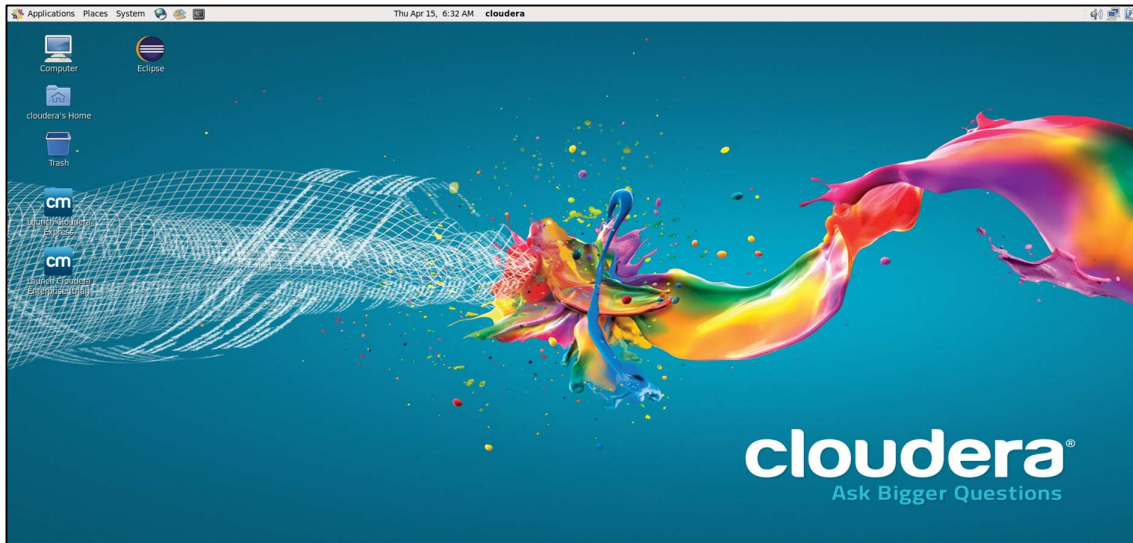


## MapReduce for PageRank algorithm on Hadoop:

Let's implement the PageRank algorithm on an input web graph and find the page rank of all the web pages.

### Step 1: Open Cloudera Quickstart VM.

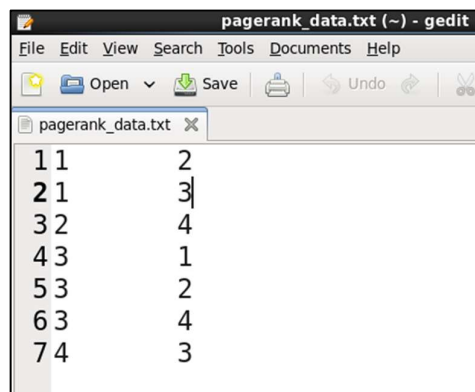


### Step 2: Clone the following repository on your local machine.

[www.github.com/NSTiwari/Hadoop-MapReduce-Programs](https://www.github.com/NSTiwari/Hadoop-MapReduce-Programs)

You'll see the **PageRank** folder inside this repository along with other folders. The **PageRank** folder contains the following files –

- **pagerank\_data.txt** – A placeholder text file for input web graph.
- **pagerank\_mapper1.py** – First mapper file for PageRank algorithm.
- **pagerank\_reducer1.py** – First reducer file for PageRank algorithm.
- **pagerank\_mapper2.py** – Second mapper file for PageRank algorithm.
- **pagerank\_reducer2.py** – Second reducer file for PageRank algorithm.



In the **pagerank\_data.txt** file, the first column represents the 'from-nodes' and the second column represents the 'to-nodes'; both separated by tab.

Thus, there are total seven edges in the input web graph.

### **pagerank\_mapper1.py:**

```
#!/usr/bin/python
```

```
import sys

for line in sys.stdin:
    if line.startswith('#'):
        continue
    else:
        print("%s" % (",".join(line.strip().split())))
```

### **pagerank\_reducer1.py:**

```
#!/usr/bin/python
```

```
import sys

cur_node = None
prev_node = None
adj_list = []

for line in sys.stdin:
    cur_node, dest_node = line.strip().split(',')
    if cur_node == prev_node:
        adj_list.append(dest_node)
    else:
        if prev_node:
            print("%s\t%s" % (prev_node, ",".join(sorted(adj_list))))

            prev_node = cur_node
            del adj_list[:]
            adj_list.append(dest_node)

if cur_node == prev_node:
    print("%s\t%s" % (prev_node, ",".join(sorted(adj_list))))
```

### **pagerank\_mapper2.py:**

```
#!/usr/bin/python
```

```
import sys

pageranks = {'1': '1', '2': '1', '3': '1', '4': '1'}

for line in sys.stdin:
    node, adj_list = line.strip().split('\t')
    adj_list = adj_list.split(',')
    out_num = len(adj_list)

    print("%s,%s" % (node, '0.0'))
    #print("Adjacency list:", adj_list)

    for out_link in adj_list:
        out_link_contrib = float(pageranks[node]) / out_num
        print("%s,%s" % (out_link, out_link_contrib))
```

## pagerank\_reducer2.py:

```
#!/usr/bin/python
```

```
import sys
```

```
cur_node = None
```

```
prev_node = None
```

```
contrib_sum = 0
```

```
damping_factor = 0.85
```

```
for line in sys.stdin:
```

```
    cur_node, contrib = line.strip().split(',')
```

```
    print("\nCurrent node: %s\nContribution by this node: %s" %  
        (cur_node, contrib))
```

```
if cur_node == prev_node:
```

```
    contrib_sum += float(contrib)
```

```
else:
```

```
    if prev_node:
```

```
        new_pr = (1 - damping_factor) + (damping_factor *  
        contrib_sum)
```

```
        new_pr = round(new_pr, 5)
```

```
        print("Previous node: %s\nNew page rank of node %s: %s" %  
            (prev_node, cur_node, new_pr))
```

```
    prev_node = cur_node
```

```
    contrib_sum = float(contrib)
```

```
if cur_node == prev_node:
```

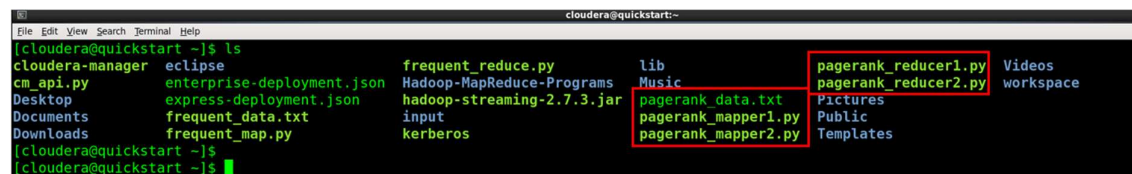
```
    new_pr = (1 - damping_factor) + (damping_factor * contrib_sum)
```

```
    new_pr = round(new_pr, 5)
```

```
    print("Previous node: %s\nNew page rank of node %s: %s" %  
        (prev_node, cur_node, new_pr))
```

Copy these five files inside **/home/cloudera** directory. Once done, check if they are copied at the desired location properly.

ls



```
cloudera@quickstart:~$ ls
cm_api.py      eclipse        frequent_reduce.py  lib           pagerank_reducer1.py  Videos
enterprise-deployment.json  express-deployment.json  frequent_data.txt   Hadoop-MapReduce-Programs  Music                workspace
frequent_map.py  hadoop-streaming-2.7.3.jar  input              kerberos      pagerank_data.txt     Pictures
pagerank_mapper1.py  pagerank_mapper2.py      pagerank_reducer2.py  Templates
```

All the five required files are present.

### Step 3: Test MapReduce programs locally.

Before we run MapReduce program on Hadoop, let's test it locally and see if the results obtained are correct as expected.

To do so, open the terminal and type the following commands one-by-one:

Run the first mapper on the input web graph data.

```
cat pagerank_data.txt | python pagerank_mapper1.py
```

```
[cloudera@quickstart ~]$ cat pagerank_data.txt | python pagerank_mapper1.py
1,2
1,3
2,4
3,1
3,2
3,4
4,3
[cloudera@quickstart ~]$
```

This will reformat the input data by separating the ‘from-nodes’ and ‘to-nodes’ with a comma (originally separated by a tab).

Run the first reducer.

```
cat pagerank_data.txt | python pagerank_mapper1.py | python pagerank_reducer1.py
```

```
[cloudera@quickstart ~]$ cat pagerank_data.txt | python pagerank_mapper1.py | python pagerank_reducer1.py
1      2,3
2      4
3      1,2,4
4      3
[cloudera@quickstart ~]$
```

The first reducer file aggregates the ‘to-nodes’ for every ‘from-node’ respectively as shown in the above figure.

As you can see,  $1 \rightarrow 2$  and  $1 \rightarrow 3$  are reduced to  $1 \rightarrow (2, 3)$ . Similarly, all the ‘from-nodes’ are reduced as shown in the figure above.

Now, run the second mapper.

```
cat pagerank_data.txt | python pagerank_mapper1.py | python pagerank_reducer1.py | python pagerank_mapper2.py
```

```
[cloudera@quickstart ~]$ cat pagerank_data.txt | python pagerank_mapper1.py | python pagerank_reducer1.py | python pagerank_mapper2.py
1,0,0 → Contribution of self-edge for node1 i.e. (1 → 1) = 0 (since self-edge for node1 doesn't exist in the input web graph)
2,0,5 → Contribution of edge (1 → 2) = 0.5
3,0,5 → Contribution of edge (1 → 3) = 0.5
2,0,0 → Contribution of self-edge for node2 i.e. (2 → 2) = 0 (since self-edge for node2 doesn't exist in the input web graph)
4,1,0 → Contribution of edge (2 → 4) = 1
3,0,0 → Contribution of self-edge for node3 i.e. (3 → 3) = 0 (since self-edge for node3 doesn't exist in the input web graph)
1,0.3333333333333333 → Contribution of edge (3 → 1) = 0.333
2,0.3333333333333333 → Contribution of edge (3 → 2) = 0.333
4,0.3333333333333333 → Contribution of edge (3 → 4) = 0.333
4,0,0 → Contribution of self-edge for node4 i.e. (4 → 4) = 0 (since self-edge for node4 doesn't exist in the input web graph)
3,1,0 → Contribution of edge (4 → 3) = 1
```

The interpretation of the output obtained after applying the second mapper is commented in the image above.

Now finally, run the second reducer.

```
cat pagerank_data.txt | python pagerank_mapper1.py | python pagerank_reducer1.py | python pagerank_mapper2.py | python pagerank_reducer2.py
```

```

[cloudera@quickstart ~]$ cat pagerank_data.txt | python pagerank_mapper1.py | python pagerank_reducer1.py | python pagerank_mapper2.py | python pagerank_reducer2.py

Current node: 1
Contribution by this node: 0.0

Current node: 2
Contribution by this node: 0.5
Previous node: 1
New page rank of node 2: 0.15

Current node: 3
Contribution by this node: 0.5
Previous node: 2
New page rank of node 3: 0.575

Current node: 2
Contribution by this node: 0.0
Previous node: 3
New page rank of node 2: 0.575

Current node: 4
Contribution by this node: 1.0
Previous node: 2
New page rank of node 4: 0.15

Current node: 3
Contribution by this node: 0.0
Previous node: 4
New page rank of node 3: 1.0

Current node: 1
Contribution by this node: 0.333333333333
Previous node: 3
New page rank of node 1: 0.15 → Final PR(1) = 0.15

Current node: 2
Contribution by this node: 0.333333333333
Previous node: 1
New page rank of node 2: 0.43333 → Final PR(2) = 0.4333

Current node: 4
Contribution by this node: 0.333333333333
Previous node: 2
New page rank of node 4: 0.43333 → Final PR(4) = 0.4333

Current node: 4
Contribution by this node: 0.0

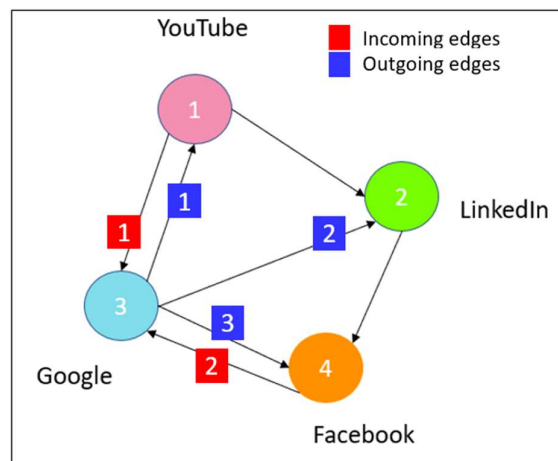
Current node: 3
Contribution by this node: 1.0
Previous node: 4
New page rank of node 3: 0.43333
Previous node: 3
New page rank of node 3: 1.0 → Final PR(3) = 1

[cloudera@quickstart ~]$

```

So, the final page ranks for node1, node2, node3 and node4 are 0.15, 0.4333, 1 and 0.4333 respectively. From this, it can be deduced that node1 is of least importance and node3 is the most important page.

This can also be supported by comparing the page rank results obtained, with the input web graph.



As you can see, there are 2 incoming edges and 3 outgoing edges to and from node3 respectively. Higher the no. of edges (incoming or outgoing) associated with a node, higher is its importance.

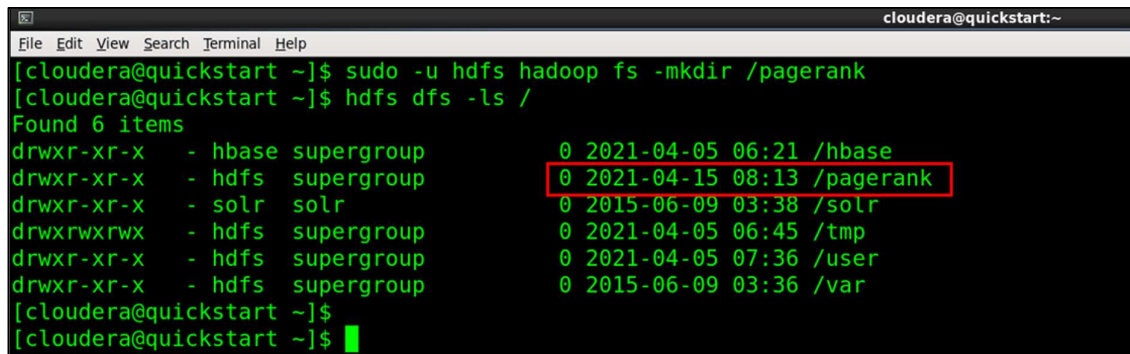
Hence, the order of importance of all the nodes is:

1. Google
2. LinkedIn
3. Facebook
4. YouTube

Now that we know the results of the MapReduce programs we executed are as expected, we are good to go for the Hadoop implementation of the same.

#### Step 4: Create a directory on HDFS.

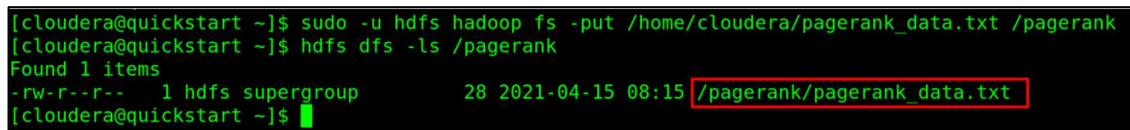
```
sudo -u hdfs hadoop fs -mkdir /pagerank
hdfs dfs -ls /
```

A terminal window titled 'cloudera@quickstart:~' showing the execution of 'hdfs dfs -ls /'. The output lists several directories: /hbase, /pagerank, /solr, /tmp, /user, and /var. The entry for /pagerank is highlighted with a red box, showing its permissions as 'drwxr-xr-x', owner as 'hdfs', group as 'supergroup', and timestamp as '0 2021-04-15 08:13'.

```
cloudera@quickstart:~$ sudo -u hdfs hadoop fs -mkdir /pagerank
cloudera@quickstart:~$ hdfs dfs -ls /
Found 6 items
drwxr-xr-x  - hbase  supergroup          0 2021-04-05 06:21 /hbase
drwxr-xr-x  - hdfs   supergroup          0 2021-04-15 08:13 /pagerank
drwxr-xr-x  - solr   solr              0 2015-06-09 03:38 /solr
drwxrwxrwx  - hdfs   supergroup          0 2021-04-05 06:45 /tmp
drwxr-xr-x  - hdfs   supergroup          0 2021-04-05 07:36 /user
drwxr-xr-x  - hdfs   supergroup          0 2015-06-09 03:36 /var
cloudera@quickstart:~$
```

#### Step 5: Copy input file on HDFS.

```
sudo -u hdfs hadoop fs -put /home/cloudera/pagerank_data.txt /pagerank
hdfs dfs -ls /pagerank
```

A terminal window showing the execution of 'hdfs dfs -ls /pagerank'. The output shows a single file 'pagerank/pagerank\_data.txt' with permissions '-rw-r--r--', owner 'hdfs', group 'supergroup', and timestamp '28 2021-04-15 08:15'. The file name is highlighted with a red box.

```
cloudera@quickstart:~$ sudo -u hdfs hadoop fs -put /home/cloudera/pagerank_data.txt /pagerank
cloudera@quickstart:~$ hdfs dfs -ls /pagerank
Found 1 items
-rw-r--r--  1 hdfs  supergroup          28 2021-04-15 08:15 /pagerank/pagerank_data.txt
cloudera@quickstart:~$
```

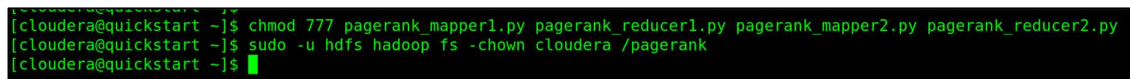
The input file is copied successfully inside **pagerank** directory.

#### Step 6: Configure permissions to run MapReduce for PageRank algorithm on Hadoop.

Now, before we execute the MapReduce jobs on Hadoop, we need to give permission to read, write and execute the MapReduce program. We also need to provide permission for the default user (cloudera) to write the output file on HDFS.

To do so, run the following commands.

```
chmod 777 pagerank_mapper1.py pagerank_reducer1.py pagerank_mapper2.py pagerank_reducer2.py
sudo -u hdfs hadoop fs -chown cloudera /pagerank
```

A terminal window showing the execution of 'chmod 777' on the four Python files and 'hdfs dfs -chown cloudera /pagerank'. The commands are executed successfully.

```
cloudera@quickstart:~$ chmod 777 pagerank_mapper1.py pagerank_reducer1.py pagerank_mapper2.py pagerank_reducer2.py
cloudera@quickstart:~$ sudo -u hdfs hadoop fs -chown cloudera /pagerank
cloudera@quickstart:~$
```

#### Step 7: Run MapReduce on Hadoop.

Execute the first MapReduce job on Hadoop.

```
hadoop jar /home/cloudera/hadoop-streaming-2.7.3.jar \
> -input /pagerank/pagerank_data.txt \
> -output /pagerank/output \
> -mapper /home/cloudera/pagerank_mapper1.py \
> -reducer /home/cloudera/pagerank_reducer1.py
```



```
cloudera@quickstart:~$ hadoop jar /home/cloudera/hadoop-streaming-2.7.3.jar \
  -input /pagerank/pagerank_data.txt \
  -output /pagerank/output \
  -mapper /home/cloudera/pagerank_mapper1.py \
  -reducer /home/cloudera/pagerank_reducer1.py
packageJobJar: [/usr/jars/hadoop-streaming-2.6.0-cdh5.4.2.jar] /tmp/streamjob6618049121681426426.jar tmpDir=null
21/04/15 08:24:00 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/10.0.2.15:8032
21/04/15 08:24:01 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/10.0.2.15:8032
21/04/15 08:24:02 INFO mapred.FileInputFormat: Total input paths to process : 1
21/04/15 08:24:02 INFO mapreduce.JobSubmitter: number of splits:2
21/04/15 08:24:02 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1618493038072_0001
21/04/15 08:24:03 INFO impl.YarnClientImpl: Submitted application application_1618493038072_0001
21/04/15 08:24:03 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1618493038072_0001/
21/04/15 08:24:03 INFO mapreduce.Job: Running job: job_1618493038072_0001
21/04/15 08:24:18 INFO mapreduce.Job: Job job_1618493038072_0001 running in uber mode : false
21/04/15 08:24:18 INFO mapreduce.Job: map 0% reduce 0%
21/04/15 08:24:36 INFO mapreduce.Job: map 100% reduce 0%
21/04/15 08:24:49 INFO mapreduce.Job: map 100% reduce 100%
21/04/15 08:24:50 INFO mapreduce.Job: Job job_1618493038072_0001 completed successfully
21/04/15 08:24:50 INFO mapreduce.Job: Counters: 49
File System Counters
  FILE: Number of bytes read=59
  FILE: Number of bytes written=347903
  FILE: Number of read operations=0
  FILE: Number of large read operations=0
  FILE: Number of write operations=0
  HDFS: Number of bytes read=262
  HDFS: Number of bytes written=22
  HDFS: Number of read operations=9
  HDFS: Number of large read operations=0
  HDFS: Number of write operations=2
Job Counters
```

← Execute MapReduce job1

← MapReduce job1 status

```
cloudera@quickstart:~$ hdfs dfs -ls /pagerank/output
Found 2 items
-rw-r--r-- 1 cloudera supergroup 0 2021-04-15 08:24 /pagerank/output/_SUCCESS
-rw-r--r-- 1 cloudera supergroup 22 2021-04-15 08:24 /pagerank/output/part-000000
```

Output directory of  
MapReduce job1

Output files of  
MapReduce job1

Let's read the output of the first MapReduce job.

`hdfs dfs -cat /pagerank/output/part-00000`

```
cloudera@quickstart:~$ hdfs dfs -cat /pagerank/output/part-00000
1      2,3
2      4
3      1,2,4
4      3
```

The output matches with that obtained during local implementation. Now, we move forward to execute the second MapReduce job. The output of the first MapReduce job becomes the input of the second MapReduce job.

Execute the second MapReduce job on Hadoop.

```
hadoop jar /home/cloudera/hadoop-streaming-2.7.3.jar \
> -input /pagerank/output/part-00000 \
> -output /pagerank/final_output \
> -mapper /home/cloudera/pagerank_mapper2.py \
> -reducer /home/cloudera/pagerank_reducer2.py
```

```
cloudera@quickstart:~$ hadoop jar /home/cloudera/hadoop-streaming-2.7.3.jar \
  -input /pagerank/output/part-00000 \
  -output /pagerank/final_output \
  -mapper /home/cloudera/pagerank_mapper2.py \
  -reducer /home/cloudera/pagerank_reducer2.py
packageJobJar: [/usr/jars/hadoop-streaming-2.6.0-cdh5.4.2.jar] /tmp/streamjob155858756575039143.jar tmpDir=null
21/04/15 08:37:23 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/10.0.2.15:8032
21/04/15 08:37:24 INFO client.RMProxy: Connecting to ResourceManager at quickstart.cloudera/10.0.2.15:8032
21/04/15 08:37:25 INFO mapred.FileInputFormat: Total input paths to process : 1
21/04/15 08:37:25 INFO mapreduce.JobSubmitter: number of splits:2
21/04/15 08:37:25 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1618493038072_0002
21/04/15 08:37:26 INFO impl.YarnClientImpl: Submitted application application_1618493038072_0002
21/04/15 08:37:26 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1618493038072_0002/
21/04/15 08:37:26 INFO mapreduce.Job: Running job: job_1618493038072_0002
21/04/15 08:37:40 INFO mapreduce.Job: Job job_1618493038072_0002 running in uber mode : false
21/04/15 08:37:40 INFO mapreduce.Job:  map 0% reduce 0%
21/04/15 08:37:57 INFO mapreduce.Job:  map 100% reduce 0%
21/04/15 08:38:08 INFO mapreduce.Job:  map 100% reduce 100%
21/04/15 08:38:10 INFO mapreduce.Job: Job job_1618493038072_0002 completed successfully
21/04/15 08:38:10 INFO mapreduce.Job: Counters: 49
  File System Counters
    FILE: Number of bytes read=94
    FILE: Number of bytes written=347990
    FILE: Number of read operations=0
    FILE: Number of large read operations=0
    FILE: Number of write operations=0
    HDFS: Number of bytes read=253
    HDFS: Number of bytes written=800
    HDFS: Number of read operations=0
    HDFS: Number of large read operations=0
    HDFS: Number of write operations=2
  Job Counters
```

← Execute MapReduce job2

← MapReduce job2 status

```
Combine output records=0
Reduce input groups=11
Reduce shuffle bytes=113
Reduce input records=11
Reduce output records=41
Spilled Records=22
Shuffled Maps =2
Failed Shuffles=0
Merged Map outputs=2
GC time elapsed (ms)=312
CPU time spent (ms)=2760
Physical memory (bytes) snapshot=378667008
Virtual memory (bytes) snapshot=2100383744
Total committed heap usage (bytes)=152174592

Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0

File Input Format Counters
  Bytes Read=33
File Output Format Counters
  Bytes Written=800
21/04/15 08:38:10 INFO streaming.StreamJob: Output directory: /pagerank/final_output
cloudera@quickstart ~$
cloudera@quickstart ~$ hdfs dfs -ls /pagerank/final_output
Found 2 items
-rw-r--r-- 1 cloudera supergroup 0 2021-04-15 08:38 /pagerank/final_output/ SUCCESS
-rw-r--r-- 1 cloudera supergroup 800 2021-04-15 08:38 /pagerank/final_output/part-00000
cloudera@quickstart ~$
```

Output directory of  
MapReduce job2

Output files of  
MapReduce job2

Alright, both the MapReduce jobs have been executed successfully. We can now read the final output. The final output is written inside the **final\_output** directory on HDFS.

To read the file, simply run the command below.

`hdfs dfs -cat /pagerank/final_output/part-00000`



```
cloudera@quickstart:~$ hdfs dfs -cat /pagerank/final_output/part-00000
Current node: 1
Contribution by this node: 0.0
Current node: 1
Contribution by this node: 0.333333333333
Current node: 2
Contribution by this node: 0.0
Previous node: 1
New page rank of node 2: 0.43333
Current node: 2
Contribution by this node: 0.333333333333
Current node: 2
Contribution by this node: 0.5
Current node: 3
Contribution by this node: 0.0
Previous node: 2
New page rank of node 3: 0.85833
Current node: 3
Contribution by this node: 0.5
Current node: 3
Contribution by this node: 1.0
Current node: 4
Contribution by this node: 0.0
```

Great, we received the same output as we did while executing locally.

Congratulations, for successfully executing PageRank algorithm using MapReduce on Hadoop. Note that the page rank values would vary, depending upon the no. of nodes in the input web graph, no. of incoming and outgoing edges associated with each node and the damping factor you choose for the same.