**Algorithm Description**

1. preprocessing

In order to calculate the page rank value of a certain iteration, the page rank value of the previous iteration and the node linking relationships have to be preserved.

Therefore, we transform the input file. Each line of the input file has the format:

n1 n2 n4 0.2 0

where n1, n2, n4 are node names. 0.2 is the current page rank value, and 0 is the previous one. In this way, convergence can be detected as well by subtracting the two value.

After the first round, this line will look like

n1 n2 n4 0.066 0.2

This corresponds to figure 1.

During this initial step, many basic information can be acquired, such as node number, edge number,  (min, max, avg) of out-degree for each node

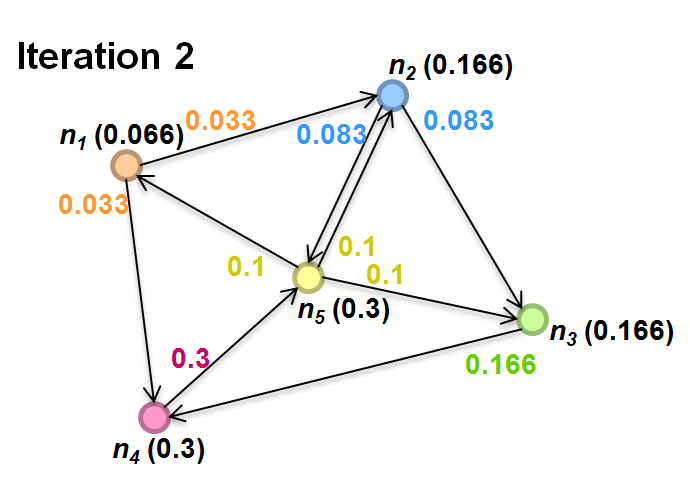
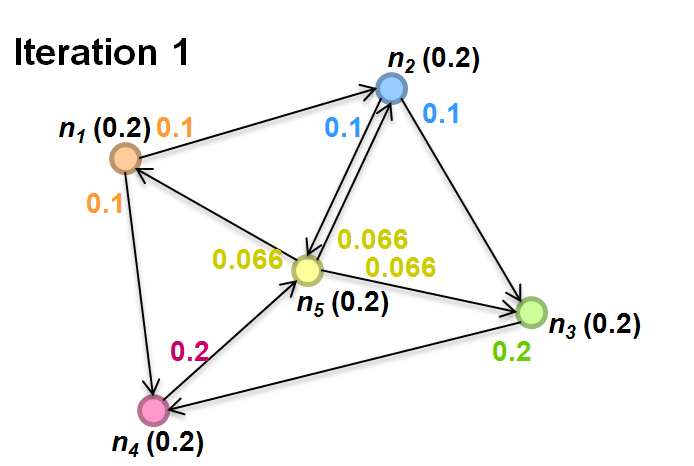


Figure 1.

2. Main()

In the main function, for every iteration, we set up the JobConf and let it run. The start time of the process is recorded and the iteration time is counted.

3. Mapper();

In the mapper function, each line in the previous file is read in and parsed, to generate pairs of 4 forms:

{Converge: previouPR-currentPR<limit?true:false}

{Node: neighbor list }

{Node: currentPR}

{Neighbor node: currentPR/neighborNum}

Since the 2nd, 3rd, 4th pairs have the same kind of key values (node names), symbols like "~" and "\*" are added to value strings, so reducer can detect their type. This means these symbols are forbidden in the input file.

4. Reducer();

In reducer, {Converge: previouPR-currentPR<limit?true:false} pairs are used to determine if the results are converged. Only if all values received by a mapper are true can the program exits by calling "System.exit(0)". The program may retry this iteration depending on the hadoop configuration. The convergence limit is from command line arguments. 0.01 of the initial page rank is used in the tests.

Values from {Neighbor node: currentPR/neighborNum} are summed up, and damping factor 0.85 is used to update the current page rank value. This value is concatenated with values with the same key from {Node: neighbor list }, {Node: currentPR}. Together they are written to a output file, which can be used as input for the next iteration.

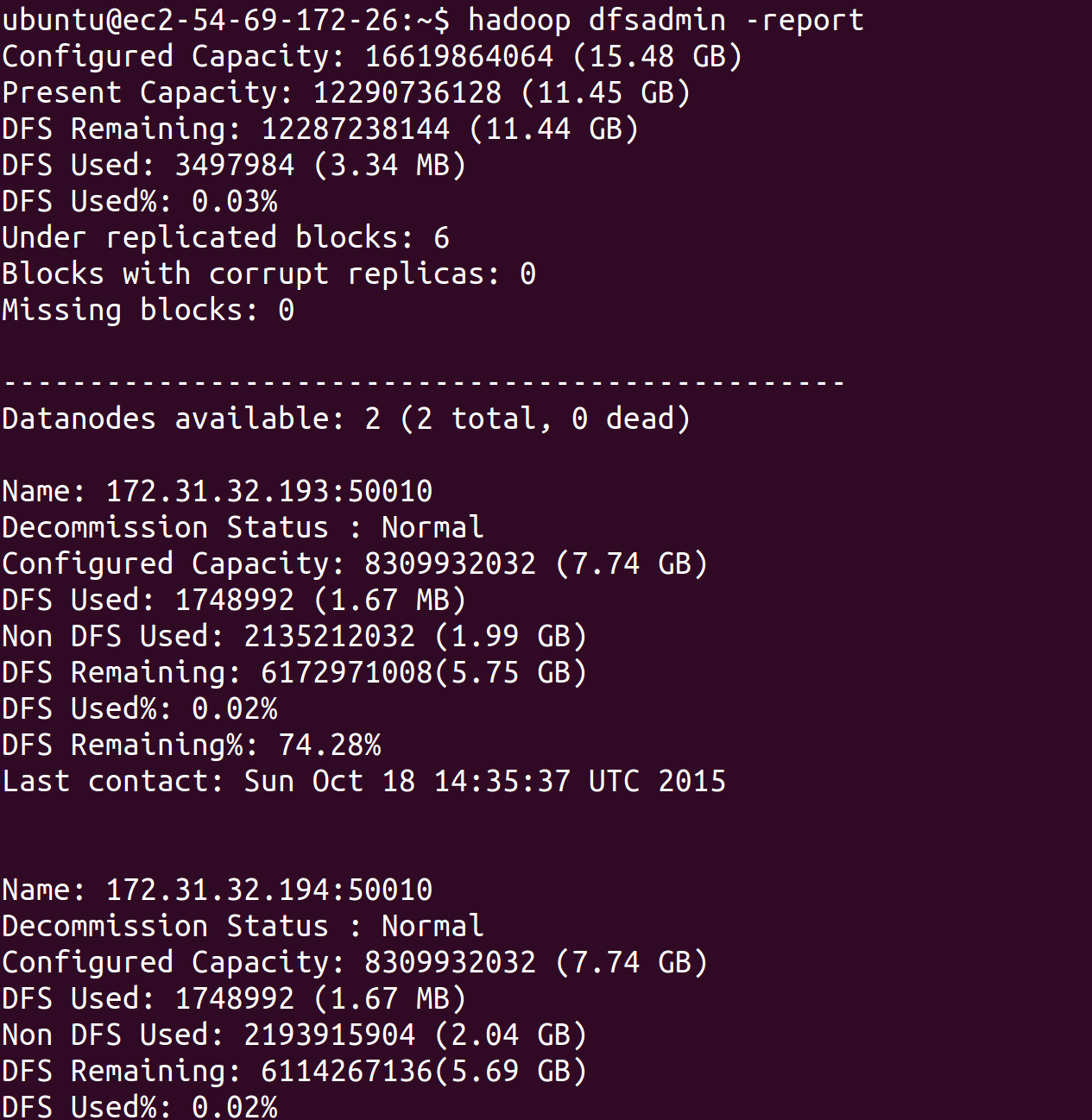
**Flow Chart**

The flow chart of the above design is shown in figure 2.

Figure 2.

**Test on AWS**

3 micro.t2 EC2s are used, with one of them as master node, and two as slave nodes.

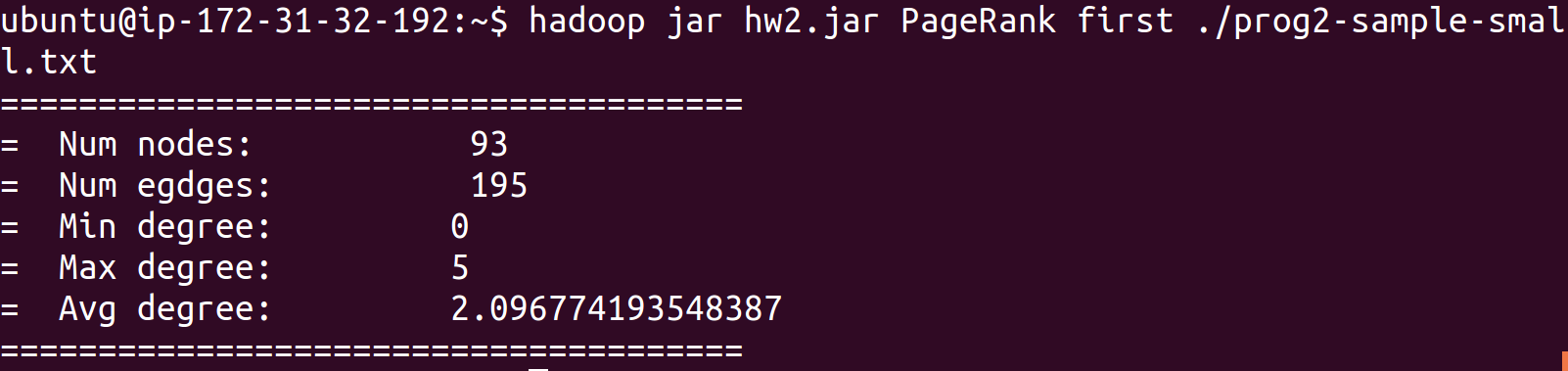


1. Test 1 - prog2-sample-small

a. Run command with keyword "first" to generate the basic information about the input file, and produce a input file for map reduce. You have to first create a directory called "in" in both hdfs and local directory：

mkdir in

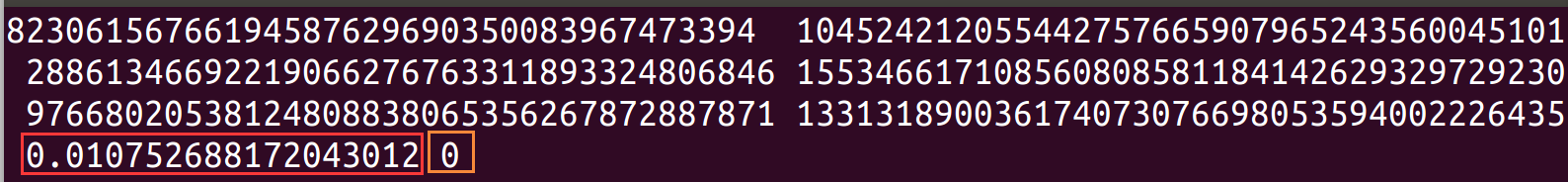
hadoop fs -mkdir in



b. Copy generated file (in/output0) to directory "in" in HDFS:

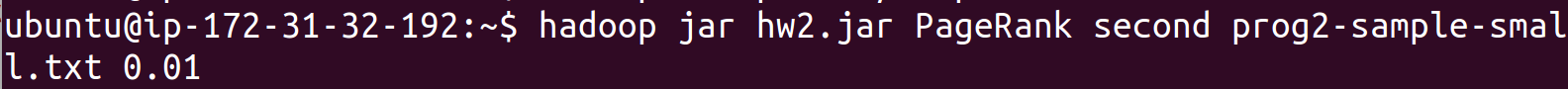
hadoop fs -put in/output0 in

The first line in output0 is as below:

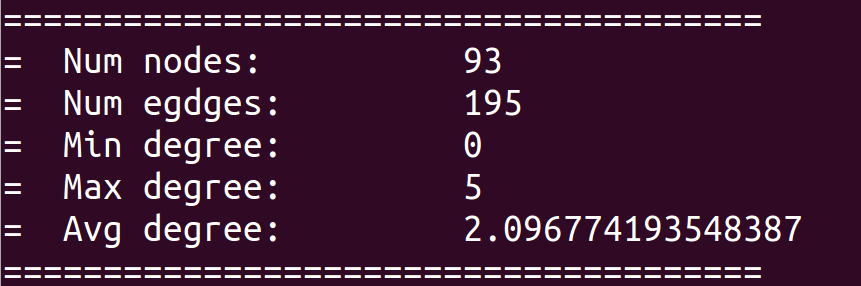


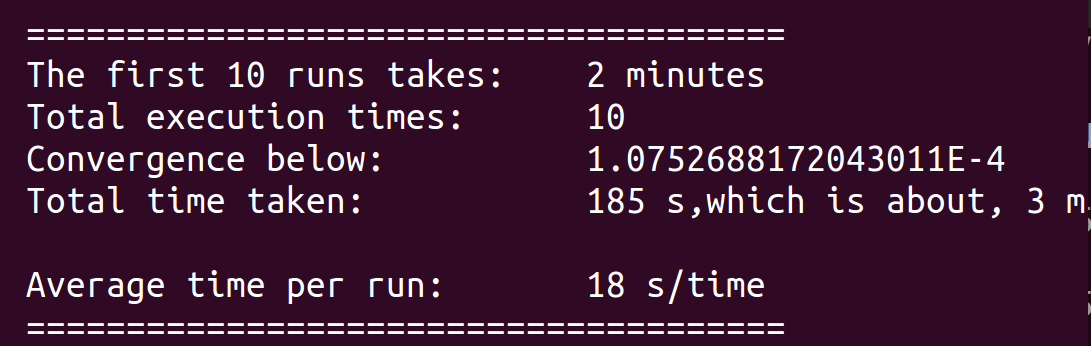
The last number (in orange) is the previous page rank value, and the second last number is the current page rank value. This is used to calculate convergence.

c. Run command to run the map reduce task. "second" suggest this is the second time to run this program, and 0.01 is the convergence accuracy. In this case, the convergence limit is 0.01075\*0.01.



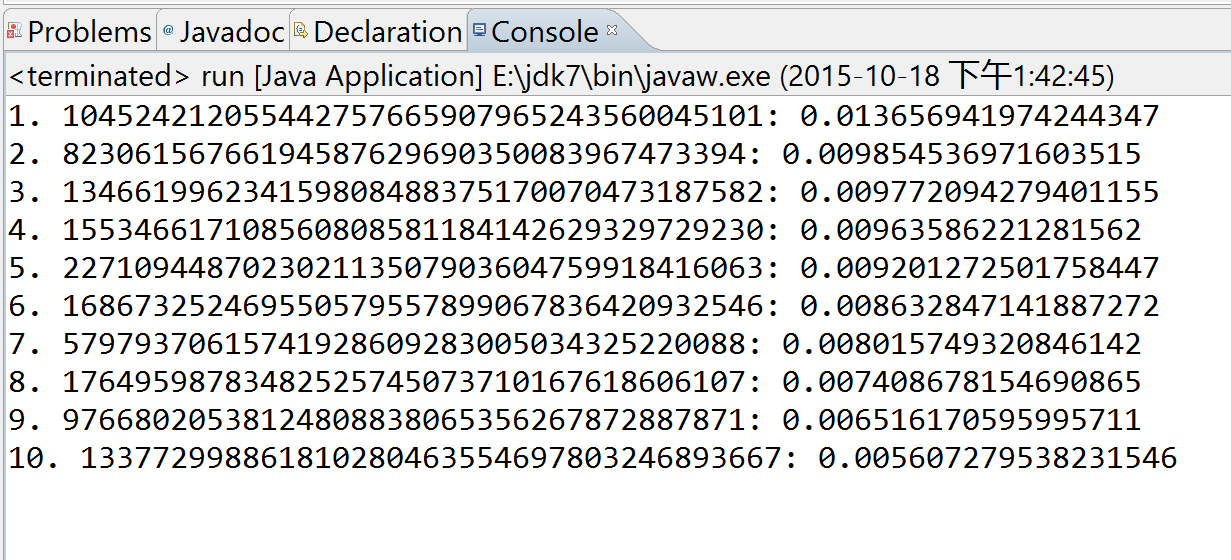
And you can see the results at the end.





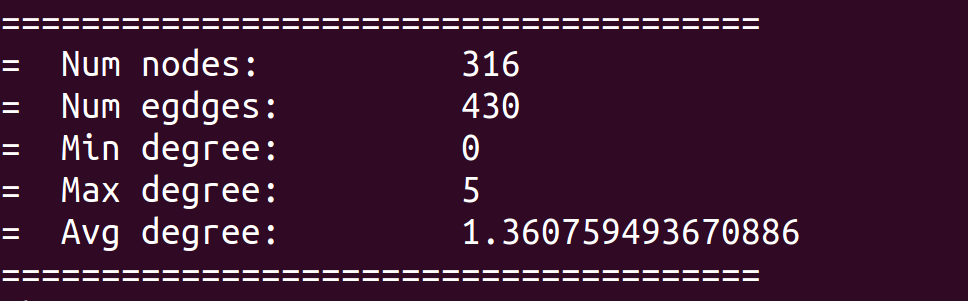
d. Get top 10 value

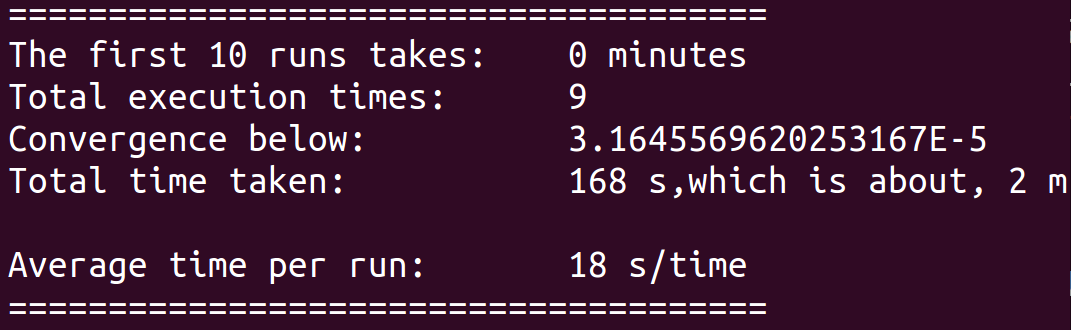
Find the output file of the last iteration, and run it with java sort program in Eclipse.



2. Test 1 - prog2-sample-medium

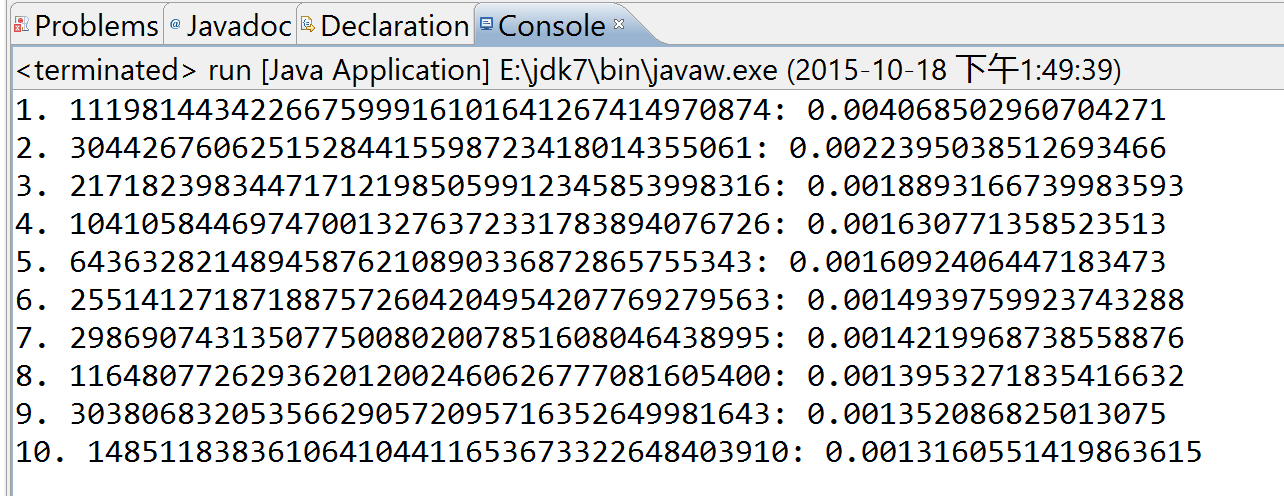
The steps are the same as above. The result is:



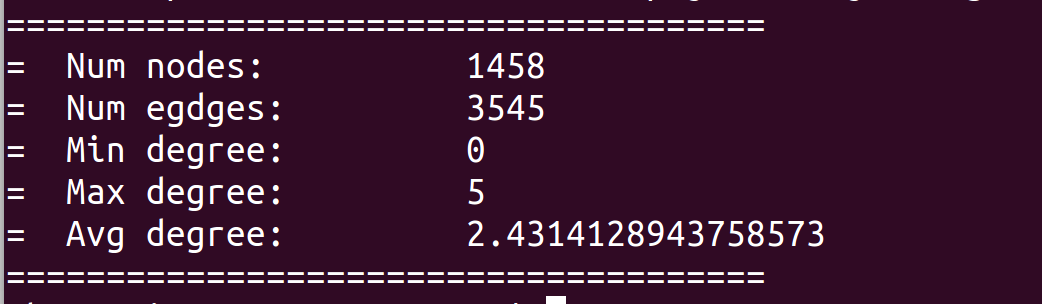


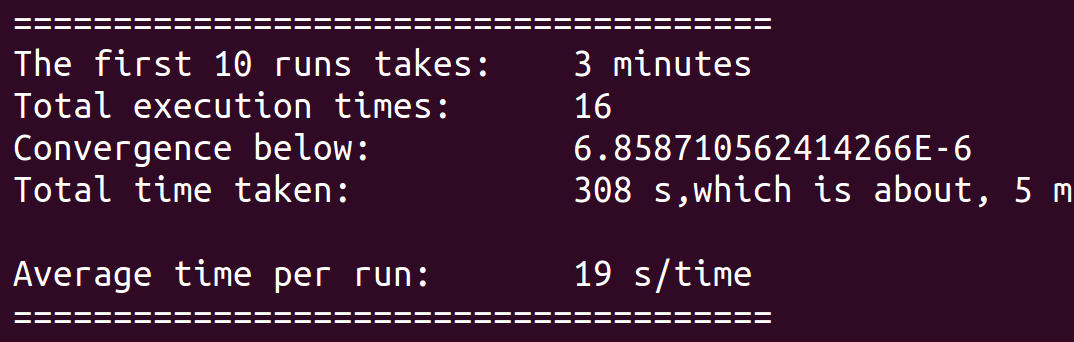
Note that the first 10 runs is 0 minutes, because it only takes 9 times to converge.

4. The top 10 nodes are:

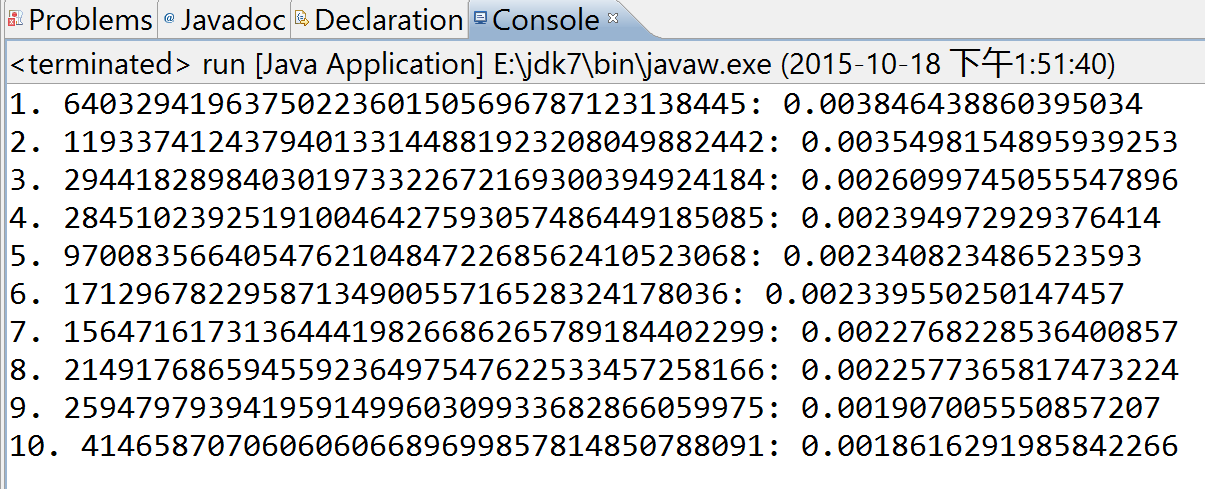


3. Test 1 - prog2-sample-large





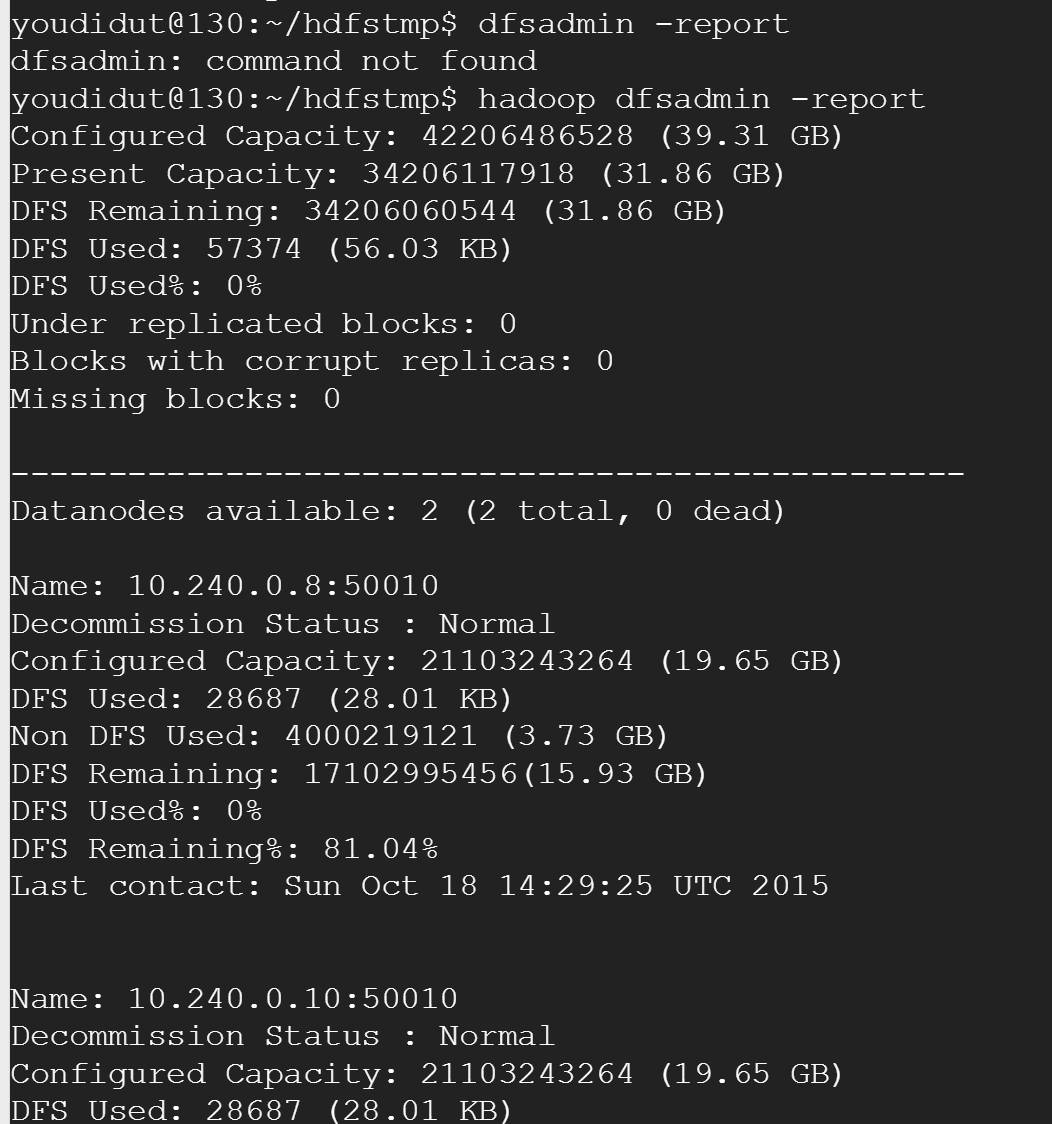
4. The top 10 nodes are:



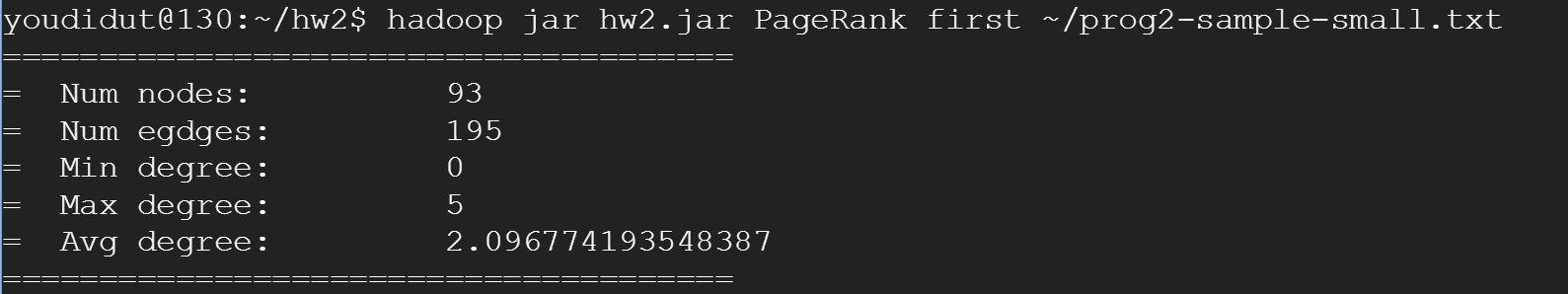
**Test on Google Cloud**

(Steps are the same as AWS. Only screenshots are presented.)

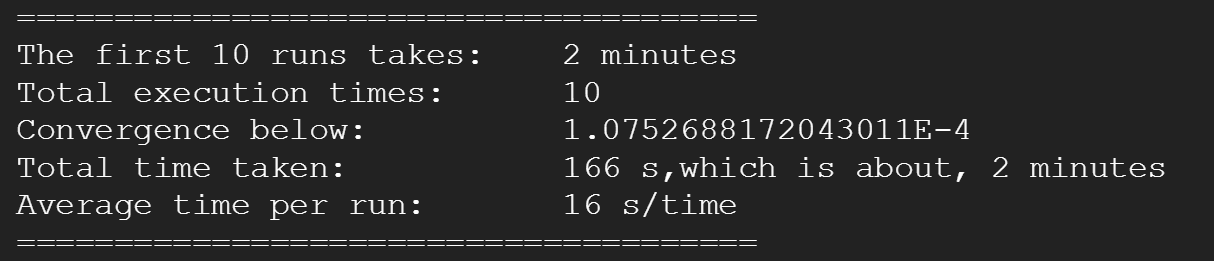
3 n1-standard-2 (2 vCPUs, 7.5 GB memory) instances are used, with one as master, and two as slaves.

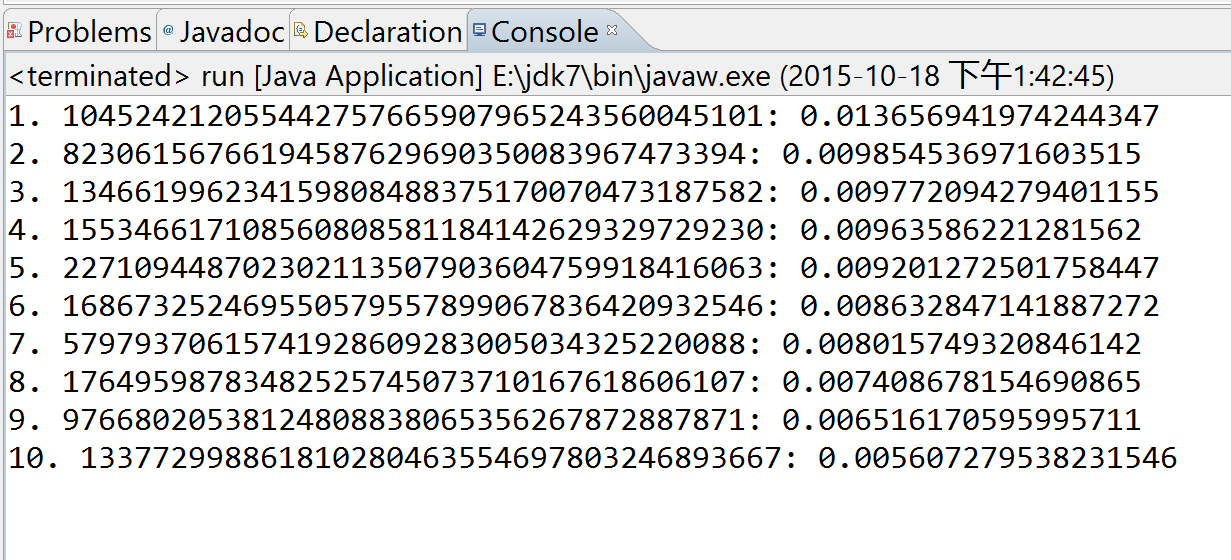


1. Test 1 - prog2-sample-small

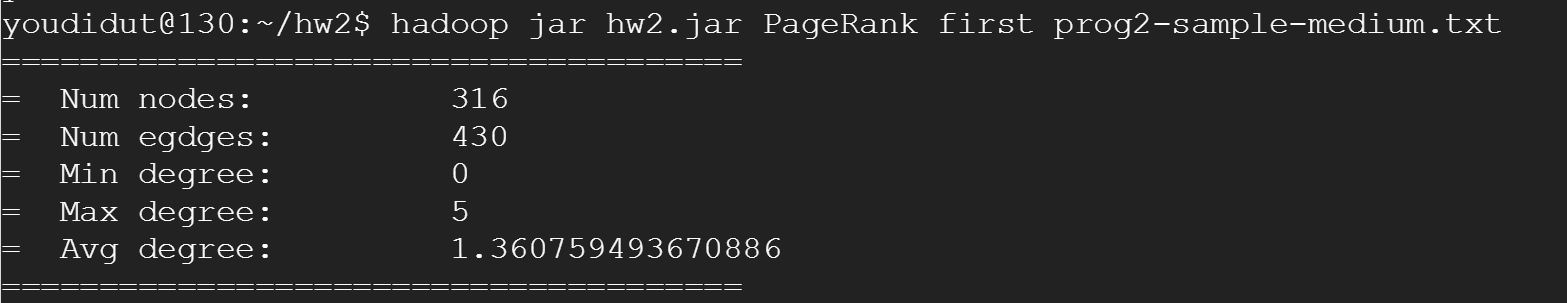




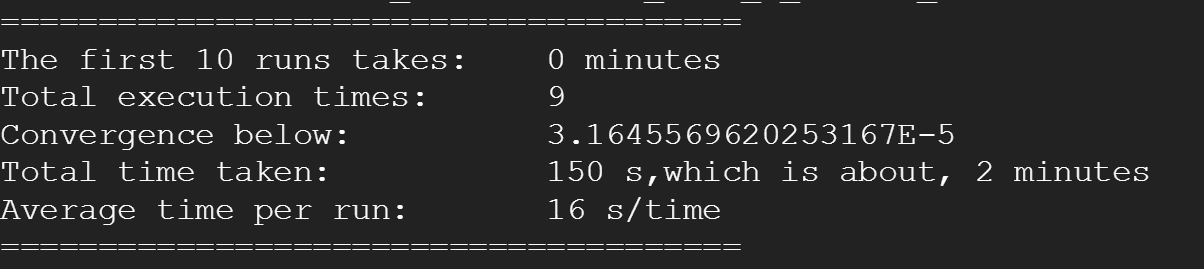


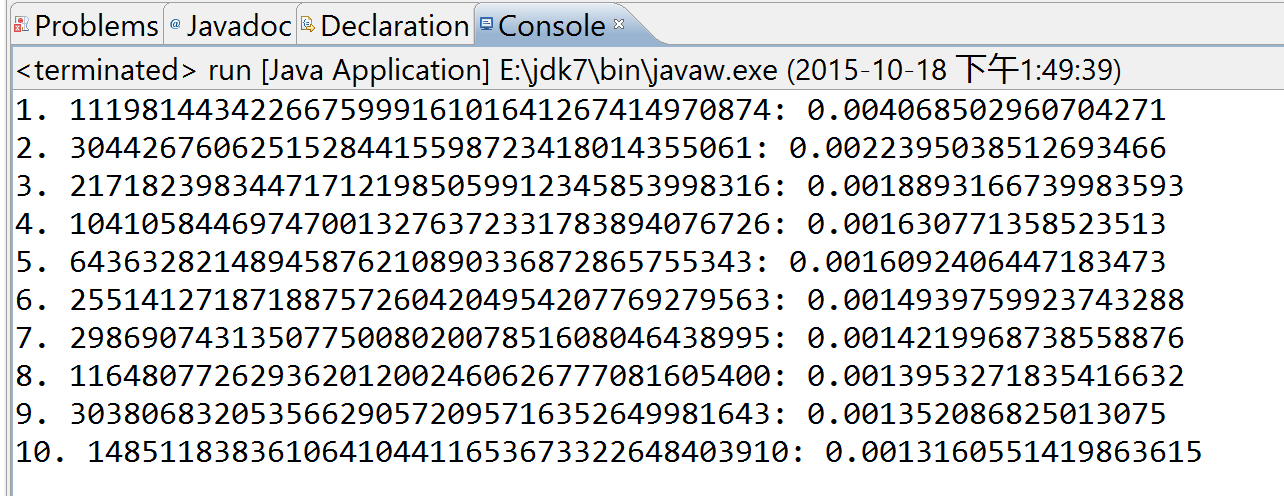


2. Test 2- prog2-sample-medium

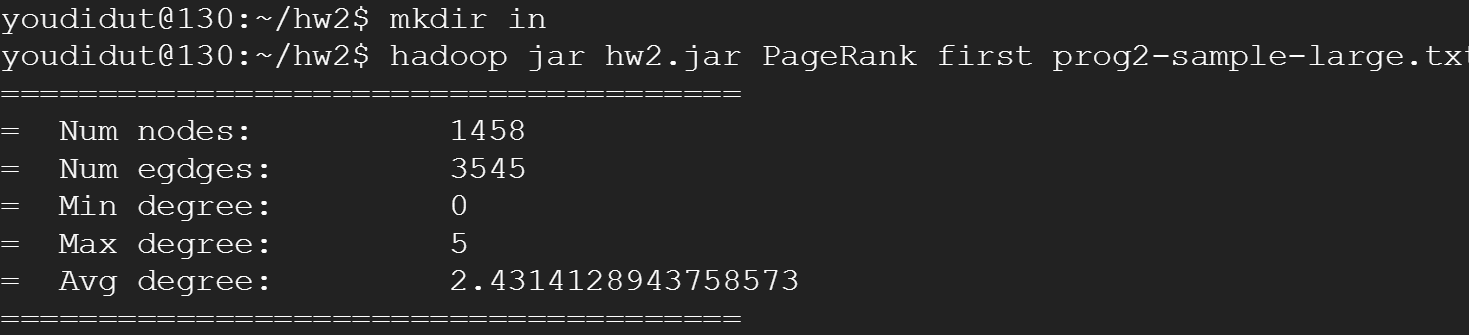




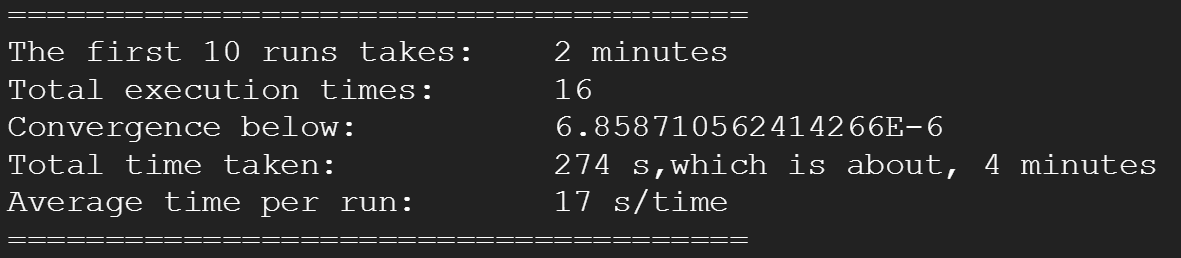


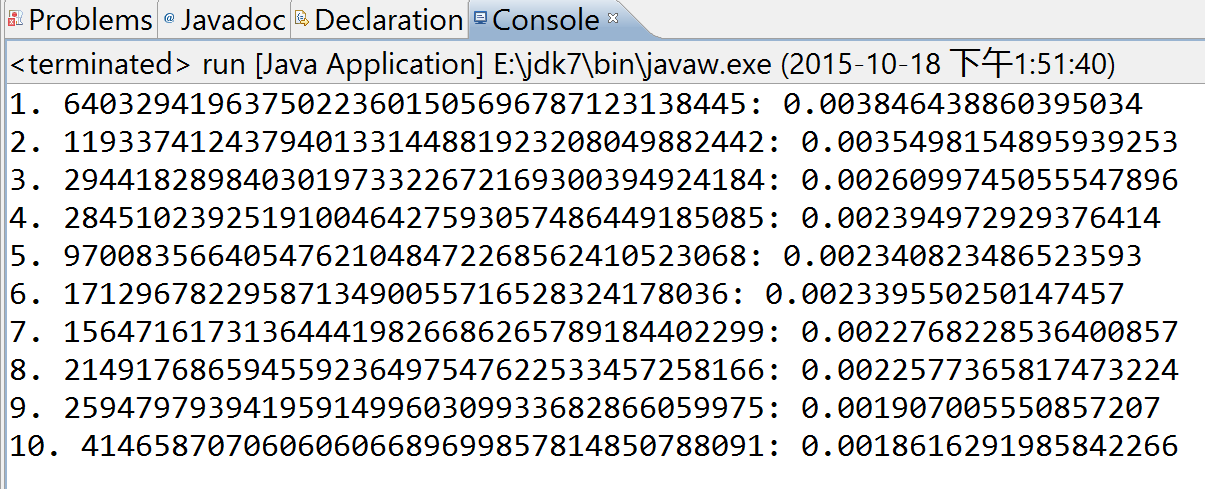


3. Test 2- prog2-sample-large









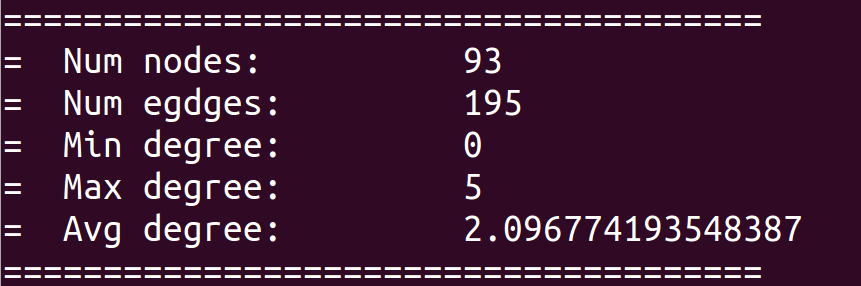
**Comparison of the AWS and Google Cloud**

|  |  |  |
| --- | --- | --- |
|  | AWS | Cloud |
| Instance Type | t2.micro, 1v CPU, 1 GB memory | n1-standard-2, 2 vCPUs, 7.5 GB memory |
| Instance Number | 1 master, 2 slaves | 1 master, 2 slaves |
| prog2-sample-small |  |  |
| First 10 run | 185s | 166s |
| Total execution time | 185s | 166s |
| Total execution times | 10 | 10 |
| Average Time per Run | 18s | 16s |
| Link to last iteration's output | <https://s3-us-west-2.amazonaws.com/hw-di/hw2-1/in/output10/part-00000> | |
| prog2-sample-medium |  |  |
| First 10 run | NA | NA |
| Total execution time | 168s | 150s |
| Total execution times | 9 | 9 |
| Average Time per Run | 18s | 16s |
| Link to last iteration's output | <https://s3-us-west-2.amazonaws.com/hw-di/hw2-3/in/output9/part-00000> | |
| prog2-sample-large |  |  |
| First 10 run | 189s | 163s |
| Total execution time | 308s | 274s |
| Total execution times | 9 | 9 |
| Average Time per Run | 19s | 17s |
| Link to last iteration's output | <https://s3-us-west-2.amazonaws.com/hw-di/hw2-2/in/output16/part-00000> | |

The alleged memory of GCEs is 7.5 times of ec2s, and the performance is slightly better.

**Summary of Graph Property**

1. Test 1 - prog2-sample-small

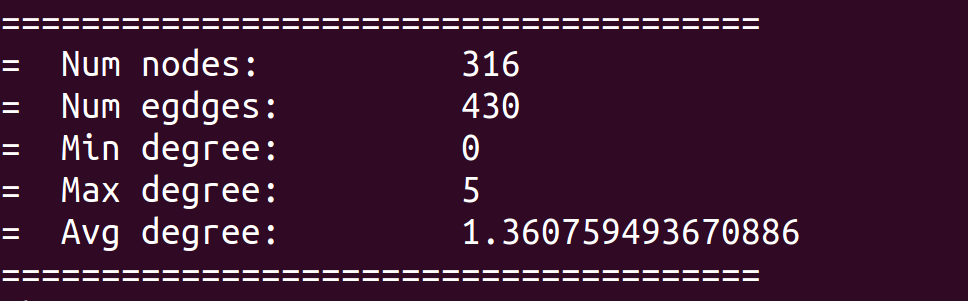


Total number of iterations to converge: 10.

The converge difference is 0.01 of the initial value.

2. Test 1 - prog2-sample-medium

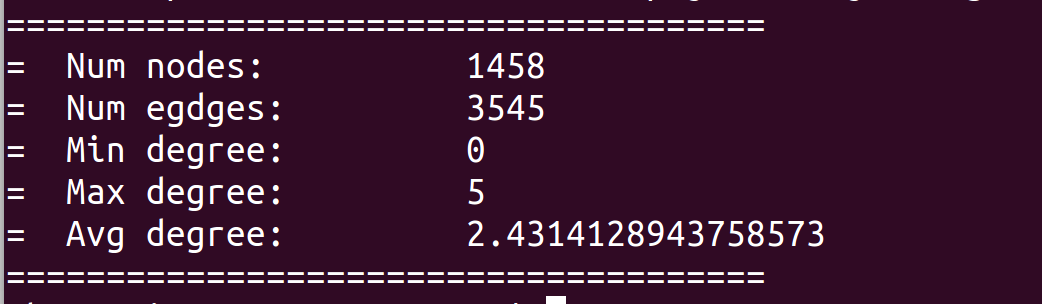
The steps are the same as above. The result is:



Total number of iterations to converge: 9

The converge difference is 0.01 of the initial value.

3. Test 1 - prog2-sample-large



Total number of iterations to converge: 16

The converge difference is 0.01 of the initial value.