CS 5433: Bigdata Management
Programming Assignment 1
PART 3 – MapReduce Program for Hashtag Count using Partitioner
CWID: A20343337

```
import com.google.gson.JsonElement; //importing JsonElement from gson library
import com.google.gson.JsonObject; //importing JsonObject from gson library
import com.google.gson.JsonParser;//importing JsonParser from gson library
import java.util.Set;
import org.apache.hadoop.fs.*;
import java.util.HashSet;
import java.util.StringTokenizer;//importing StringTokenizer from util library
import java.util.ArrayList; //importing ArrayLists
import org.apache.commons.lang3.StringUtils;
// Below are mapreduce packages
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Partitioner;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;
import java.util.Scanner;


public class HashPart2 {
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
//Extending mapper class
    {


        @Override
        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException {
            JsonParser parser = new JsonParser(); //creating the object for
JsonParser
```

```
            JsonElement totalTweet = parser.parse(value.toString()); //parsing
the json data into string
            JsonObject totalTweetAsJsonObject = totalTweet.getAsJsonObject();
//Creating the object for JsonObject
            String text = totalTweetAsJsonObject.get("text").toString();
//Retreiving the text field from the tweets
            Configuration configuration = context.getConfiguration(); //Creating
configuration object
        ArrayList<String> words = new ArrayList<>();//creation of array list
            String temp = "";
            String keyword = configuration.get("keyword");//Retriving the keyword
dynamically from the driver code
            if (StringUtils.containsIgnoreCase(text, keyword)) { //Ignores the
cases in the keyword
                String line = text.replaceAll("[^a-zA-Z0-9#]", " "); //replaces
the special characters with spaces in the text
                StringTokenizer tokens = new StringTokenizer(line); //Tokenizes
the text into words
                while(tokens.hasMoreTokens())
                {
                    temp=tokens.nextToken();
                    words.add(temp);
                }
        int i=0;
        String[] token = new String[words.size()];
        words.toArray(token);
                while (i<token.length) { // iterating the loop over the token
array length
            String wor  = token[i];
                    if (wor.startsWith("#")) { //if the word starts with '#' then
outputs the word+keyword and value 1
                            context.write(new Text(wor+ " " + keyword), new
IntWritable(1));
                    }
        i++;
                }
            }
        }
    }


    public static class Reduce extends Reducer<Text, IntWritable, Text,
IntWritable> { //extending the reducer class
    Set<String> parsedKeys = new HashSet<>(); //intializing the hash set
        @Override
```

```
        public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
    parsedKeys.add(key.toString()); //appending the keys to the hashset
          if (parsedKeys.size() > 10) { //if the hashset size greater than 10
then the reduce method will comeout from the reducer class
             return;
            }
        int hashtagsSum = 0;
            for(IntWritable val : values) {
                hashtagsSum += val.get(); //calculates the hashtags count
            }
            context.write(key, new IntWritable(hashtagsSum));
        }
    }
    public static class myPartitioner extends Partitioner < Text, IntWritable >
//extending the Partitioner class
   {

    public int getPartition(Text key, IntWritable value, int numReduceTasks)
    {

        String[] str = key.toString().split(" "); // Spliting the key into two
parts
        String hash = str[0]; // Taking the #hashtag into hash
        int charAscii = hash.toUpperCase().charAt(1); //Converting the first
letter in hashtag after "#" to uppercase
        if(numReduceTasks == 0)
        {
            return 0;
        }

        if((charAscii >= 65 && charAscii <= 71)) // if first letter in hashtag is
in between 'A'-'G' then the key-value pairs goes to reducer 0
        {
           // Send to paritioner 1

          return 0;
        }
        if((charAscii >= 72 && charAscii <= 85)) // if first letter in hashtag is
in between 'H'-'U' then the key-value pairs goes to reducer 1
        {
           // Send to paritioner 2

            return 1%numReduceTasks;
        }
```

```
        else {
            // Send to partitioner 3 // if first letter in hashtag is in between
'V-'Z' then the key-value pairs goes to reducer 2
            return 2%numReduceTasks;
    }

      }
   }

    public static void main(String[] args) throws Exception {
        Configuration conf1 = new Configuration(); //creating configuration
object
        System.out.println("Enter a keyword to search in tweet: ");
        Scanner sc = new Scanner(System.in); //Scanning the input for keyword
        String inputKeyword = sc.nextLine();
        conf1.set("keyword", inputKeyword);
        Job conf = Job.getInstance(conf1, "HashPart2"); //creating job
        conf.setJarByClass(HashPart2.class);  //setting HashPart2 class
        conf.setMapperClass (Map.class); // setting mapper class
        conf.setPartitionerClass(myPartitioner.class);// setting partitioner
class
        conf.setNumReduceTasks(3); // setting number of reducers
        conf.setReducerClass (Reduce.class); // setting reducer class
        conf.setOutputKeyClass(Text.class);                    //setting output
key type
        conf.setOutputValueClass(IntWritable.class);           //setting output
key value
        FileInputFormat.addInputPath(conf, new Path(args[0])); //input path
        Path outputfolder = new Path(args[1]);
        FileOutputFormat.setOutputPath(conf, outputfolder); // output path
        FileSystem fs = FileSystem.get(conf1); //creating filesystem object
        fs.delete(outputfolder, true); //deletes the output folder if it already
exists
        System.exit(conf.waitForCompletion(true) ? 0 : 1);
    }
}
```