Assignment 5: PageRank

The goal of this assignment was to find the find the most popular pages in a set of Web pages. You will use your implementation of PageRank to find the 'most popular' pages in a dump of Wikipedia pages.

Decomposing of program, that is breaking program into few small, simple and sequential tasks which can be executed parallel by multiple computers is most important. The basic workflow should be:

initial mapreduce -> iterate mapreducr -> evaluate difference -> join name -> finish

I should emphasize on the finish phase. I have broken down the FINISH phase into two parts: ***finishjoin, finishcombine***. Meanwhile, one must change the threshold to adapt to different tasks. Let's move on the next process, DIFF, of which major responsibility is to calculate difference between the intermediate values produced by ITER. If the difference is small enough, we then consider mapreduce is stable and output the result. ITER is a process where we compute the rank. INIT is the first phase we initialize all the value.

Below is the summary of code written in the various files.

In DiffMap1.java file, completed the block of code in Map method to read node-rank pair and emit: key:node, value:rank.

```
18          /**
19           *  TODO: read node-rank pair and emit: key:node, value:rank
20           */
21          //Separating Node and Rank
22          String[] noderank = sections[0].split("\\+");
23          //Emits Node and Rank
24          context.write(new Text(noderank[0]),  new Text(noderank[1]));
25
26      }
```

In DiffMap2.java, completed the block of code in map method to emit the key, value pair where key is the difference and value is difference calculated in DiffRed1.java.

```
14          /*
15           * TODO: emit: key:"Difference" value:difference calculated in DiffRed1
16           */
17
18          String[] noderank = s.split("\t+");
19          context.write(new Text("Difference"), new Text(noderank[1]));
20      }
21
```

In DiffRed1.java file completed the block of code in reduce method which creates the list that contains the two ranks and out difference between those two ranks.

```
13          /*
14           * TODO: The list of values should contain two ranks.  Compute and output their difference.
15           */
16
17          Iterator<Text> iter = values.iterator();
18          double diff = 0;
19          // Rank 1 Calculation
20          if(iter.hasNext()) {
21              ranks[0] = Double.valueOf(iter.next().toString());
22          }
23          // Rank 2 Calculation
24          if(iter.hasNext()) {
25              ranks[1] = Double.valueOf(iter.next().toString());
26          }
27          // Difference Calculation
28          diff = Math.abs(ranks[0] - ranks[1]);
29          System.out.println( key.toString() + "   " + diff);
30          context.write(key, new Text(String.valueOf(diff)));
31      }
```

In DiffRed2.java file, completed the block of code in reduce method which computes the emits the maximum differences.

```
12          /*
13           * TODO: Compute and emit the maximum of the differences
14           */
15
16          Iterator<Text> iter = values.iterator();
17          // Find max difference and print in output
18          while(iter.hasNext()) {
19              double diff = Double.valueOf(iter.next().toString());
20              diff_max = diff_max > diff ? diff_max : diff;
21          }
22          context.write(new Text(""), new Text(String.valueOf(diff_max)));
23      }
```

In FindJoinMapper.java and FindJoinReducer.java written the code which joins the final output with linked name. The detailed code of the two files is as below:

FindJoinMapper.java

```
public void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException, IllegalArgumentException {
    String line = value.toString(); // Converts Line to a String
    /*
     * Join final output with link name
     * input: key: nodeId+rank, text: adjacent list
     * input: key: nodeId, text: name
     *
     * output: key: nodeId, text: rank
     * ouput: key: nodeId, text:names
     */

    String[] section;
    if (line.contains(":")) {
        int ind = line.indexOf(":");
        section = new String[2];
        section[0] = line.substring(0, ind);
        section[1] = line.substring(ind + 1, line.length());
    } else {
        section = line.split("\t"); // Splits it into two parts. Part 1: node+rank | Part 2: adj list
    }


    if (section.length > 2) // Verfiy Correct Data Format
    {
        throw new IOException("INVALID DATA FORMAT");
    }
    String[] noderank = section[0].split("\\+");
    if(noderank.length == 1) {
        // nodeID along with name
        context.write(new Text(noderank[0]), new Text(PageRankDriver.MARKER_NAME + section[1].trim()));
    }

    if(noderank.length == 2) {
        // nodeID along with  rank
        context.write(new Text(noderank[0]), new Text(PageRankDriver.MARKER_RANK + noderank[1]));
    }
}
```

FindJoinReducer.java

```
package edu.stevens.cs549.hadoop.pagerank;

import java.io.IOException;

public class FindJoinReducer extends Reducer<Text, Text, Text, Text> {

    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException, IllegalArgumentException {

        /*
         * Join final output with linked name
         * input: key: nodeId, val: (mark_rank)rank
         * input: key: nodeId, val: (mark_name))name
         *
         * output: key: nodeId+names, text: rank
         */
        Iterator<Text> iter = values.iterator();
        String nodeName = "";
        String rank  = "";
        while(iter.hasNext()) {
            String temp = iter.next().toString();
            if(temp.startsWith(PageRankDriver.MARKER_NAME)) {
                nodeName = temp.replaceAll(PageRankDriver.MARKER_NAME, "");
            }
            if(temp.startsWith(PageRankDriver.MARKER_RANK)) {
                rank = temp.replaceAll(PageRankDriver.MARKER_RANK, "");
            }
        }

        context.write(new Text(key + "+" + nodeName) , new Text(rank));

    }
```

In FinMapper.jav file, completed the block of code which output the rank and node as key, value.

```java
public class FinMapper extends Mapper<LongWritable, Text, DoubleWritable, Text> {

    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException, IllegalArgumentException {
        String line = value.toString(); // Converts Line to a String
        /*
         * TODO output key:-rank, value: node
         * See IterMapper for hints on parsing the output of IterReducer.
         */

        String[] section = line.split("\t"); // nodeId+nodeName | rank

        if (section.length > 2) //Verify correct data format
        {
            throw new IOException("INVALID DATA FORMAT");
        }
        if (section.length != 2) {
            return;
        }
        // Reverse shuffle redce logic
        context.write(new DoubleWritable(0 - Double.valueOf(section[1])), new Text(section[0]));

    }
```

In FinReducer.java completed the block of code to emit each key value pair of value and rank.

```java
public class FinReducer extends Reducer<DoubleWritable, Text, Text, Text> {

    public void reduce(DoubleWritable key, Iterable<Text> values, Context context) throws IOException,
            InterruptedException {
        /*
         * TODO: For each value, emit: key:value, value:-rank
         */

        Iterator<Text> iter = values.iterator();
        String node;
        while(iter.hasNext()) {
            node = iter.next().toString();
            // convert -rank back to rank
            context.write(new Text(node), new Text(String.valueOf(0 - key.get())));
        }
    }
}
```

In InitMapper.java file, completed the block of code to echo the input which is already in th e adjacency list.

```
10    public void map(LongWritable key, Text value, Context context) throws IOException, In
11            IllegalArgumentException {
12        String line = value.toString(); // Converts Line to a String
13        /*
14         * TODO: Just echo the input, since it is already in adjacency list format.
15         */
16
17        /*
18         * split the line by symbol ":", and output key, adjacent list to reducer
19         */
20        String[] p = line.split(":");
21        if(p != null && p.length == 2) {
22            context.write(new Text(p[0].trim()), new Text(p[1]));
23        }
24
25    }
26
27 }
28
```

In InitRedcuer.java file, completed the block of code which outputs the node rank and value in the adjacency list.

```
3 import java.io.*;
7
8 public class InitReducer extends Reducer<Text, Text, Text, Text> {
9
10    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
11        /*
12         * TODO: Output key: node+rank, value: adjacency list
13         */
14
15        /*
16         * Since default rank is 1, so we need only output node+rank and adjacency list
17         */
18        int defualt_rank = 1;
19        Iterator<Text> val = values.iterator();
20        while(val.hasNext()) {
21            context.write(new Text(key + "+" + defualt_rank), val.next());
22        }
```

In IterMapper.java completed the block of code to emit adjacent vertex and computed weight as key, value pair. While doing so I have taken care to emit input adjacency list of the node.

```
10    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException,
11            IllegalArgumentException {
12        String line = value.toString(); // Converts Line to a String
13        String[] sections = line.split("\t"); // Splits it into two parts. Part 1: node+rank | Part 2: adj list
14
15        if (sections.length > 2) // Checks if the data is in the incorrect format
16        {
17            throw new IOException("Incorrect data format");
18        }
19        if (sections.length != 2) {
20            return;
21        }
22
23        /*
24         * TODO: emit key: adj vertex, value: computed weight.
25         *
26         * Remember to also emit the input adjacency list for this node!
27         * Put a marker on the string value to indicate it is an adjacency list.
28         */
29        // split node+rank
30        String[] node_rank = sections[0].split("\\+");
31        String node = String.valueOf(node_rank[0]);
32        double rank = Double.valueOf(node_rank[1]);
33        String adj_list = sections[1].toString().trim();
34
35        String[] adj_nodes = adj_list.split(" ");
36        int N = adj_nodes.length; // outgoing links number
37        // 1/n * rank
38        double weightOfPage = (double)1/N * rank; // calculate current page weight if outgoing to other links
39        for(String adj_node : adj_nodes) {
40            context.write(new Text(adj_node), new Text(String.valueOf(weightOfPage)));
41        }
42        // at the same time, emit current node's adj_list list with marker "ADJ:"
43        context.write(new Text(node), new Text(PageRankDriver.MARKER + sections[1]));
44
45    }
```

In IterReducer.java file, completed bock of code in reduce method which outputs the node+rank and adjacency list as key, value pair. Here as suggested by professor, I have used PageRank algorithm to compute the rank from weights contributed by incoming edges.

```
public void reduce(Text key, Iterable<Text> values, Context context) throw
    double d = PageRankDriver.DECAY; // Decay factor
    /*
     * TODO: emit key:node+rank, value: adjacency list
     * Use PageRank algorithm to compute rank from weights contributed by
     * Remember that one of the values will be marked as the adjacency li
     */

    Iterator<Text> iter = values.iterator();
    double curr_Rank = 0; // default rank is 1 - d
    String adj_list = "";
    while(iter.hasNext()) {
        String line = iter.next().toString();
        if(!line.startsWith(PageRankDriver.MARKER)) {
            curr_Rank += Double.valueOf(line);
        } else {
            adj_list = line.replaceAll(PageRankDriver.MARKER, "");
        }
    }
    //sum calculations
    curr_Rank = 1 - d + curr_Rank * d;
    context.write(new Text(key + "+" + curr_Rank), new Text(adj_list));
}
```

In PageRankDriver.java file, written FinishJoin method through which operation is implemented to perform the join function. Through which you can set the Mapper and Reducer.

```
214  static void finishJoin(String input, String output, int reducers)
215          throws Exception {
216      String temp_Join_Dir = "tempJoin";
217      System.out.println("Finish Job Started");
218      Job j = Job.getInstance(); // Creates a new Job
219      j.setJarByClass(PageRankDriver.class); // Sets the Driver class
220      j.setNumReduceTasks(reducers); // Sets the number of reducers
221
222      FileInputFormat.addInputPath(j, new Path(input)); // Adds input and output paths
223      //Input
224      FileInputFormat.addInputPath(j, new Path(CACHE_DIR));
225      FileOutputFormat.setOutputPath(j, new Path(temp_Join_Dir));
226      // Sets Mapper and Reducer Classes
227      j.setMapperClass(FindJoinMapper.class);
228      j.setReducerClass(FindJoinReducer.class);
229      // Sets Mapper and Reducer output types
230      j.setMapOutputKeyClass(Text.class);
231      j.setMapOutputValueClass(Text.class);
232
233      j.setOutputKeyClass(Text.class);
234      j.setOutputValueClass(Text.class);
235
236      // Prints message on successful completion or error.
237      if(j.waitForCompletion(true)) {
238          finishCombine(temp_Join_Dir, output, reducers);
239      }
240  }
```

Within the same file, written a function named finishCombine which will finish the job end and make the job started.

```
static void finishCombine(String input, String output, int reducers)
        throws Exception {
    System.out.println("Finish Join Job end, and Finish Job Started");
    // Creates a new Job
    Job j = Job.getInstance();
     // Sets the Driver class
    j.setJarByClass(PageRankDriver.class);
    // Sets the number of reducers
    j.setNumReduceTasks(reducers);
    // Adds input and output paths
    FileInputFormat.addInputPath(j, new Path(input));
    FileOutputFormat.setOutputPath(j, new Path(output));

    j.setMapperClass(FinMapper.class); // Sets Mapper and Reducer Classes
    j.setReducerClass(FinReducer.class);

    j.setMapOutputKeyClass(DoubleWritable.class); // Sets Mapper and Reducer output types
    j.setMapOutputValueClass(Text.class);

    j.setOutputKeyClass(Text.class);
    j.setOutputValueClass(Text.class);

    // Prints message on successful completion or error.
    System.out.println(j.waitForCompletion(true) ? "Finish Job Completed" : "Finish Job Error");
    // Exits once job finishes.

    deleteDirectory("tempJoin"); // Deletes the input directory
}
```

In Composite method, as suggested I have added my name and CWID.

```
273     public static void composite(String input, String output, String interim1,
274             String interim2, String diff, int reducers) throws Exception {
275         /*
276          * TODO
277          */
278         System.out.println("Abhilash Ugaonkar (10415787)");
279
```

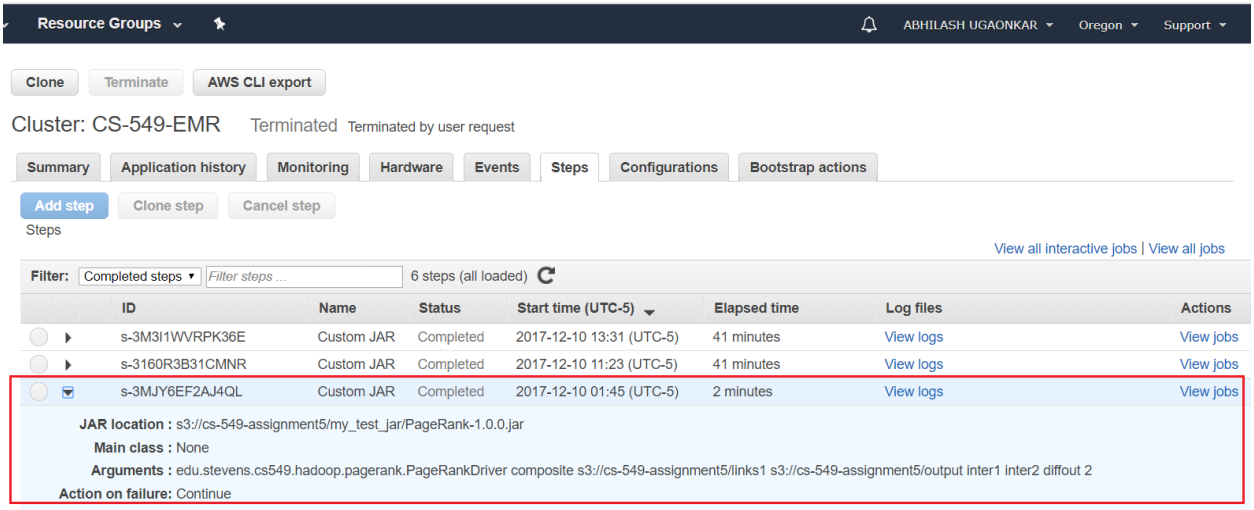
Experimenting with different number of Reducers

On EMR, I first ran the job on test data given in the assignment specification. While running that particular job I had used Reducers= 2

The output of the above job is as below:

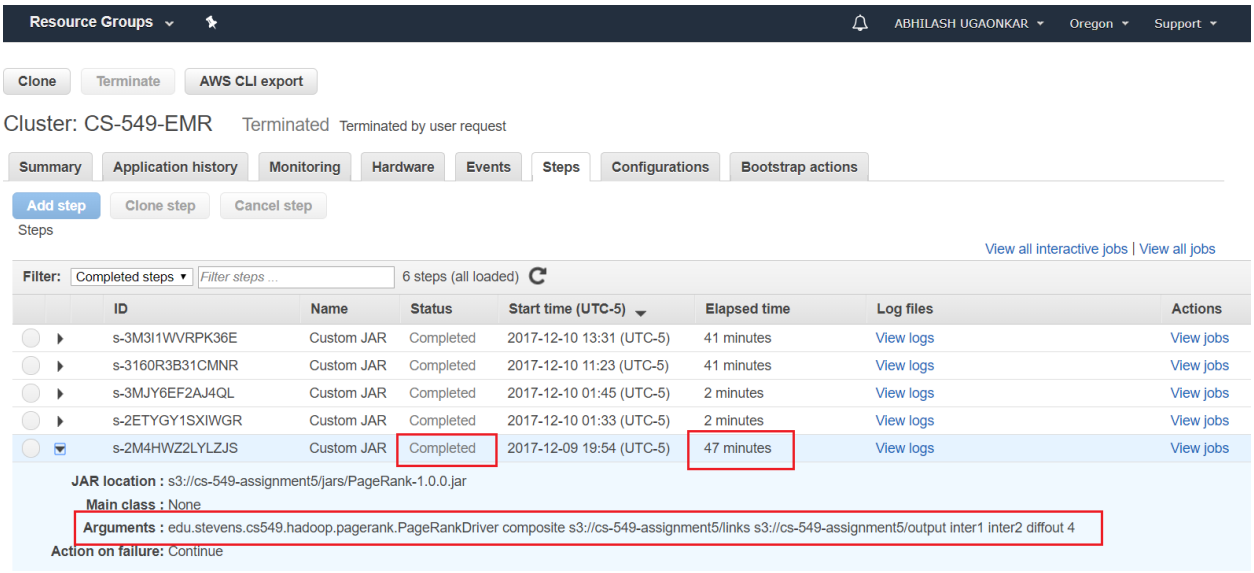
```
D   1.708333333333333
A   0.8583333333333333
C   0.575
B   0.575
E   0.43333333333333335
```

The job was completed in two minutes. Here are the details of job run.



Thereafter, I ran the job on Wikipedia data downloaded from the links given in the assignment specification.
Reducers = 4

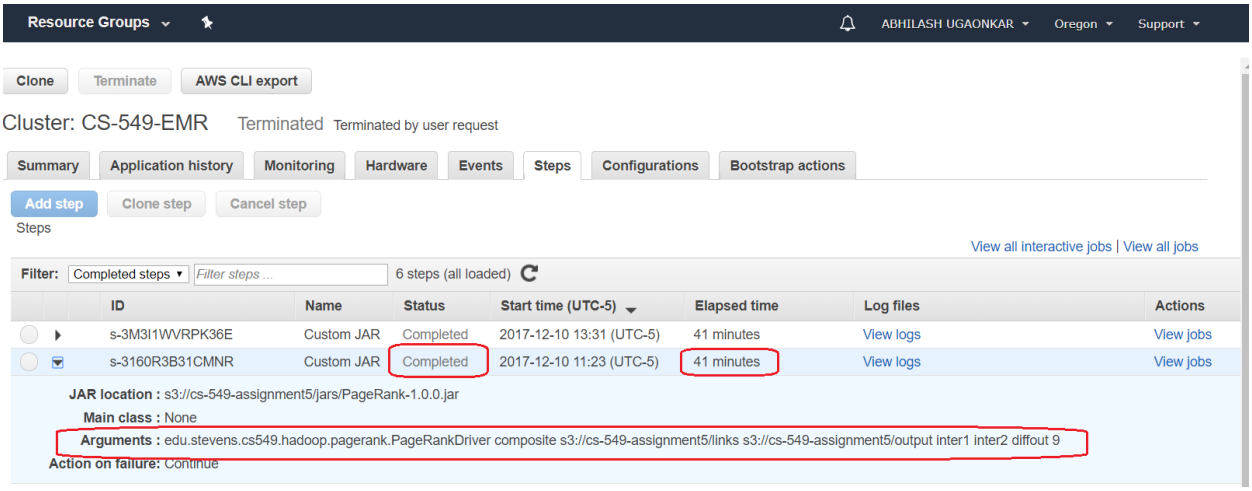
edu.stevens.cs549.hadoop.pagerank.PageRankDriver composite s3://cs-549-assignment5/links s3://cs-549-assignment5/output inter1 inter2 diffout 4



It took 47 minutes to finish the job. Output of the job is attached in my submission.

Going further, I increased the number of reducers from 4 to 9.
Reducers = 9

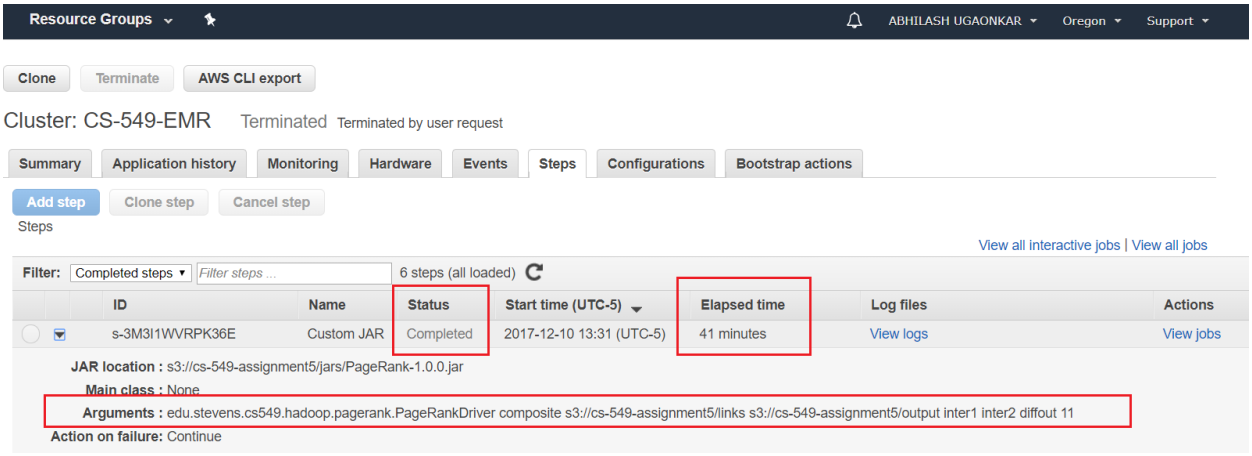
edu.stevens.cs549.hadoop.pagerank.PageRankDriver composite s3://cs-549-assignment5/links s3://cs-549-assignment5/output inter1 inter2 diffout 9



Surprisingly, even though the number of reducers were increased it took only 41 minutes to complete the job. I have recorded the execution steps in one of the demo videos.

After this I thought of increasing number of reducers further to 11 and below were the results.
Reducers = 11

edu.stevens.cs549.hadoop.pagerank.PageRankDriver composite s3://cs-549-assignment5/links s3://cs-549-assignment5/output inter1 inter2 diffout 11



Surprisingly, it again took 41 minutes to finish the job. I have attached the output of the job in the submission.

I had thought that as the number of reducers will increase time to execute will also increase, however, in my case when I ran the above jobs the taken decreased and remain same equal for reducers = 9 and reducers = 11.

Ideally, my experience is as the number of reducers increases it takes longer to complete job.

I have attached the demonstration video, code, jars and report in the submission.