Report of Home Assignment One

Yufan Xu 7909-7439

In this assignment, I firstly calculate the number of nodes, number of edges, and (min, max, average) of out-degree for each node in three different size graph. The graph consists of ID of the node and its adjacent list. After graph statistic, I use PageRank algorithm to calculate the rank of each of node in graph. Finally, I run the same program on both AWS and Google Cloud to compare their performance.

Part I

Firstly, I calculate the number of nodes, the number of edges, minimal out-degree of node, maximal out-degree of node and average out-degree of node in graph. *Picture 1* shows the details of Map-Reduce progress.

```
public class GraphStatistic{
    public static class Map extends Mapper<LongWritable, Text , Text, IntWritable>{
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            StringTokenizer st = new StringTokenizer(line);
            IntWritable res = new IntWritable();
            res.set(st.countTokens()-1);
            //<node, NumofNeighbour>
            context.write(new Text("node"), res);
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text, Text>{
        public void reduce(Text key, Iterable<IntWritable> value, Context context) throws IOException, InterruptedException {
            int numOfNode = 0;
            int numOfEdge = 0;
            int minDegree = Integer.MAX_VALUE;
            int maxDegree = Integer.MIN_VALUE;

            for (IntWritable v : value){
                int temp = v.get();
                numOfNode++;
                numOfEdge += temp;
                minDegree = Math.min(temp, minDegree);
                maxDegree = Math.max(temp, maxDegree);
            }

            float avg = 0;
            if(numOfNode != 0)
                avg = (float) ((numOfEdge)/numOfNode);

            context.write(new Text("Number of edges : "), new Text(Integer.toString(numOfEdge)));
            context.write(new Text("Number of nodes : "), new Text(Integer.toString(numOfNode)));
            context.write(new Text("Minimum out-degree : "), new Text(Integer.toString(minDegree)));
            context.write(new Text("Maximum out-degree : "), new Text(Integer.toString(maxDegree)));
            context.write(new Text("Average out-degree : "), new Text(Float.toString(avg)));
        }
    }
```

Picture 1

Second, the aim is to calculate different ranks of each node in graph. To calculate rank value of a node, I used PageRank algorithm. PageRank is a link analysis algorithm and it assigns a numerical weighting to each element of a hyperlinked set of documents. And I used this expression(*Picture 2*) to calculate rank of each node. The coefficient d is a damping factor. And I set d = 0.85 in my program and initial rank value is 1.

$$PR(A) = 1 - d + d\left(\frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} + \cdots\right).$$

Picture 2

So, at the very beginning, I build the matrix for each graph. One line in matrix means current node with its out-links and rank values. The difficult of calculating rank value in distribute system using Map-Reduce function is to get previous rank value from previous round calculation. So, I handle this problem using store each step's results until each rank value converge. *Picture 3 and 4* shows the details of map function and reduce function.

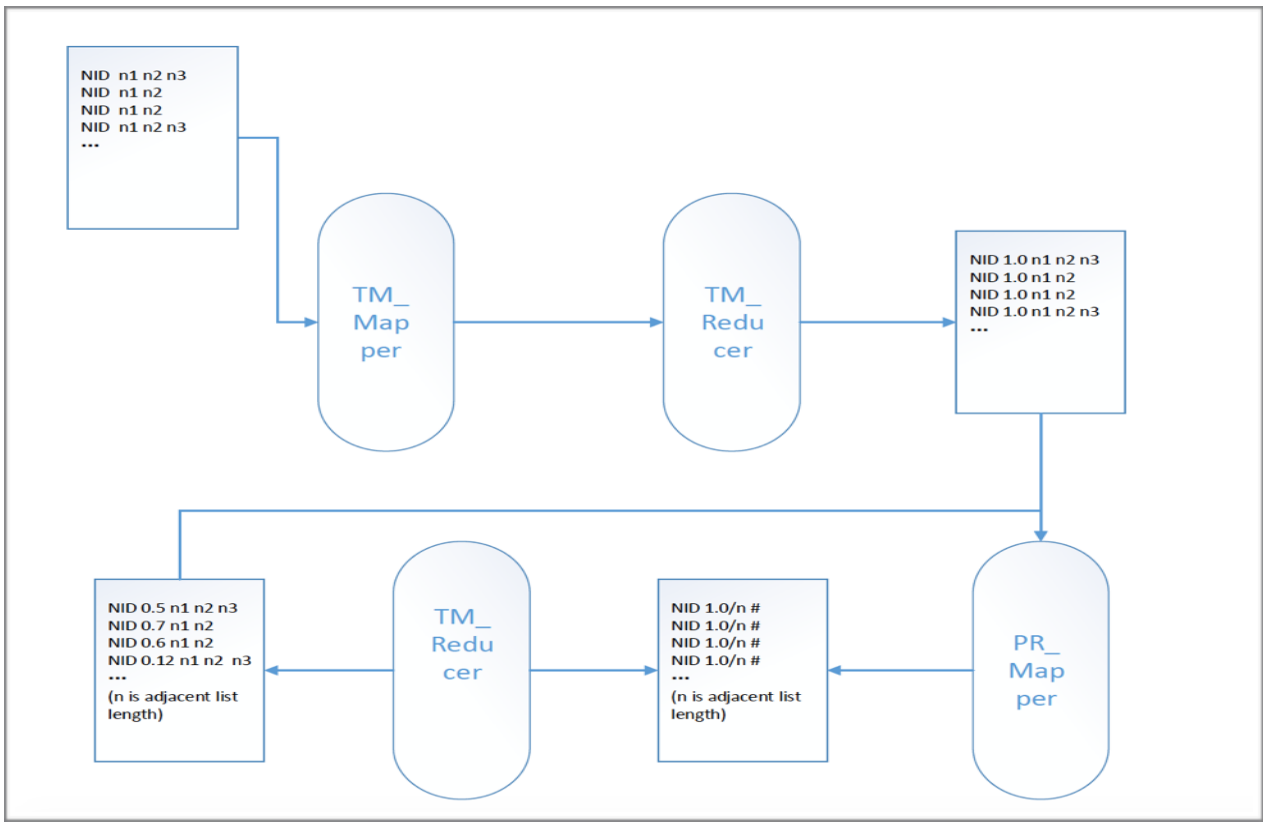
```
public static class Map extends Mapper<LongWritable, Text, Text, Text>{
    private Text nodeID = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String line = value.toString();
        StringTokenizer st = new StringTokenizer(line);
        nodeID.set(st.nextToken());
        float rank = new Float(st.nextToken());
        int numOfNeighbours = st.countTokens();

        float rankEach = 0.00f;
        if (numOfNeighbours == 0)
            rankEach = rank;
        else
            rankEach = rank / numOfNeighbours;
        String emitStr = Float.toString(rank)+" ";
        while (st.hasMoreTokens()) {
            Text next = new Text(st.nextToken());
            emitStr += next.toString();
            emitStr += " ";
            // add "#" symbol to let reducer calculate
            //<NID, rank#>
            context.write(next, new Text(Float.toString(rankEach)+"#"));
        }
        //keep matrix and old rank
        //<NID oldrank n1, n2, n3...>
        context.write(nodeID, new Text(emitStr));

    }
}
```

Picture 3

```java
public static class Reduce extends Reducer<Text, Text, Text, Text>{
    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        float sumRank = 0.00f;
        String neighbours = "";

        Configuration conf = context.getConfiguration();
        String damp = conf.get("damp");
        Float df = Float.parseFloat(damp);

        float oldRank = 0.00f;
        for (Text value : values) {
            if (value.toString().endsWith("#")) {
                StringTokenizer st = new StringTokenizer(value.toString());
                sumRank += Float.parseFloat(st.nextToken("#"));
            }
            else {
                String str = value.toString();
                String[] ss = str.split(" ", 2);
                oldRank= Float.parseFloat(ss[0]);
                neighbours = ss[1];
            }
        }

        float newRank = sumRank * df + (1 - df);
        if ((Math.abs(newRank - oldRank) > 0.001))
            context.getCounter(PAGERANKCOUNTER.Count).increment(1);
        //write into new temporary file
        //<NID newRank n1, n2, n3...>
        context.write(key, new Text(Float.toString(newRank) + " " + neighbours));
    }
}
```

Picture 4

The whole process of my program is represented by following chart. It is an iterative process.



Part II

In this part, I show results of three different graphs running in AWS EC2 instance. In small size network, the round of convergence is about 18 times. *Picture 5* shows result. And after calculation, *Picture 6* shows top 10 rank values and related nodes.

```
ubuntu@Master:~/hadoop-2.7.1$ bin/hadoop fs -cat /outputtop/part-r-00000
1045242120554427576659079652435600451O1  1.2415977
8230615676619458762969035008396747339 4  0.8939416
1346619962341598084883751700704731875 82 0.88566643
1553466171085608085811841426293297292 30 0.87416553
2271094487023021135079036047599184160 63 0.83387303
1686732524695505795578990678364209325 46 0.781567
5797937061574192860928300503432522008 8  0.73122406
1764959878348252574507371016761860610 7  0.6769259
9766802053812480883806535626787288787 1  0.591439
1337729988618102804635546978032468936 67 0.5134593
```

Picture 5

```
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:39 /input
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output1
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output10
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output11
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output12
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output13
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output14
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output15
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output16
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output17
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output18
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output2
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output3
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output4
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output5
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output6
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output7
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output8
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /output9
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:40 /outputtop
```

Picture 6

In medium size network, the round of convergence is about 13 times. *Picture 7* shows result. And after calculation, *Picture 8* shows top 10 rank values and related nodes.

```
ubuntu@Master:~/hadoop-2.7.1$ bin/hadoop fs -cat /outputtop/part-r-00000
111981443422667599916101641267414970874 1.2809062
304426760625152844155987234180143550 61  0.70466423
217182398344717121985059912345853998316 0.5931263
104105844697470013276372331783894076726 0.5148459
643632821489458762108903368728657553 43  0.5061476
255141271871887572604204954207769279563 0.46991488
298690743135077500802007851608046438995 0.44697845
116480772629362012002460626777081605400 0.4382173
303806832053566290572095716352649981643 0.4250967
148511838361064104411653673322648403910 0.41369197
```

Picture 7

```
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:33 /input
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output1
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output10
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output11
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output12
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output13
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output2
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output3
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output4
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output5
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output6
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output7
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output8
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /output9
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:35 /outputtop
```

Picture 8

In large size network, the round of convergence is about 13 times. *Picture 9* shows result. And after calculation, *Picture 10* shows top 10 rank values and related nodes.

```
ubuntu@Master:~/hadoop-2.7.1$ bin/hadoop fs -cat /outputtop/part-r-00000
64032941963750223601505696787123138445  5.5752816
11933741243794013314488192320804988244 2 5.1584167
29441828984030197332267216930039492418 4 3.7825537
28451023925191004642759305748644918508 5 3.4762263
97008356640547621048472268562410523068  3.4050694
17129678229587134900557165283241780 36   3.3882518
15647161731364441982668626578918440229 9 3.3090153
21491768659455923649754762253345725816 6 3.2748156
25947979394195914996030993368286605997 5 2.760459
41465870706060606689699857814850788091  2.7112608
```

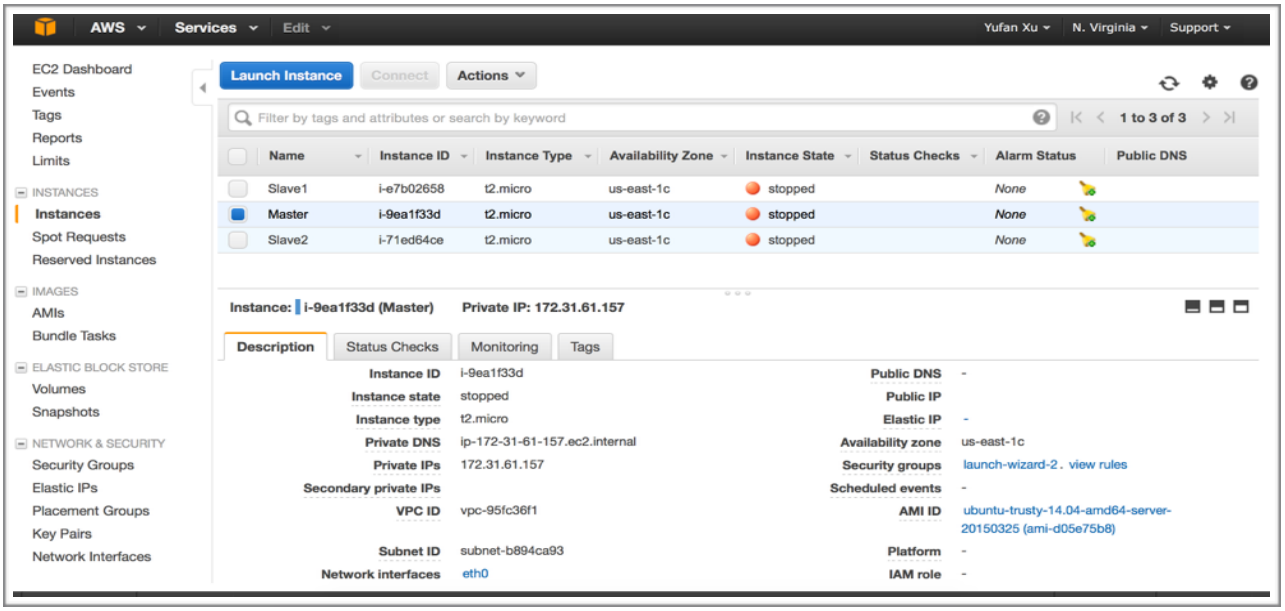
Picture 9

```
ubuntu@Master:~/hadoop-2.7.1$ bin/hadoop fs -ls /
Found 29 items
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:27 /input
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:27 /output
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output1
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output10
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output11
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output12
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output13
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output14
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output15
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output16
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output17
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output18
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output19
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output2
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output20
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output21
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output22
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output23
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output24
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output25
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output26
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output3
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output4
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output5
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output6
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output7
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output8
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /output9
drwxr-xr-x   - ubuntu supergroup          0 2015-10-15 20:28 /outputtop
```

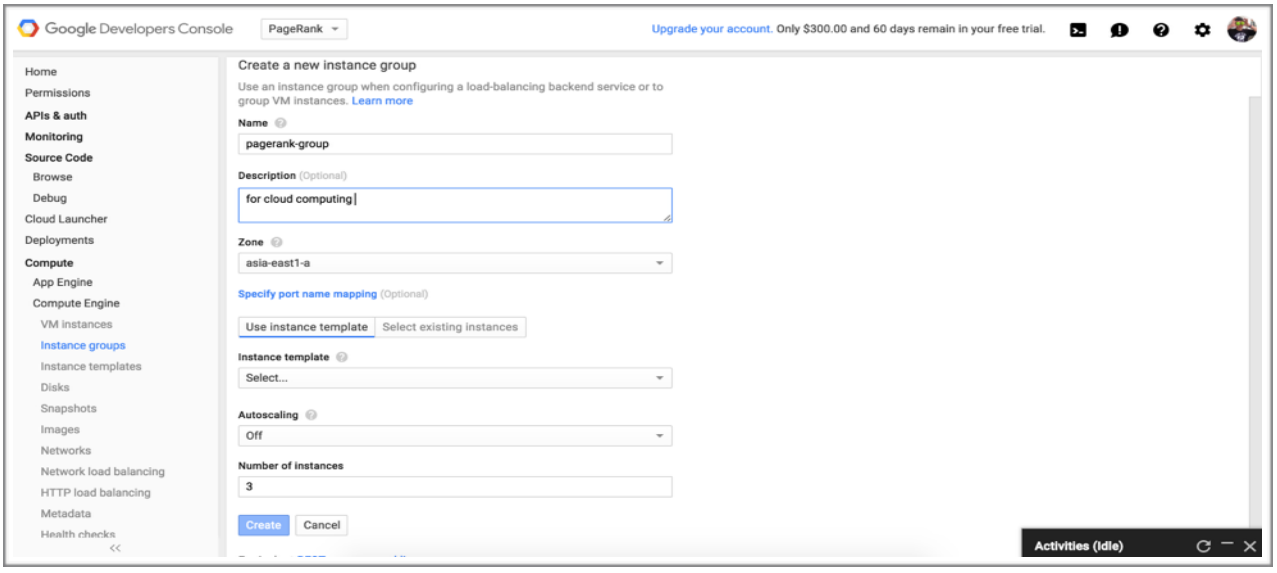
Picture 10

Part III

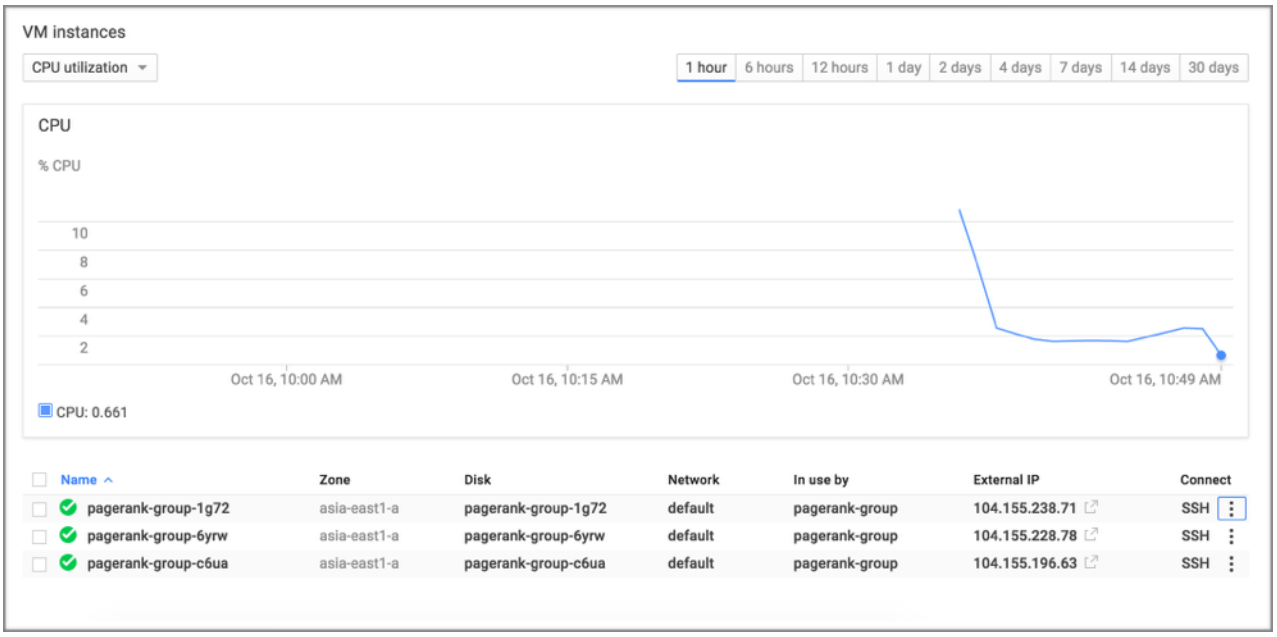
In this part, I run the same program in two different platform to compare the performance of different platform. First, I created three instances in AWS and google cloud platform. *Picture 11,12 and 13* show the creation and configuration progress.



Picture 11



Picture 12



Picture 13

So after running the same program in two platform, *Table1* shows running time of calculating rank value and *Table 2* show running time of sorting top 10 rank value nodes. So, I found there is no big difference in these two platform.

	Small Size	Medium Size	Large Size
AWS EC2	19618	14077	29557
Google Cloud Platform	18930	15900	31456

Table 1

	Small Size	Medium Size	Large Size
AWS EC2	1090	1133	1082
Google Cloud Platform	1123	1022	1193

Table 2

Part IV

After this project, I understand the PageRank algorithm and know how to implement algorithm in Hadoop framework. PageRank algorithm is an important algorithm to provide high quality searching result for common users.