BIG DATS PROGRAMING

ASSIGNMENT-2

1 To implement the PageRank algorithm using map reduce in hadoop, given adjacency list and initial page rank values in two different text files. The primary step here is to read the two text files and split them, then following the map reduce process for page rank algorithm implementation. But this process contains of reading two text files lead to more number of mappers and its very computationally expensive. To overcome this in this assignment Distributed cache of hadoop is used so that one of the files is cached and the other is used as input in here. The Intial Page rank values file is cached and Adjacency list file is used as input file.

The overall design flow of map reduce for Page Rank is shown in below:

The Page rank algorithm is a iterative algorithm and In each iteration rank values are computes and updated with previous values until the change in values from present and previous iteration is around 10^-6, i.e. the values are very similar with minute error.

Based on the given graph the rank values are computed using the standard page rank formula. And the generalized version for computing ranks is shown below.

$$\mathbf{r} = c\mathbf{P}^\top\mathbf{r} + (1 - c)/n$$

In each iteration we take the updated page rank value file and upload it into the cache and also the rank values are outputted into new file. This type of iteration is also called as power iteration

$$\mathbf{r}^t = c \cdot \mathbf{P}^\top \cdot \mathbf{r}^{t-1} + (1 - \mathrm{c})\mathbf{1}/n$$

r_i^t : PageRank value of node i at iteration t

$$\begin{aligned}
r_1^t &= c \cdot r_4^{t-1} \cdot p_{4,1} + (1-c)/5 \\
r_2^t &= c \cdot r_1^{t-1} \cdot p_{1,2} + (1-c)/5 \\
r_3^t &= c \cdot r_2^{t-1} \cdot p_{2,3} + c \cdot r_5^{t-1} \cdot p_{5,3} + (1-c)/5 \\
r_4^t &= c \cdot r_2^{t-1} \cdot p_{2,4} + c \cdot r_3^{t-1} \cdot p_{3,4} + (1-c)/5 \\
r_5^t &= c \cdot r_4^{t-1} \cdot p_{4,5} + (1-c)/5
\end{aligned}$$

1.1 Design of Mapper, This takes the input adjacency list row and also the Page Rank values, But as mentioned above, The page rank values are stored in Distributed cache and not given as input file path like adjacency list.

Let's see how the Page rank rank values are used in Mapper from Distributed cache, The pagerank.txt file is stored in distributed cache, This file is used in mapper class in such a way the values of PageRank are stored in a Hash Map.

Distributed cache, The hdfs file system sends small text files to each datanodes while running map reduce job through distributed cache. And these files are deleted after the job is completed from data nodes. This usage will decrease the time taken by reading file each and every time from hdfs.

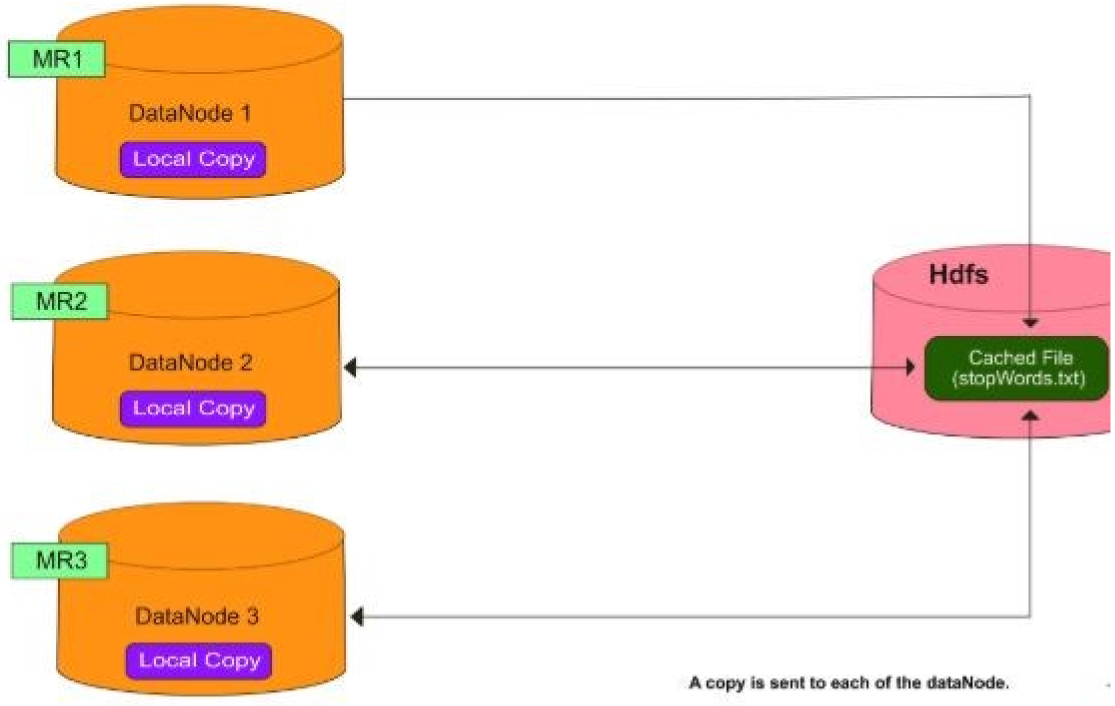


Fig: Showing the working of Distributed cache

Before Mapper design, the data from distributed cache is loaded and stored in a Hash Map, The Hashmap consists of key as nth node index and values as the rank of that node. This can be achieved by overriding the setup method, And this process follow as shown below.

As a initial step read the cached files into URI array and check whether there are any files, If there are any files create File system object and pass configuration object into it, File system is a abstract class of fairly generic file system, After this open the file using File system object and convert the byte stream into character with input stream reader and store it as buffer reader. As next step check this buffered for not empty and read line by line until its empty, Using string tokenize each line is tokenized and the first token is used as key i.e nth node index and the value as PageRank until all the buffer reader is empty. The output here is the Hash Map with key as nth node index and PageRank value of it.

The Hash table created above is used for fetching page rank values based on the adjacent list node index values. As shown below.

Hash Map:

Key	value
1	0.2
2	0.2
3	0.2
4	0.2
5	0.2

Adjacency List:

I	j
1	[2]
2	[3, 4]
3	[4]
4	[1, 5]
5	[3]

As show above the Mapper would read each line of adjacency list file and in this line is in <Text value> and the key be object, From the Object the value is converted into string and string tokenize it. After this the first value of tokenizer is used as key to hash map for getting the page rank value (from above I is the key for hashmap). Once we got the key next all tokens are taken into loop and compute rank*transition probability of the nodes and output it with <token, rank*transition probability> the transition probability is calculates using len(tokens)-1

Output of mapper for line 2 of adjacency list is shown below

<3, hashmap[2]*1/len(tokenizers-1)>

<4, hashmap[2]*1/len(tokenizers-1)>

In above 3, 4 represents nodes and in a similar for all nodes its calculated.

1.2 Reducer:

After Shuffling and sorting the input for reducer is <node number, list of values>. For example

<1,[0.25]>

<3, [0.2, 0.1]> etc…..

In the reducer For each of this input we compute the final using formula mentioned above and the output is written as <node number, rank>

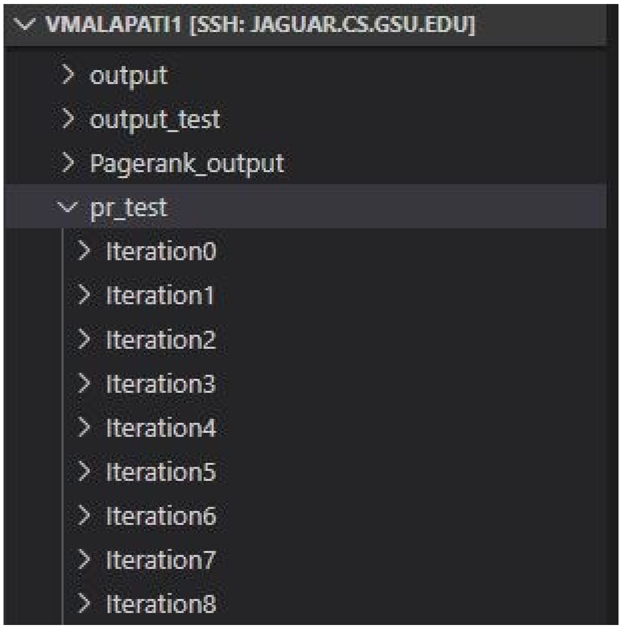
And this process is continued until 30 iterations in the main/driver class of map reduce process.

1.3 I haven’t used a combiner in this process, as the data and graph is small it doesn’t have that great effect on the network congestion. And if we use combiner in this problem it will be after mapper output and before shuffle and sorting so it does not have any effect. And also the combiner is nothing but the reducer and one can use same reducer as combiner in many cases but it fails if the reducer that is implemented here for a single mapper in this case doesn’t act as combiner. May be one can write separate class for reducer that act as combiner and further more customized class for reducer. This Combiner system may not work for multiple mapper if the same reducer is used.

2. The Experimentation results are shown below.

1	0.15555492363830498
2	0.16223821815273626
3	0.23119216214996707
4	0.29545977242068655
5	0.15555492363830498

2.1



VMALAPATI1 [SSH: JAGUAR.CS.GSU.EDU]

output

output_test

Pagerank_output

pr_test

Iteration0

Iteration1

Iteration2

Iteration3

Iteration4

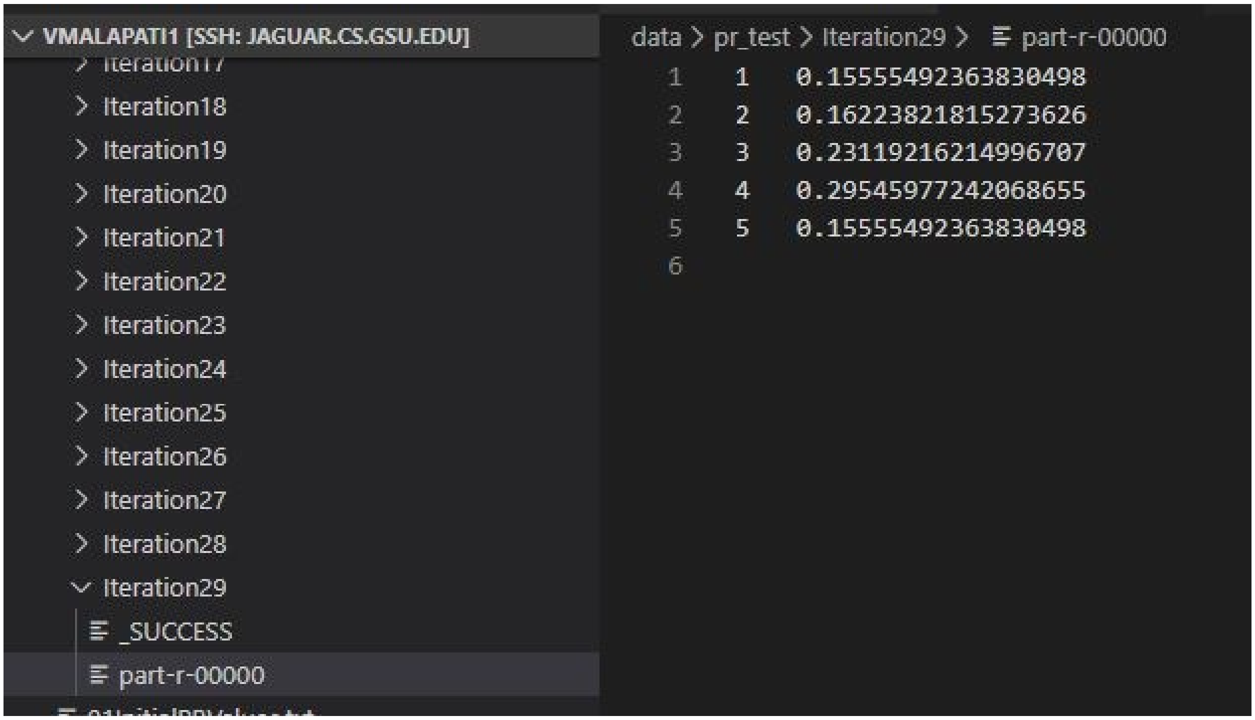
Iteration5

Iteration6

Iteration7

Iteration8

Fig: Showing output files created after 30 iterations

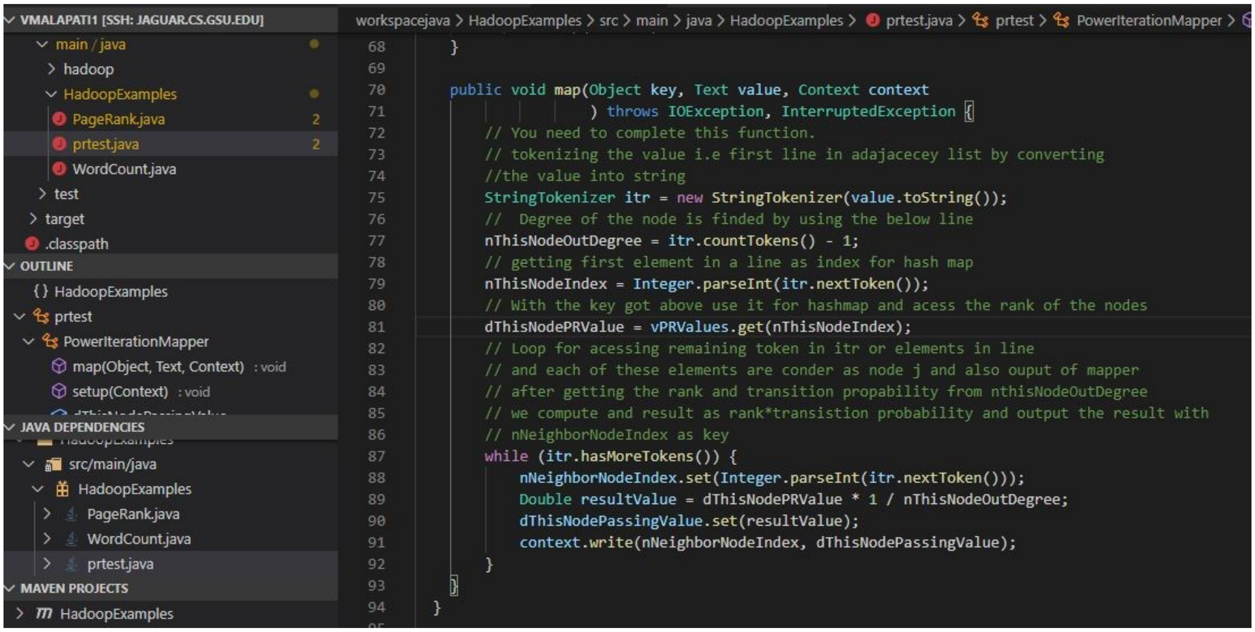


```
1   0.15555492363830498
2   0.16223821815273626
3   0.23119216214996707
4   0.29545977242068655
5   0.15555492363830498
```

Fig: showing result after 30 iterations

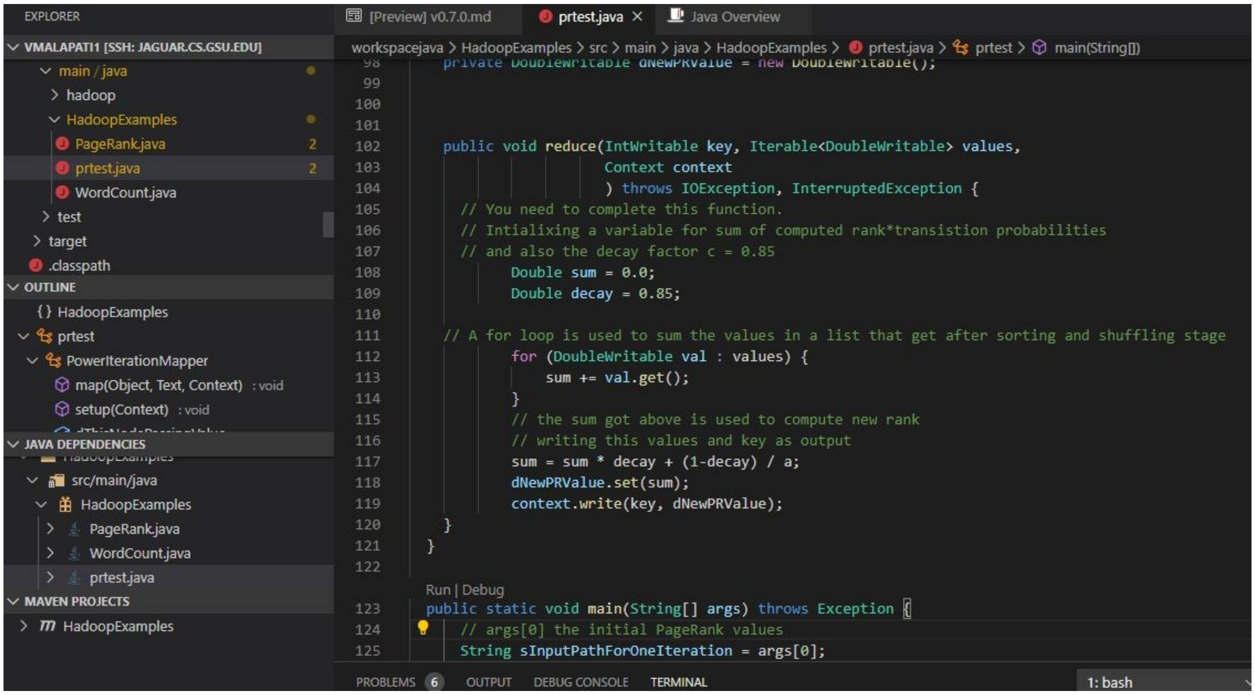
The results are as shown above and these results are from the 30th iteration. The error in results from 29th iteration to 30th iteration is nearly 10^(-4). Since there is very minute change in results we stopped iterations at this point.

The results are as equal to the results that are given in the class.



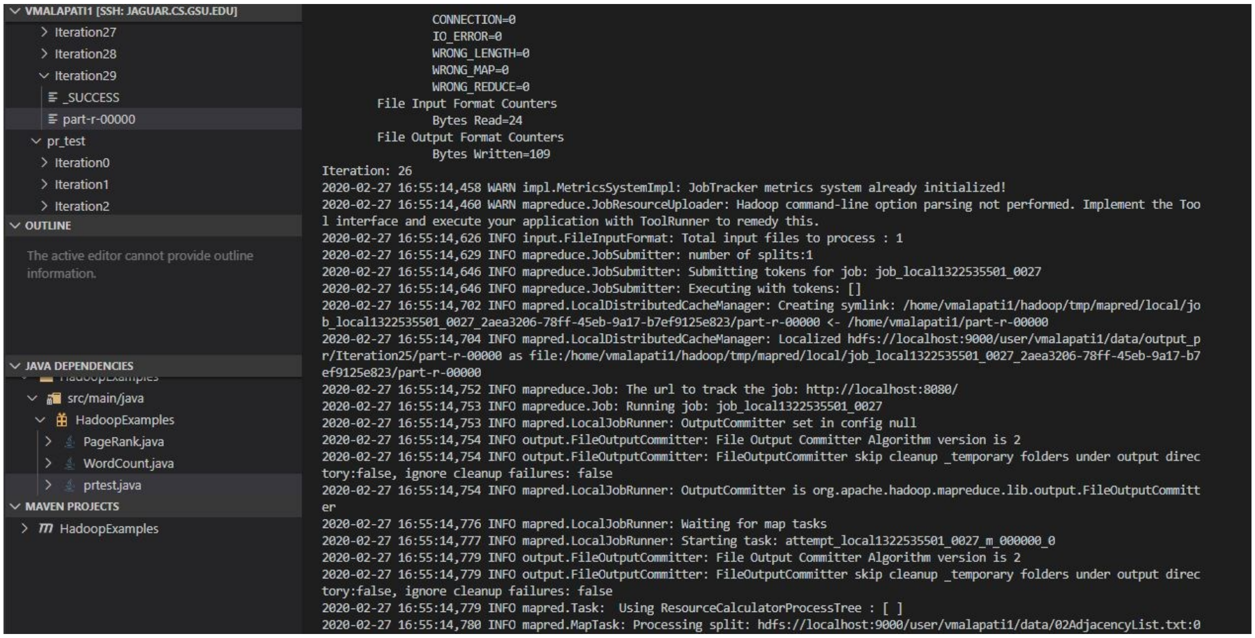
```java
    }

    public void map(Object key, Text value, Context context
                ) throws IOException, InterruptedException {
        // You need to complete this function.
        // tokenizing the value i.e first line in adajacecey list by converting
        //the value into string
        StringTokenizer itr = new StringTokenizer(value.toString());
        //  Degree of the node is finded by using the below line
        nThisNodeOutDegree = itr.countTokens() - 1;
        // getting first element in a line as index for hash map
        nThisNodeIndex = Integer.parseInt(itr.nextToken());
        // With the key got above use it for hashmap and acess the rank of the nodes
        dThisNodePRValue = vPRValues.get(nThisNodeIndex);
        // Loop for acessing remaining token in itr or elements in line
        // and each of these elements are conder as node j and also ouput of mapper
        // after getting the rank and transition propability from nthisNodeOutDegree
        // we compute and result as rank*transistion probability and output the result with
        // nNeighborNodeIndex as key
        while (itr.hasMoreTokens()) {
            nNeighborNodeIndex.set(Integer.parseInt(itr.nextToken()));
            Double resultValue = dThisNodePRValue * 1 / nThisNodeOutDegree;
            dThisNodePassingValue.set(resultValue);
            context.write(nNeighborNodeIndex, dThisNodePassingValue);
        }
    }
}
```

Fig: Showing the mapper code



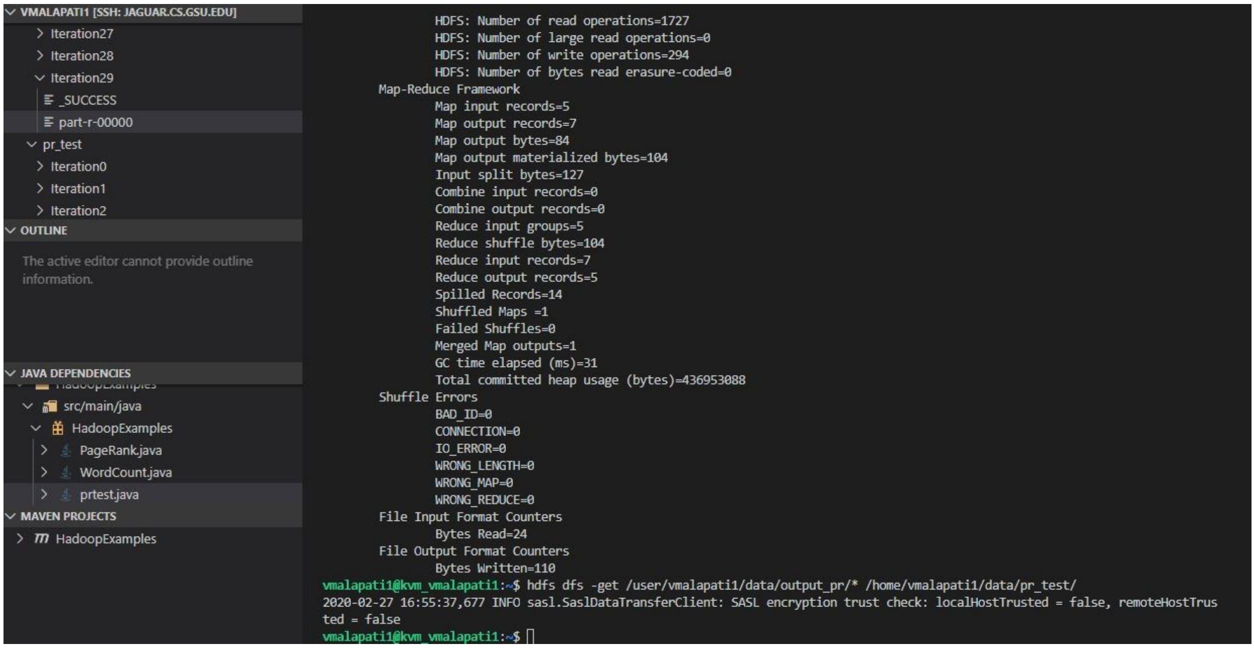
```java
    private DoubleWritable dNewPRValue = new DoubleWritable();


    public void reduce(IntWritable key, Iterable<DoubleWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      // You need to complete this function.
      // Intialixing a variable for sum of computed rank*transistion probabilities
      // and also the decay factor c = 0.85
          Double sum = 0.0;
          Double decay = 0.85;

    // A for loop is used to sum the values in a list that get after sorting and shuffling stage
          for (DoubleWritable val : values) {
              sum += val.get();
          }
          // the sum got above is used to compute new rank
          // writing this values and key as output
          sum = sum * decay + (1-decay) / a;
          dNewPRValue.set(sum);
          context.write(key, dNewPRValue);
    }
  }

  public static void main(String[] args) throws Exception {
      // args[0] the initial PageRank values
      String sInputPathForOneIteration = args[0];
```

Fig: Showing reducer code



```
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=24
        File Output Format Counters
                Bytes Written=109
Iteration: 26
2020-02-27 16:55:14,458 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
2020-02-27 16:55:14,460 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Too
l interface and execute your application with ToolRunner to remedy this.
2020-02-27 16:55:14,626 INFO input.FileInputFormat: Total input files to process : 1
2020-02-27 16:55:14,629 INFO mapreduce.JobSubmitter: number of splits:1
2020-02-27 16:55:14,646 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1322535501_0027
2020-02-27 16:55:14,646 INFO mapreduce.JobSubmitter: Executing with tokens: []
2020-02-27 16:55:14,702 INFO mapred.LocalDistributedCacheManager: Creating symlink: /home/vmalapati1/hadoop/tmp/mapred/local/jo
b_local1322535501_0027_2aea3206-78ff-45eb-9a17-b7ef9125e823/part-r-00000 <- /home/vmalapati1/part-r-00000
2020-02-27 16:55:14,704 INFO mapred.LocalDistributedCacheManager: Localized hdfs://localhost:9000/user/vmalapati1/data/output_p
r/Iteration25/part-r-00000 as file:/home/vmalapati1/hadoop/tmp/mapred/local/job_local1322535501_0027_2aea3206-78ff-45eb-9a17-b7
ef9125e823/part-r-00000
2020-02-27 16:55:14,752 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2020-02-27 16:55:14,753 INFO mapreduce.Job: Running job: job_local1322535501_0027
2020-02-27 16:55:14,753 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2020-02-27 16:55:14,754 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2020-02-27 16:55:14,754 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output direc
tory:false, ignore cleanup failures: false
2020-02-27 16:55:14,754 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapreduce.lib.output.FileOutputCommitt
er
2020-02-27 16:55:14,776 INFO mapred.LocalJobRunner: Waiting for map tasks
2020-02-27 16:55:14,777 INFO mapred.LocalJobRunner: Starting task: attempt_local1322535501_0027_m_000000_0
2020-02-27 16:55:14,779 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2020-02-27 16:55:14,779 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under output direc
tory:false, ignore cleanup failures: false
2020-02-27 16:55:14,779 INFO mapred.Task:  Using ResourceCalculatorProcessTree : [ ]
2020-02-27 16:55:14,780 INFO mapred.MapTask: Processing split: hdfs://localhost:9000/user/vmalapati1/data/02AdjacencyList.txt:0
```

Fig: Showing the printed screen at 26th iteration during execution of the code



```
                HDFS: Number of read operations=1727
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=294
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=5
                Map output records=7
                Map output bytes=84
                Map output materialized bytes=104
                Input split bytes=127
                Combine input records=0
                Combine output records=0
                Reduce input groups=5
                Reduce shuffle bytes=104
                Reduce input records=7
                Reduce output records=5
                Spilled Records=14
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=31
                Total committed heap usage (bytes)=436953088
        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=24
        File Output Format Counters
                Bytes Written=110
vmalapati1@kvm_vmalapati1:~$ hdfs dfs -get /user/vmalapati1/data/output_pr/* /home/vmalapati1/data/pr_test/
2020-02-27 16:55:37,677 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrus
ted = false
vmalapati1@kvm_vmalapati1:~$ 
```

Fig: showing the get usage for getting files in hdfs into local system