Name – Jatin Taneja
HW Number – 1
Class Number – MW
Github Repo –
MapReduce - https://github.ccs.neu.edu/jatintaneja90/MapReduce
Spark - https://github.ccs.neu.edu/jatintaneja90/Spark

Map Reduce Implementation

Mapper pseudocode

```
Map(Object key, Text value, Context context)
{
// input will be a single line in the format => "A","B" \n
strValue = value.toString() //  Convert Input Text value to string
strArray = strValue.split(",") // split strValue using ',' as delimeter
Text user = new Text()
user.set(strArray[1]) // will access 1st index of the array and save it to Text type parameter as
                                  Text is serializable data type
Context.write(user, new IntWritable(1)) // intermediate result to context, adding 1 as
                                                         IntWritable as it is serializable

}
```

Reducer pseudocode

```
Reduce(Text key, Iterable<IntWritable> values, Context context){
{
Int count =0 // initializing count variable with 0
For(IntWritable val:values){ // iterating over the list of values received from context for key
Count += val.get() // add all the values one by one
}
// save the total summed up count to IntWritable data type field as it is serializable
Result.set(count);
// add the final result to context
Context.write(key, result)
}
```

Spark Implementation

// in spark, we will load the input file in the main memory via spark context
Val textFile = sc.textFile(input directory path)
// create a RDD by running split function on each line of textFile, it results in a pair RDD
strArray = textFile.map(line => line.split(“,”)
// now using only 1st index data of this paired RDD we will create another RDD whose 0th index is user and 1st index is count
userCount = strArray.map(user => (user(1),1))
// now we need to group the results based on key and save the counts, so we will achieve a pair RDD by using reducebyKey
Counts = userCount.reduceByKey((x,y) => x+y)
// then we need to save counts RDD on a file
Counts.saveAsTextFile(output dir)

I have used **RDD.toDebugString** scala commands to output execution details of my program

```
Commands used
//    logger.info("Going to print RDD execution plan ");
//    logger.info(counts.toDebugString);
```

Following is the output that I get in logs

```
2018-09-20 20:18:54 INFO  SparkContext:54 - Created broadcast 0 from textFile at CountTwitterFollowers.scala:24
2018-09-20 20:18:55 INFO  FileInputFormat:256 - Total input files to process : 1
2018-09-20 20:18:55 INFO  root:28 - Going to print RDD execution plan
2018-09-20 20:18:55 INFO  root:29 - (40) ShuffledRDD[4] at reduceByKey at CountTwitterFollowers.scala:27 []
 +-(40) MapPartitionsRDD[3] at map at CountTwitterFollowers.scala:26 []
    |  MapPartitionsRDD[2] at map at CountTwitterFollowers.scala:25 []
    |  input MapPartitionsRDD[1] at textFile at CountTwitterFollowers.scala:24 []
    |  input HadoopRDD[0] at textFile at CountTwitterFollowers.scala:24 []
```

According to the execution graph output in the logs, I understand that it’s creating a Hadoop[0] RDD from the textfile, then this HadoopRDD is sent to different partitions as MapPartitionsRDD for parallel processing and then the result of map command(in which I am splitting each input using comma) is saved on MapPartitionsRDD[2] and then result of map function of each user and count 1 is saved in MapParitionsRDD[3] .

Then this MapPartitionRDD[3] is shuffled and aggregated by reduceByKey command and uploaded to ShuffledRDD[4].

Total 40 partitions of data were created during this process.

Speed up Measurements

	MR(secs)	**Spark(secs)**
1st Run 6 servers	99	78
2nd Run 6 servers	102	74
Average	**100.5**	**76**
1st Run 11 servers	84	73
2nd Run 11 servers	63	74
Average	**73.5**	**73.5**
Speed up (6 servers performance/ 11 servers performance)	**1.36734694**	**1.03401361**

Table describing number of reducer tasks that were running.

	MR	**Spark**
6 servers	8	20
11 servers	19	20

Speed up is not optimal according to my understanding as we are doubling the number of instances and hence we should see speed up of around 2 but current speed up is around just 1 which means adding more instances didn't actually boosted parallel execution.
As we have seen in case of Map reduce, even after increasing the number of tasks to double, speedup is not near 2, and in spark, there is no difference in processing tasks even after increasing the number of workers servers. This is why I feel that speed up is not optimal.

Data transfer between mapper, master and reducer for 2 clusters configuration of map reduce

	Data transfer to mapper (bytes)	**Data transfer between mapper and reducer**	**Data transfer from reducer (bytes)**

		(bytes)	
5 clusters	1319560737	92935036	67641452
11 clusters	1319560737	92935036	67641452

<u>Output directory location</u>

MR output

6 server cluster output
https://github.ccs.neu.edu/jatintaneja90/Spark/tree/master/aws_output_6

11 servers cluster output
https://github.ccs.neu.edu/jatintaneja90/Spark/tree/master/aws_output_11

Spark output

6 server cluster output
https://github.ccs.neu.edu/jatintaneja90/MapReduce/tree/master/aws_output_6

11 servers cluster output
https://github.ccs.neu.edu/jatintaneja90/MapReduce/tree/master/aws_output_11