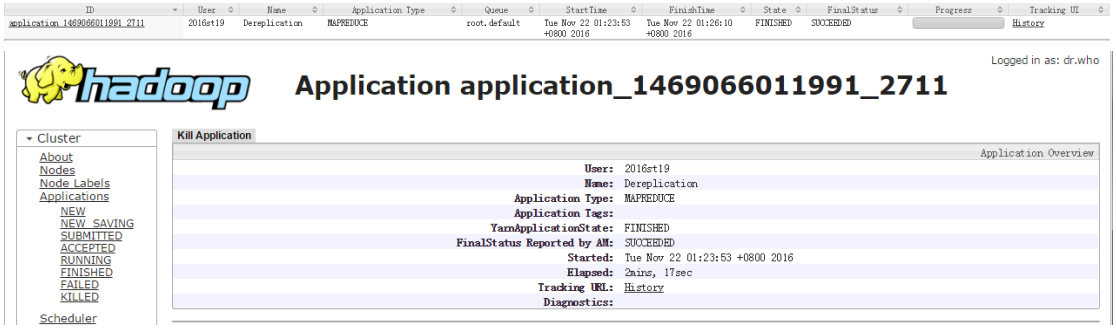
```java
private static void myOutPut(Context reducerContext) throws IOException, InterruptedException {
    for(int i = 0; i < edgeSet.size(); ++i){
        String u = edgeSet.get(i);
        reducerContext.write(new Text(CurrentItem + "#" + u), found);
        for(int j = (i + 1); j < edgeSet.size(); ++j){
            String v = edgeSet.get(j);
            if(u.compareTo(v) > 0) continue;
            reducerContext.write(new Text(u + "#" + v), need);
        }
    }
}
```

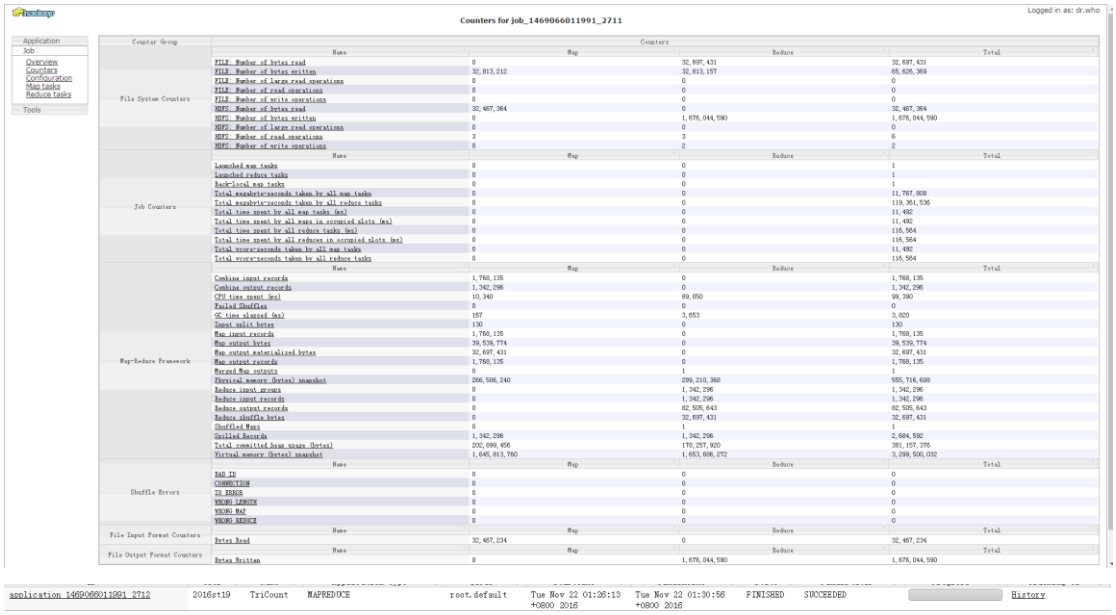
五、实验结果（非优化版本，优化见性能拓展部分，同时 Or 下的 Google 由于时间关系暂无结果，详细原因见性能分析部分）

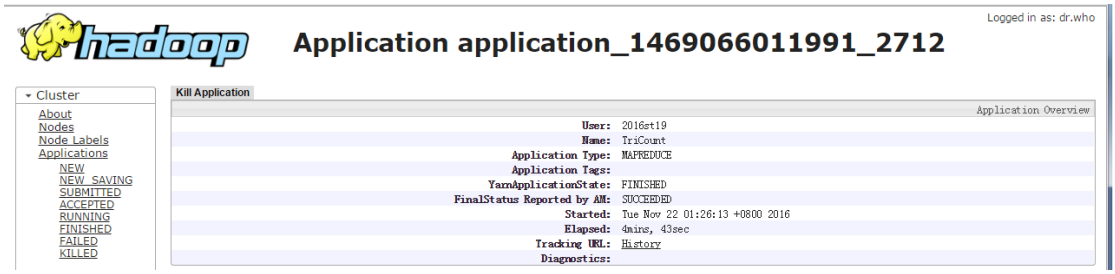
数据集（Or 逻辑）	三角形个数	耗时
Twitter	13082506	7 分 17 秒
Google+	暂无	耗时过长，任务经常失败
数据集（And 逻辑）	三角形个数	耗时
Twitter	1818304	11 分 08 秒
Google+	27018510	31 分 20 秒

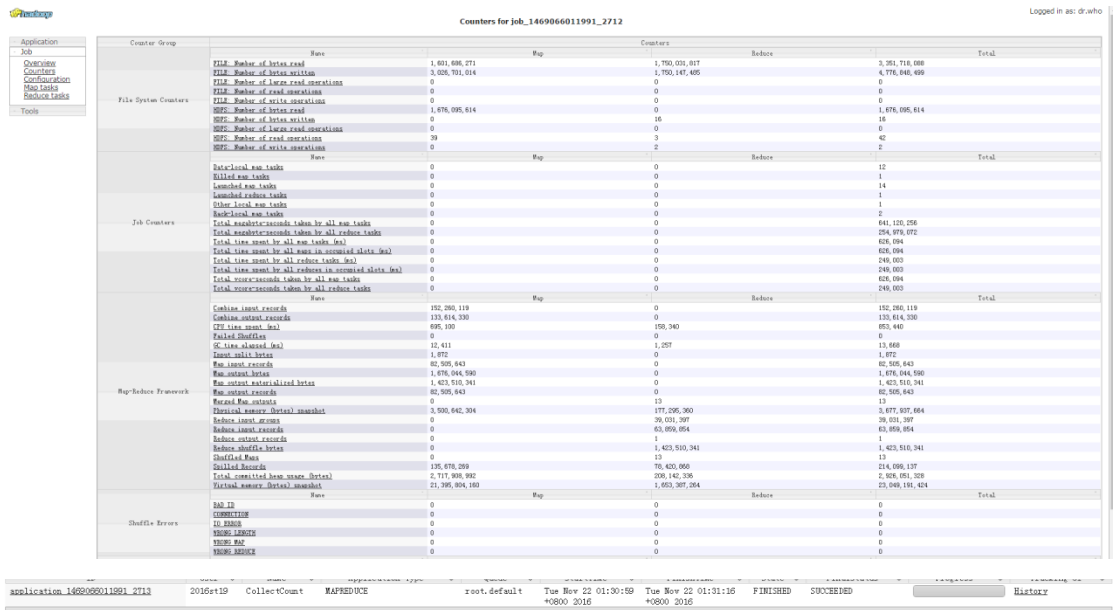
Or 逻辑：

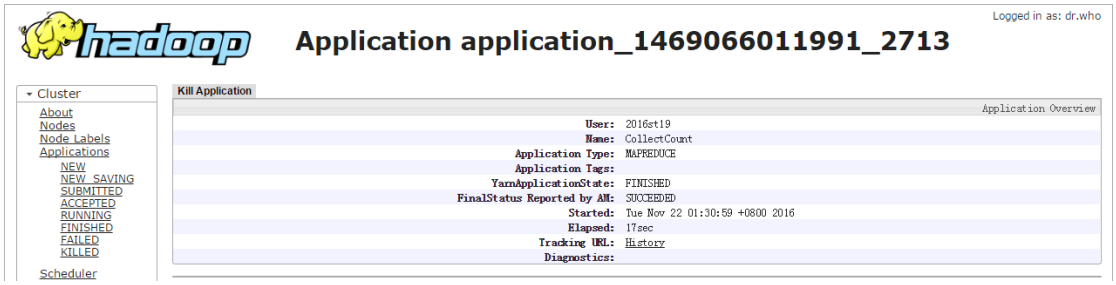
Twitter：

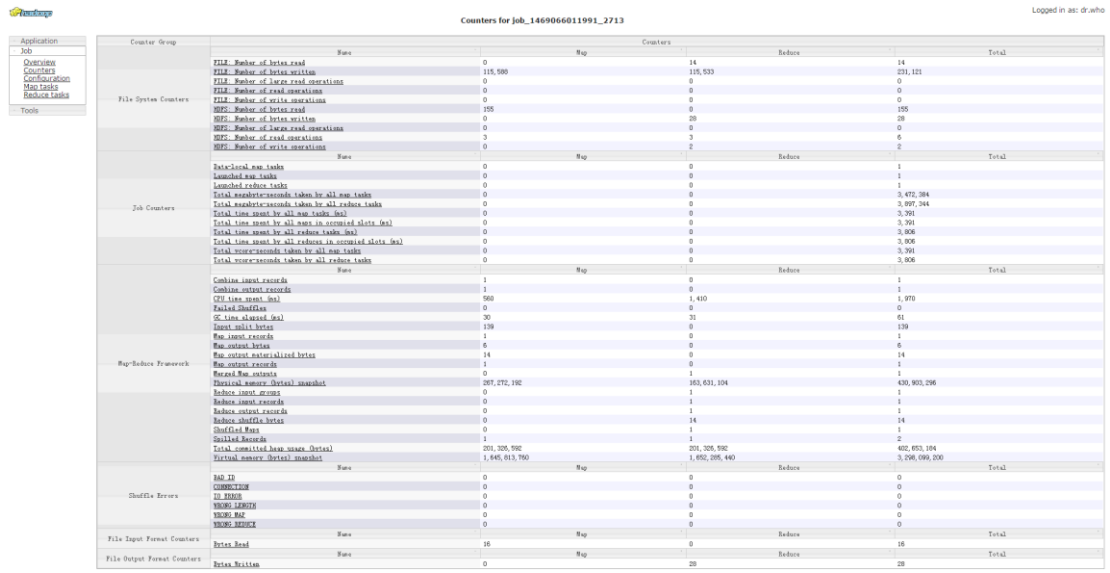



Counters for job_1469066011991_2711
Logged in as: dr.who
application_1469066011991_2712 2016st19 TriCount MAPREDUCE root.default Tue Nov 22 01:26:13 +0800 2016 Tue Nov 22 01:30:56 +0800 2016 FINISHED SUCCEEDED History


Logged in as: dr.who
Application application_1469066011991_2712
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Kill Application
Application Overview
User: 2016st19
Name: TriCount
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
FinalStatus Reported by AM: SUCCEEDED
Started: Tue Nov 22 01:26:13 +0800 2016
Elapsed: 4mins, 43sec
Tracking URL: History
Diagnostics:


Counters for job_1469066011991_2712
Logged in as: dr.who
application_1469066011991_2713 2016st19 CollectCount MAPREDUCE root.default Tue Nov 22 01:30:59 +0800 2016 Tue Nov 22 01:31:16 +0800 2016 FINISHED SUCCEEDED History

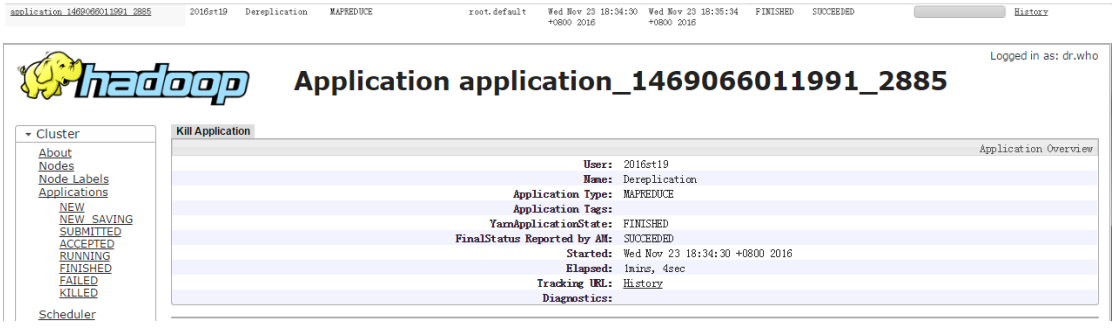

Logged in as: dr.who
Application application_1469066011991_2713
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Kill Application
Application Overview
User: 2016st19
Name: CollectCount
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
FinalStatus Reported by AM: SUCCEEDED
Started: Tue Nov 22 01:30:59 +0800 2016
Elapsed: 17sec
Tracking URL: History
Diagnostics:

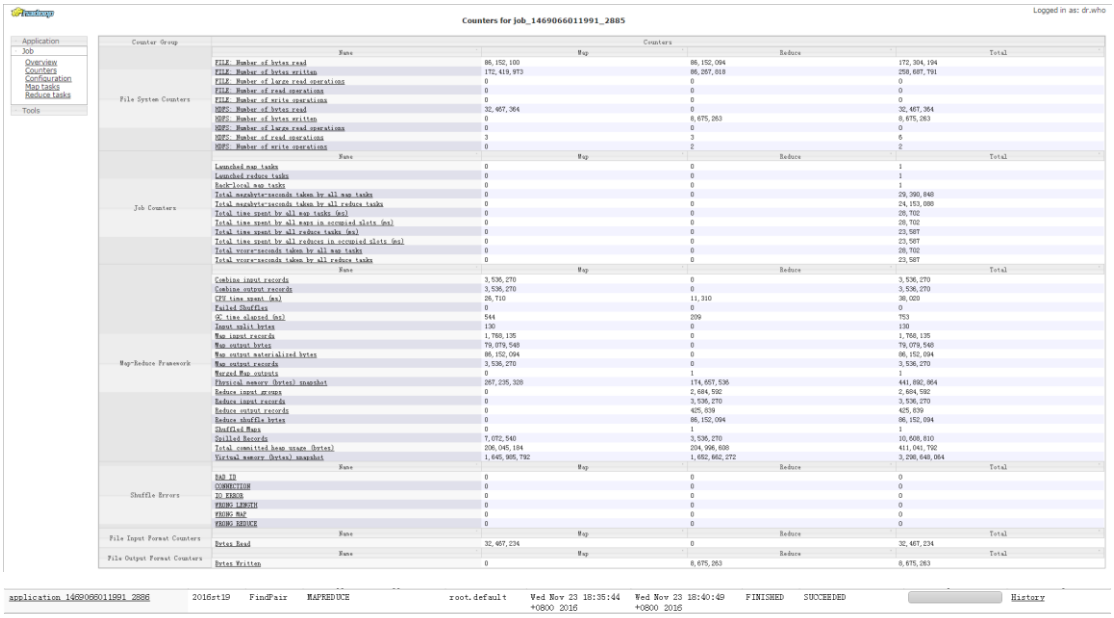


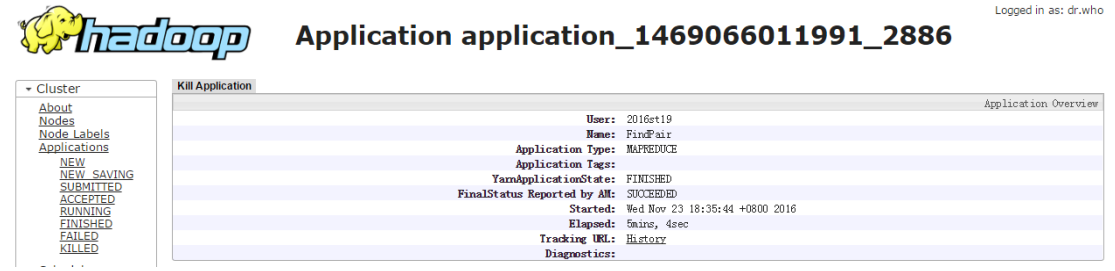

```
[2016st19@master01 ~]$ hdfs dfs -cat /user/2016st19/TriangleOrOut/result/part-r-00000
16/11/21 21:27:19 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
where applicable
Total Triangle is:      13082506
[2016st19@master01 ~]$
```

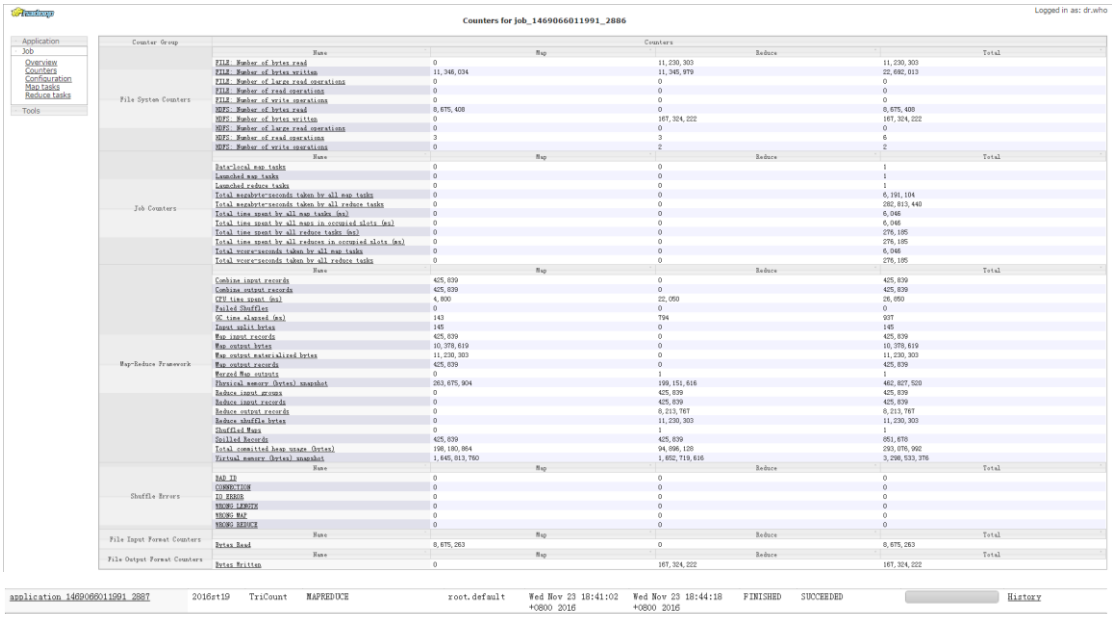
And 逻辑：

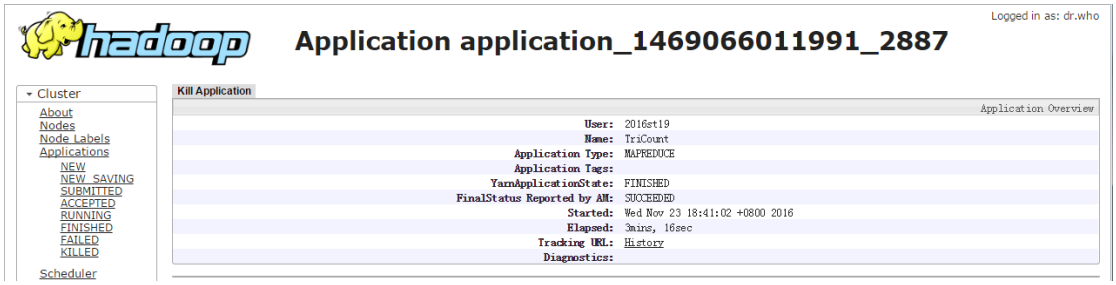
Twitter：

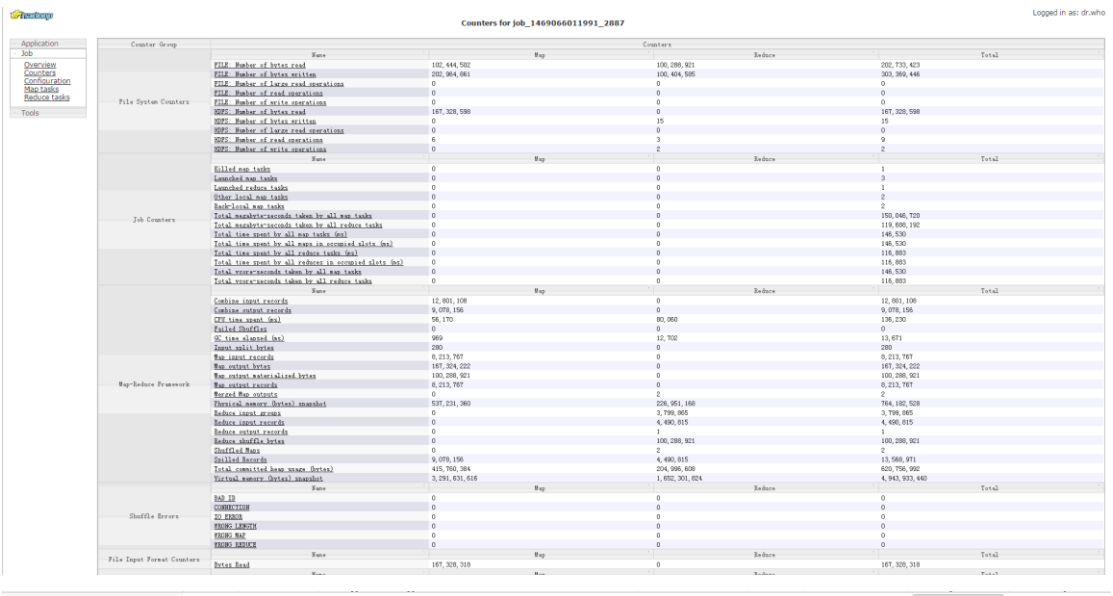


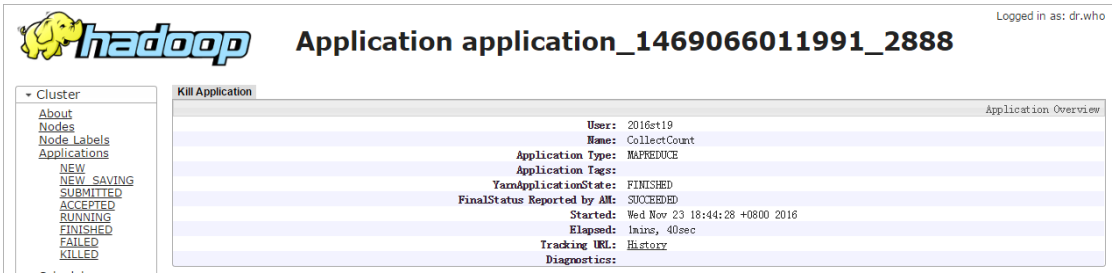


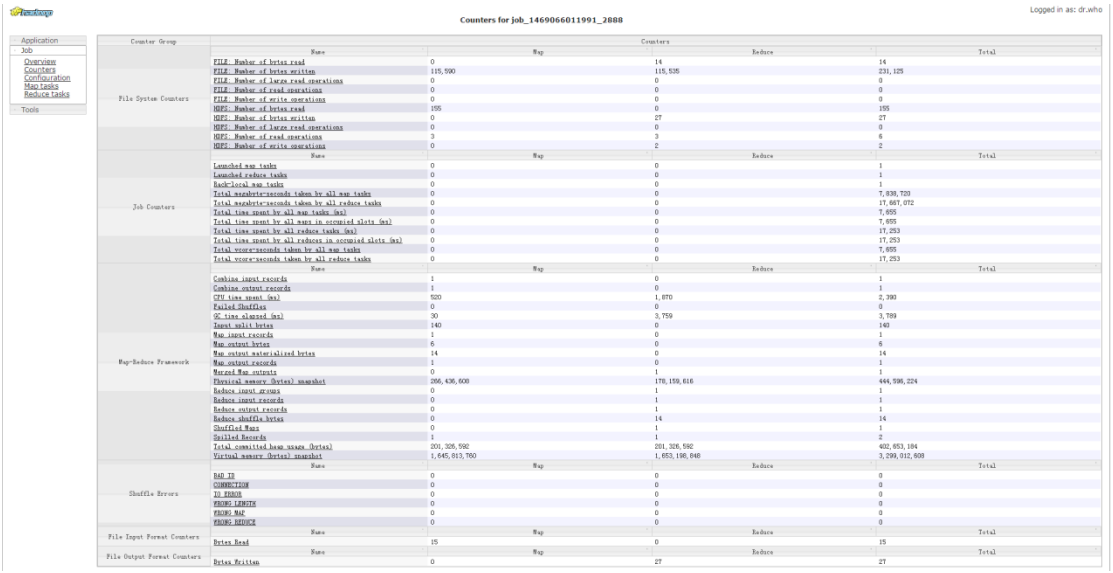

Logged in as: dr.who
Application application_1469066011991_2886
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Kill Application
Application Overview
User: 2016st19
Name: FindPair
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
FinalStatus Reported by AM: SUCCEEDED
Started: Wed Nov 23 18:35:44 +0800 2016
Elapsed: 5mins, 4sec
Tracking URL: History
Diagnostics:


Counters for job_1469066011991_2886
Logged in as: dr.who
application_1469066011991_2887 2016st19 TriCount MAPREDUCE root.default Wed Nov 23 18:41:02 +0800 2016 Wed Nov 23 18:44:18 +0800 2016 FINISHED SUCCEEDED History


Logged in as: dr.who
Application application_1469066011991_2887
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Kill Application
Application Overview
User: 2016st19
Name: TriCount
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
FinalStatus Reported by AM: SUCCEEDED
Started: Wed Nov 23 18:41:02 +0800 2016
Elapsed: 3mins, 16sec
Tracking URL: History
Diagnostics:


Counters for job_1469066011991_2887
Logged in as: dr.who
application_1469066011991_2888 2016st19 CollectCount MAPREDUCE root.default Wed Nov 23 18:44:28 +0800 2016 Wed Nov 23 18:46:09 +0800 2016 FINISHED SUCCEEDED History

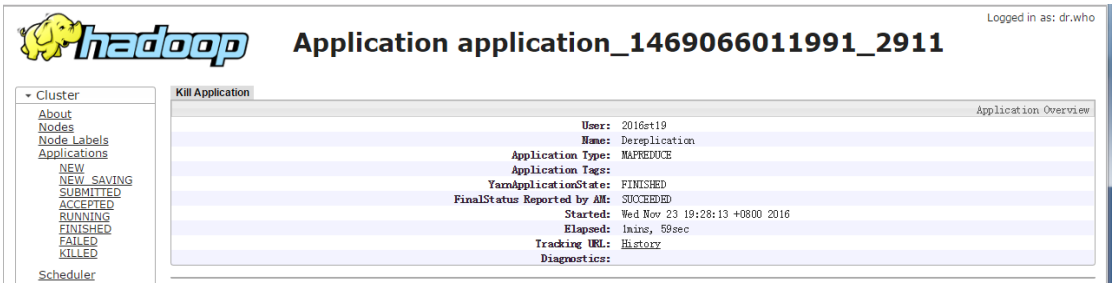


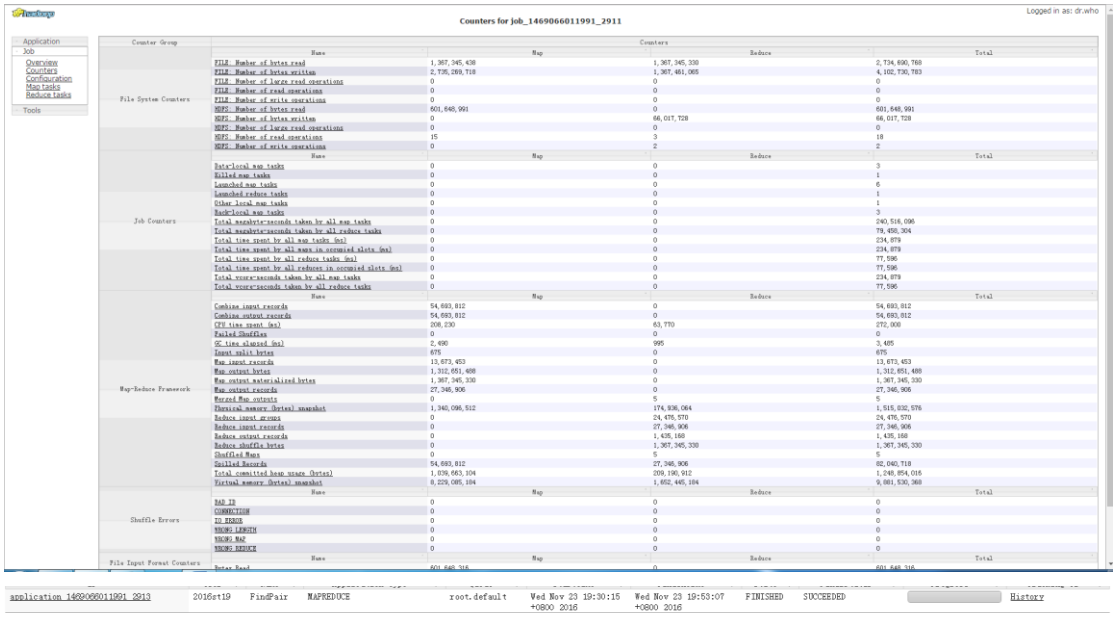


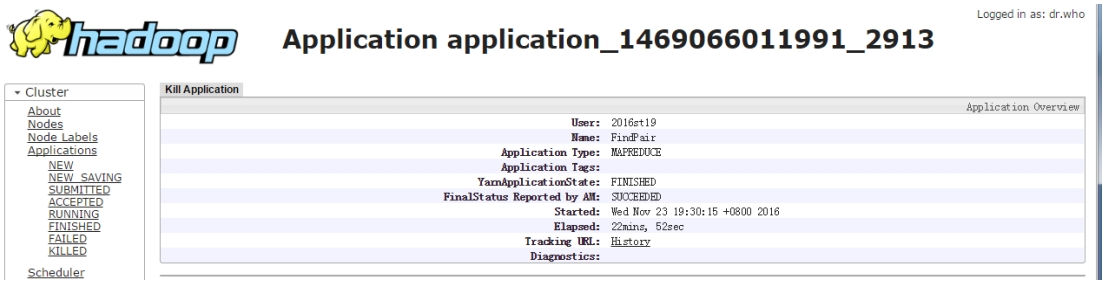

```
[2016st19@master01 ~]$ hdfs dfs -cat /user/2016st19/TriangleAndOut/result/part-r-00000
16/11/23 18:59:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes
 where applicable
Total Triangle is:      1818304
[2016st19@master01 ~]$
```

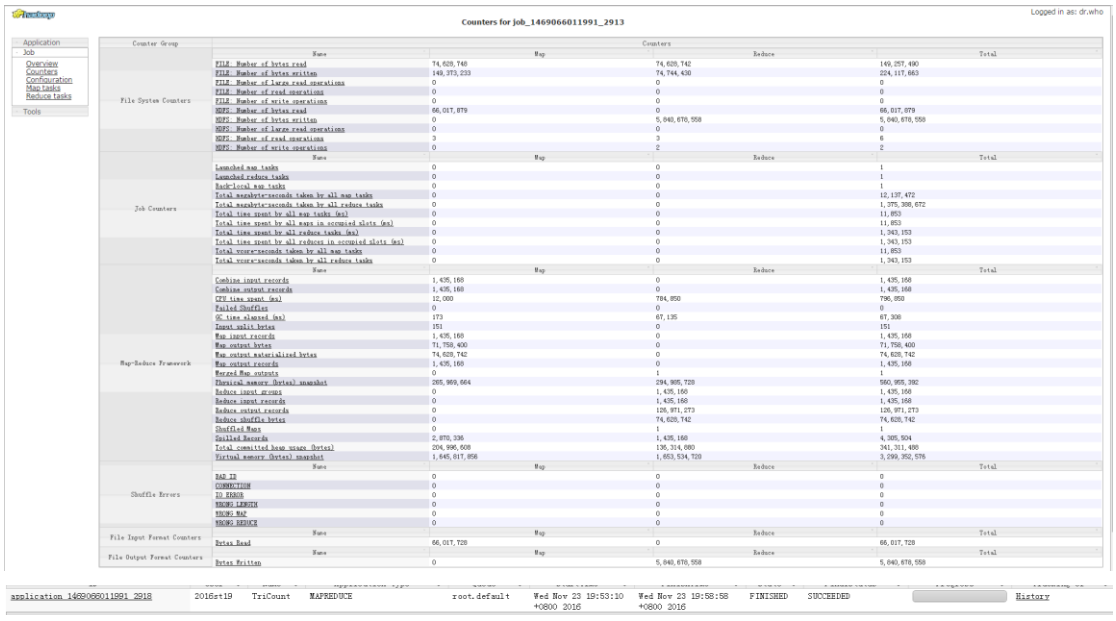
Google+：

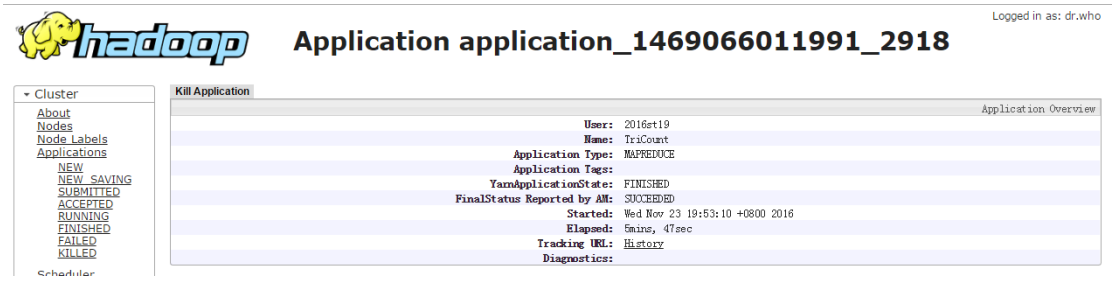
application_1469066011991_2911	2016st19	Dereplication	MAPREDUCE	root.default	Wed Nov 23 19:28:13 +0800 2016	Wed Nov 23 19:30:13 +0800 2016	FINISHED	SUCCEEDED	History

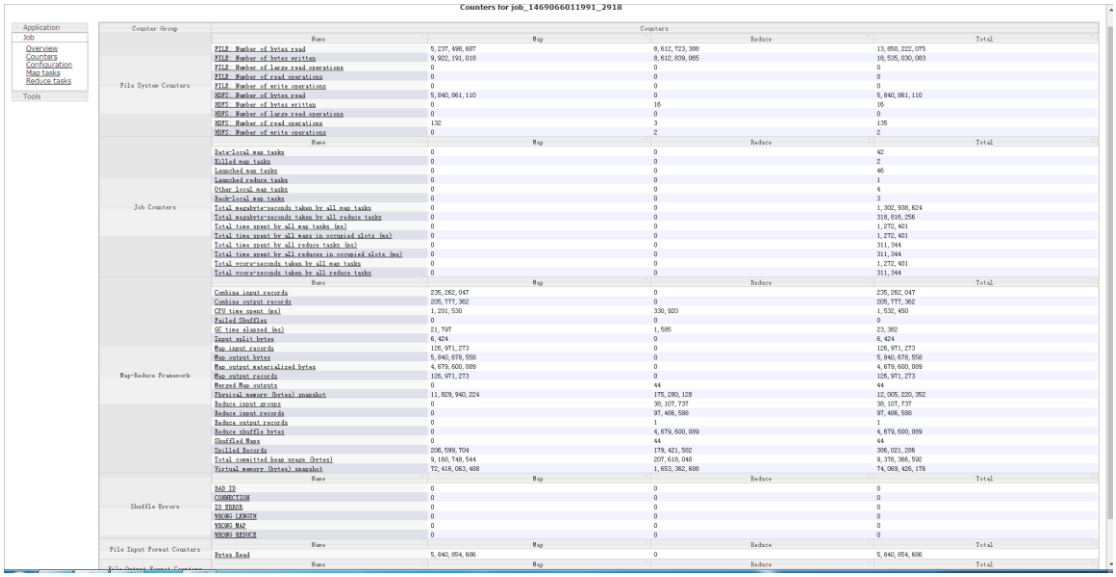


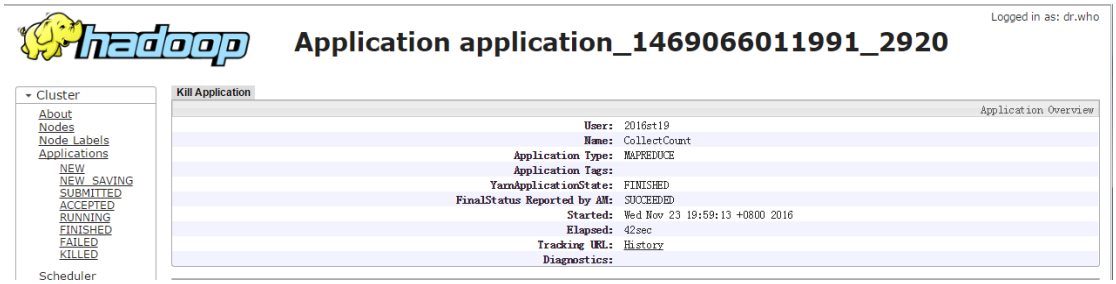

Counters for job_1469066011991_2911
Logged in as: dr.who
application_1469066011991_2913 2016st19 FindPair MAPREDUCE root.default Wed Nov 23 19:30:15 +0800 2016 Wed Nov 23 19:53:07 +0800 2016 FINISHED SUCCEEDED History

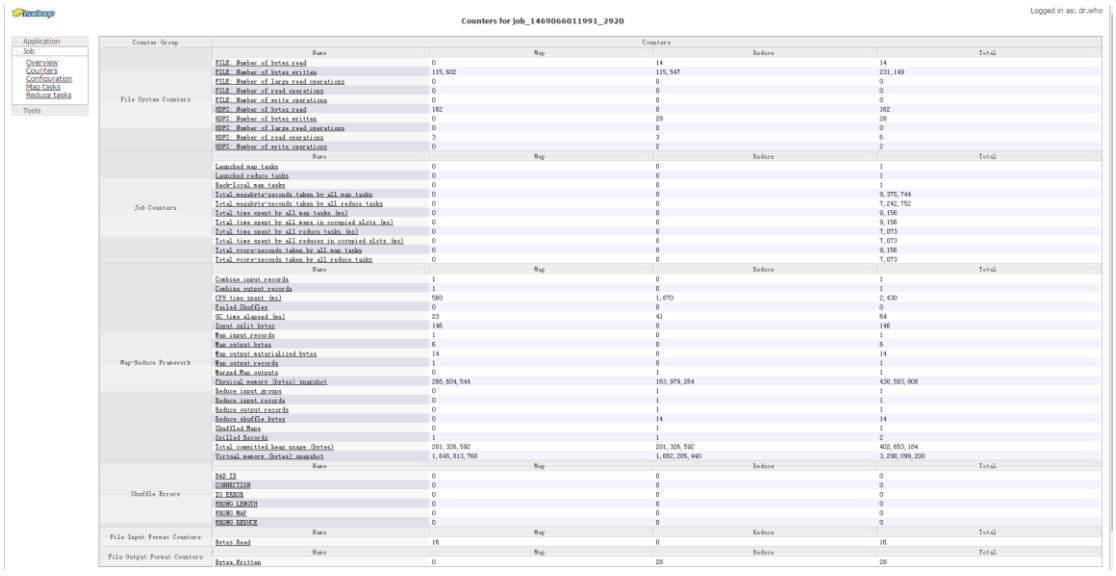

Logged in as: dr.who
Application application_1469066011991_2913
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Kill Application
Application Overview
User: 2016st19
Name: FindPair
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
FinalStatus Reported by AM: SUCCEEDED
Started: Wed Nov 23 19:30:15 +0800 2016
Elapsed: 22mins, 52sec
Tracking URL: History
Diagnostics:


Counters for job_1469066011991_2913
Logged in as: dr.who
application_1469066011991_2918 2016st19 TriCount MAPREDUCE root.default Wed Nov 23 19:53:10 +0800 2016 Wed Nov 23 19:58:58 +0800 2016 FINISHED SUCCEEDED History


Logged in as: dr.who
Application application_1469066011991_2918
Cluster
About
Nodes
Node Labels
Applications
NEW
NEW_SAVING
SUBMITTED
ACCEPTED
RUNNING
FINISHED
FAILED
KILLED
Scheduler
Kill Application
Application Overview
User: 2016st19
Name: TriCount
Application Type: MAPREDUCE
Application Tags:
YarnApplicationState: FINISHED
FinalStatus Reported by AM: SUCCEEDED
Started: Wed Nov 23 19:53:10 +0800 2016
Elapsed: 5mins, 47sec
Tracking URL: History
Diagnostics:








```
[2016st19@master01 ~]$ hdfs dfs -cat /user/2016st19/TriangleAndGoogleOut/result/part-r-00000
16/11/23 20:55:58 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where
 applicable
Total Triangle is:      27018510
[2016st19@master01 ~]$
```

六、性能分析

总体来说 And 逻辑的情况运行较快，原因是其在计数前就处理了许多单向边，这些单向边不会构成三角形（无反向边使其成为无向边）。而 Or 逻辑直到计算三角形的前一刻才知道边是否有用，能否构成一个三角形。所以其中间数据很大，几乎包含所有的单向边（我们还对其进行存在和需要的分类，数目更是巨大），所以处理起来耗时，Google 的数据集上需要

几个小时才能完成。经常遇到集群 fail 的情况，见下图：

在运行中经常遇到集群提示无中间目录存在的情况，也经常出现机器对时的情况，再加上执行往往和其他组一起，所以相对会有一定影响。

七、性能、拓展性方面不足及改进

由于无法得知图中点的分布，所以不能很好地估计 reducer 的数目，所以可能成为一个瓶颈。原因是如果某些点的度很大，那么对单个 reducer 的处理压力就会增加，而其也会成为瓶颈。所以希望知道图点的一些分布。

我们的不足也在于没有之前进行提前采样，从而对图的分布有个大致的了解，从而就能对 reducer 的配置进行估计，甚至在算法实现上进行优化。

在我们的实验中，优化前虽是 reduce，但是由于 hadoop 默认 reducer 数目是 1。所以需要手动设置 reducer 数目，但是由于运行时间较长，ssh 登录退出后集群只执行当前任务，且往往出现上图的权限问题。因此暂没有 Google Or 逻辑下的结果。优化如下：

```
97          job.setNumReduceTasks(20);
```

此数目根据 0.95*集群机器数*reducer 数目得出的经验值。由于不知道 reducer slot 数目，所以只能猜测。但是即使这样，我们做了组 twitter 下的 Or 逻辑，发现时间并不占优，原因是启动慢，任务往往要等待很长时间对时或是复制代码至别的机器才能真正执行。User/2016st19/TriangleOrOutOpt 是优化的 twitter Or 逻辑输出。

八、说明及附件

随报告附上 Or 逻辑和 And 逻辑的工程文件夹（无 out 生成目录）和其对应的 jar 包，还有其对应的优化版本*Opt.jar。

Jar 包运行方式（以其中一个为例，输入目录为输入文件位置，输出为目录，其结果会存在此目录下）：

```
[2016st19@master01 ~]$ hadoop jar TriangleAnd.jar /data/graphTriangleCount/twitter_graph_v2.txt /user/2016st19/TriangleAndOut
```