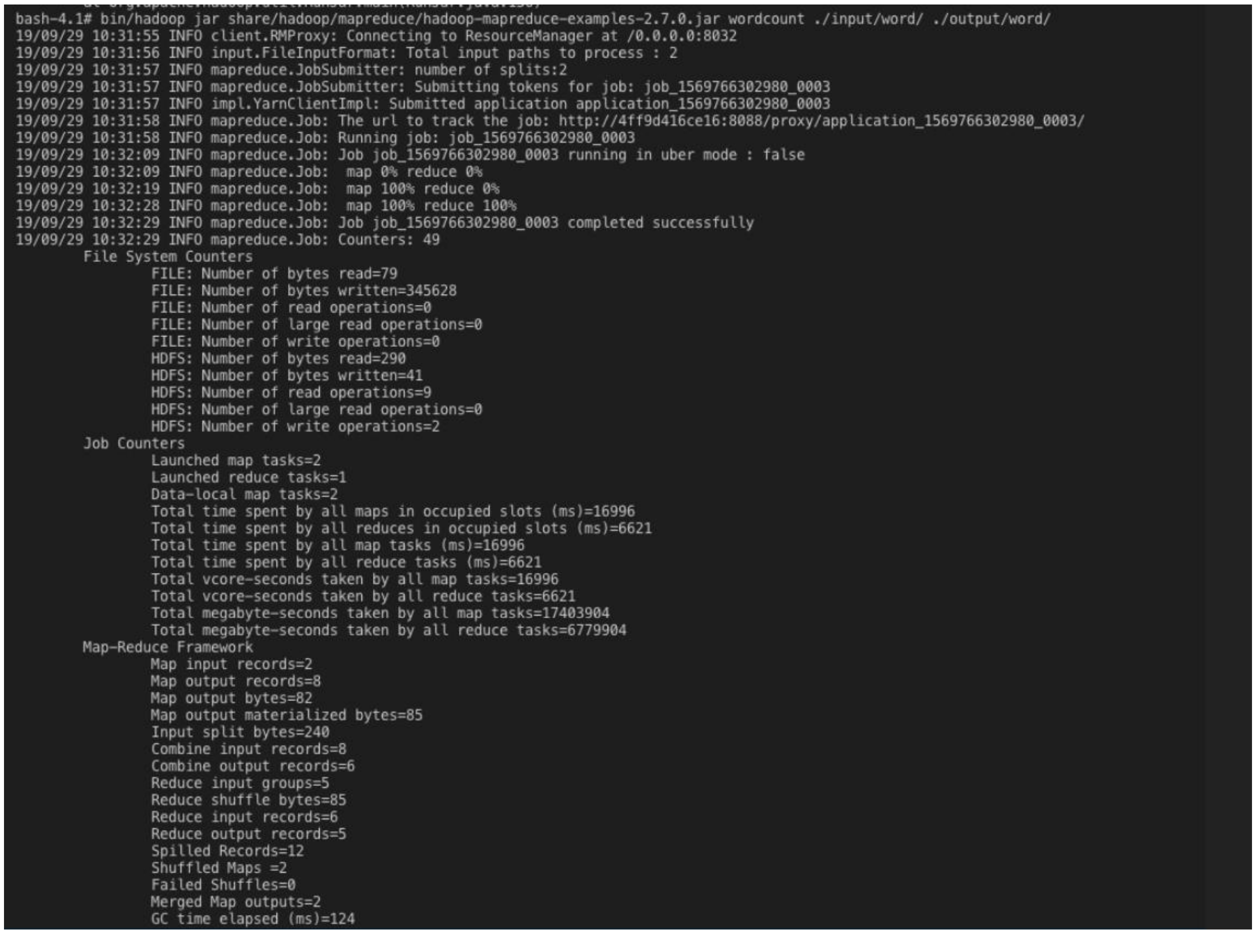
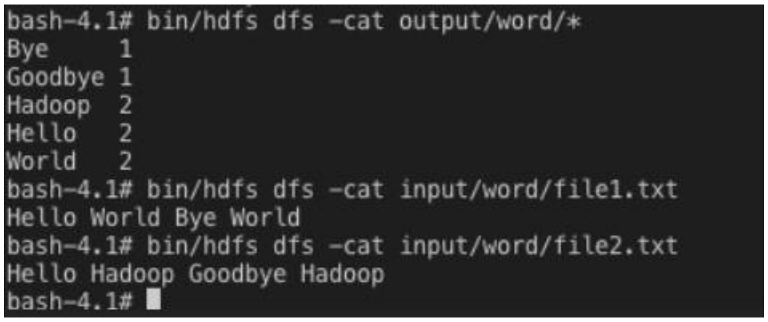
Greeshmika korrapati
U00932594

1.1



```
bash-4.1# bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.7.0.jar wordcount ./input/word/ ./output/word/
19/09/29 10:31:55 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
19/09/29 10:31:56 INFO input.FileInputFormat: Total input paths to process : 2
19/09/29 10:31:57 INFO mapreduce.JobSubmitter: number of splits:2
19/09/29 10:31:57 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1569766302980_0003
19/09/29 10:31:57 INFO impl.YarnClientImpl: Submitted application application_1569766302980_0003
19/09/29 10:31:58 INFO mapreduce.Job: The url to track the job: http://4ff9d416ce16:8088/proxy/application_1569766302980_0003/
19/09/29 10:31:58 INFO mapreduce.Job: Running job: job_1569766302980_0003
19/09/29 10:32:09 INFO mapreduce.Job: Job job_1569766302980_0003 running in uber mode : false
19/09/29 10:32:09 INFO mapreduce.Job:  map 0% reduce 0%
19/09/29 10:32:19 INFO mapreduce.Job:  map 100% reduce 0%
19/09/29 10:32:28 INFO mapreduce.Job:  map 100% reduce 100%
19/09/29 10:32:29 INFO mapreduce.Job: Job job_1569766302980_0003 completed successfully
19/09/29 10:32:29 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=79
                FILE: Number of bytes written=345628
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=290
                HDFS: Number of bytes written=41
                HDFS: Number of read operations=9
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=2
                Launched reduce tasks=1
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=16996
                Total time spent by all reduces in occupied slots (ms)=6621
                Total time spent by all map tasks (ms)=16996
                Total time spent by all reduce tasks (ms)=6621
                Total vcore-seconds taken by all map tasks=16996
                Total vcore-seconds taken by all reduce tasks=6621
                Total megabyte-seconds taken by all map tasks=17403904
                Total megabyte-seconds taken by all reduce tasks=6779904
        Map-Reduce Framework
                Map input records=2
                Map output records=8
                Map output bytes=82
                Map output materialized bytes=85
                Input split bytes=240
                Combine input records=8
                Combine output records=6
                Reduce input groups=5
                Reduce shuffle bytes=85
                Reduce input records=6
                Reduce output records=5
                Spilled Records=12
                Shuffled Maps =2
                Failed Shuffles=0
                Merged Map outputs=2
                GC time elapsed (ms)=124
```

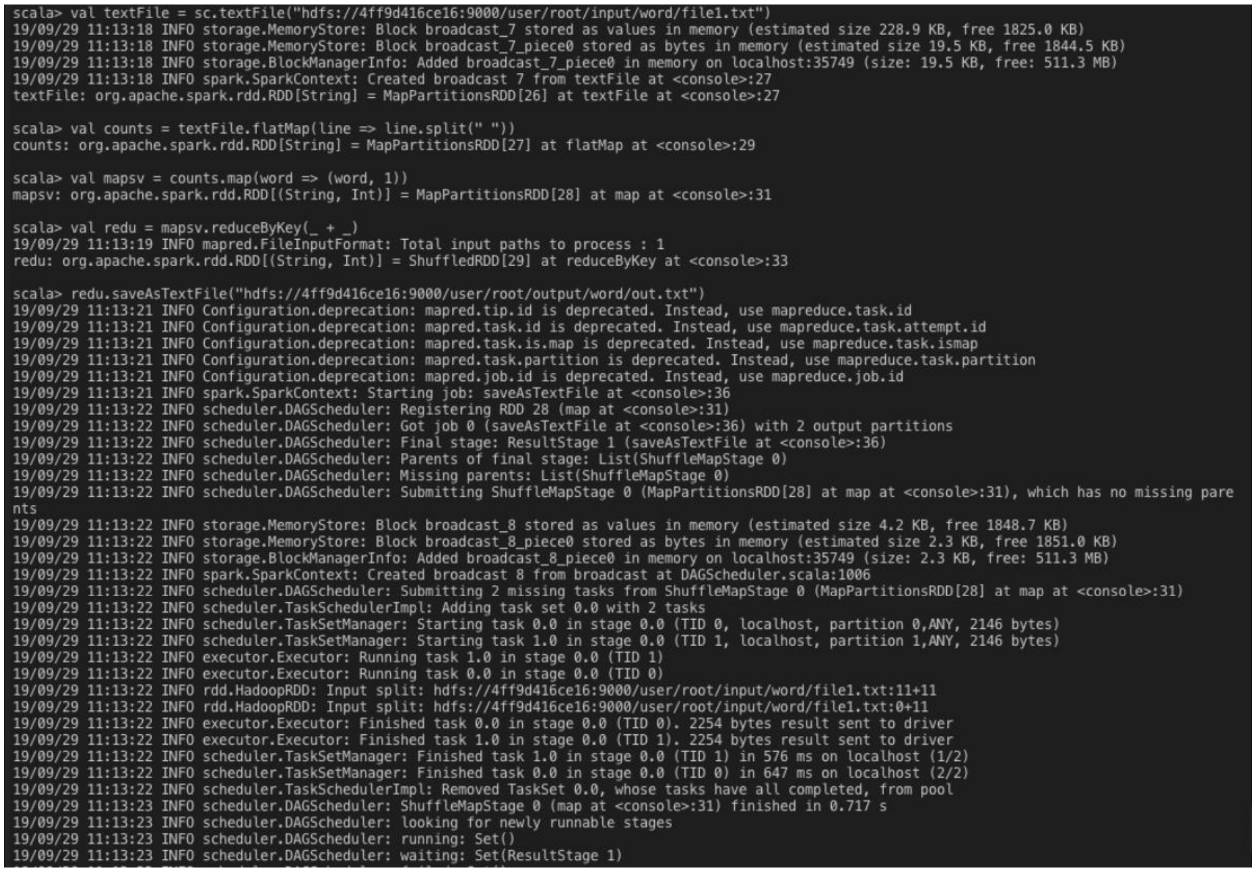
Installed Virtual box and installed cloudera. Created two input files with two sentences and kept them in one directory and then Executed wordcount from share/Hadoop/mapreduce/Hadoop-mapreduce-examples-2.7.0.jar wordcount. Execution was successful creating an output file in hdfs.



```
bash-4.1# bin/hdfs dfs -cat output/word/*
Bye     1
Goodbye 1
Hadoop  2
Hello   2
World   2
bash-4.1# bin/hdfs dfs -cat input/word/file1.txt
Hello World Bye World
bash-4.1# bin/hdfs dfs -cat input/word/file2.txt
Hello Hadoop Goodbye Hadoop
bash-4.1# 
```

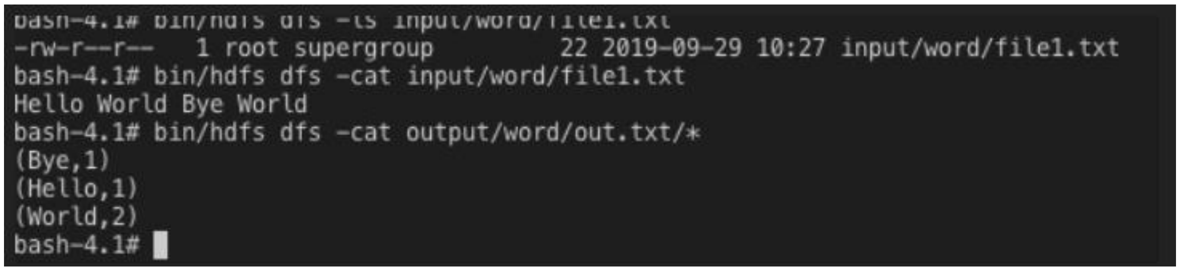
Greeshmika korrapati
U00932594

1.2



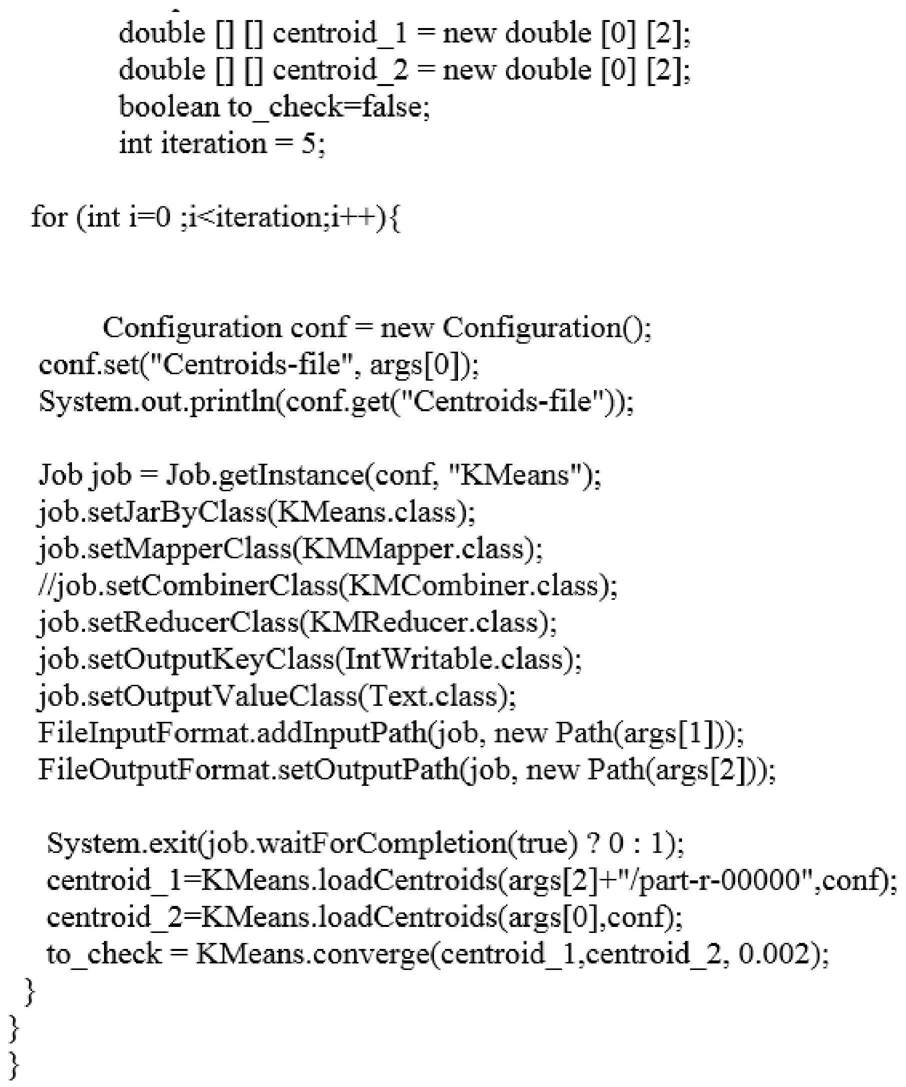
```
scala> val textFile = sc.textFile("hdfs://4ff9d416ce16:9000/user/root/input/word/file1.txt")
19/09/29 11:13:18 INFO storage.MemoryStore: Block broadcast_7 stored as values in memory (estimated size 228.9 KB, free 1825.0 KB)
19/09/29 11:13:18 INFO storage.MemoryStore: Block broadcast_7_piece0 stored as bytes in memory (estimated size 19.5 KB, free 1844.5 KB)
19/09/29 11:13:18 INFO storage.BlockManagerInfo: Added broadcast_7_piece0 in memory on localhost:35749 (size: 19.5 KB, free: 511.3 MB)
19/09/29 11:13:18 INFO spark.SparkContext: Created broadcast 7 from textFile at <console>:27
textFile: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[26] at textFile at <console>:27

scala> val counts = textFile.flatMap(line => line.split(" "))
counts: org.apache.spark.rdd.RDD[String] = MapPartitionsRDD[27] at flatMap at <console>:29

scala> val mapsv = counts.map(word => (word, 1))
mapsv: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[28] at map at <console>:31

scala> val redu = mapsv.reduceByKey(_ + _)
19/09/29 11:13:19 INFO mapred.FileInputFormat: Total input paths to process : 1
redu: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[29] at reduceByKey at <console>:33

scala> redu.saveAsTextFile("hdfs://4ff9d416ce16:9000/user/root/output/word/out.txt")
19/09/29 11:13:21 INFO Configuration.deprecation: mapred.tip.id is deprecated. Instead, use mapreduce.task.id
19/09/29 11:13:21 INFO Configuration.deprecation: mapred.task.id is deprecated. Instead, use mapreduce.task.attempt.id
19/09/29 11:13:21 INFO Configuration.deprecation: mapred.task.is.map is deprecated. Instead, use mapreduce.task.ismap
19/09/29 11:13:21 INFO Configuration.deprecation: mapred.task.partition is deprecated. Instead, use mapreduce.task.partition
19/09/29 11:13:21 INFO Configuration.deprecation: mapred.job.id is deprecated. Instead, use mapreduce.job.id
19/09/29 11:13:21 INFO spark.SparkContext: Starting job: saveAsTextFile at <console>:36
19/09/29 11:13:22 INFO scheduler.DAGScheduler: Registering RDD 28 (map at <console>:31)
19/09/29 11:13:22 INFO scheduler.DAGScheduler: Got job 0 (saveAsTextFile at <console>:36) with 2 output partitions
19/09/29 11:13:22 INFO scheduler.DAGScheduler: Final stage: ResultStage 1 (saveAsTextFile at <console>:36)
19/09/29 11:13:22 INFO scheduler.DAGScheduler: Parents of final stage: List(ShuffleMapStage 0)
19/09/29 11:13:22 INFO scheduler.DAGScheduler: Missing parents: List(ShuffleMapStage 0)
19/09/29 11:13:22 INFO scheduler.DAGScheduler: Submitting ShuffleMapStage 0 (MapPartitionsRDD[28] at map at <console>:31), which has no missing pare
nts
19/09/29 11:13:22 INFO storage.MemoryStore: Block broadcast_8 stored as values in memory (estimated size 4.2 KB, free 1848.7 KB)
19/09/29 11:13:22 INFO storage.MemoryStore: Block broadcast_8_piece0 stored as bytes in memory (estimated size 2.3 KB, free 1851.0 KB)
19/09/29 11:13:22 INFO storage.BlockManagerInfo: Added broadcast_8_piece0 in memory on localhost:35749 (size: 2.3 KB, free: 511.3 MB)
19/09/29 11:13:22 INFO spark.SparkContext: Created broadcast 8 from broadcast at DAGScheduler.scala:1006
19/09/29 11:13:22 INFO scheduler.DAGScheduler: Submitting 2 missing tasks from ShuffleMapStage 0 (MapPartitionsRDD[28] at map at <console>:31)
19/09/29 11:13:22 INFO scheduler.TaskSchedulerImpl: Adding task set 0.0 with 2 tasks
19/09/29 11:13:22 INFO scheduler.TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, localhost, partition 0,ANY, 2146 bytes)
19/09/29 11:13:22 INFO scheduler.TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, localhost, partition 1,ANY, 2146 bytes)
19/09/29 11:13:22 INFO executor.Executor: Running task 1.0 in stage 0.0 (TID 1)
19/09/29 11:13:22 INFO executor.Executor: Running task 0.0 in stage 0.0 (TID 0)
19/09/29 11:13:22 INFO rdd.HadoopRDD: Input split: hdfs://4ff9d416ce16:9000/user/root/input/word/file1.txt:11+11
19/09/29 11:13:22 INFO rdd.HadoopRDD: Input split: hdfs://4ff9d416ce16:9000/user/root/input/word/file1.txt:0+11
19/09/29 11:13:22 INFO executor.Executor: Finished task 0.0 in stage 0.0 (TID 0). 2254 bytes result sent to driver
19/09/29 11:13:22 INFO executor.Executor: Finished task 1.0 in stage 0.0 (TID 1). 2254 bytes result sent to driver
19/09/29 11:13:22 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 576 ms on localhost (1/2)
19/09/29 11:13:22 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 647 ms on localhost (2/2)
19/09/29 11:13:22 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
19/09/29 11:13:23 INFO scheduler.DAGScheduler: ShuffleMapStage 0 (map at <console>:31) finished in 0.717 s
19/09/29 11:13:23 INFO scheduler.DAGScheduler: looking for newly runnable stages
19/09/29 11:13:23 INFO scheduler.DAGScheduler: running: Set()
19/09/29 11:13:23 INFO scheduler.DAGScheduler: waiting: Set(ResultStage 1)
```

Used Spark shell Scala for running word count. Created input for this and used above commands for execution. Execution was successful creating an output file as below.



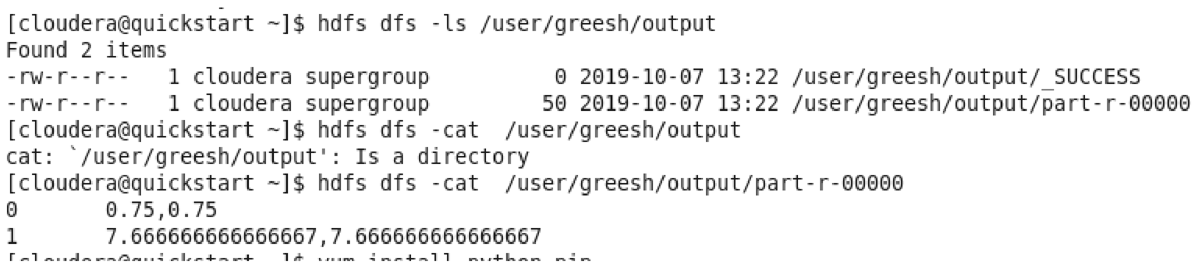
```
bash-4.1# bin/hdfs dfs -ls input/word/file1.txt
-rw-r--r--   1 root supergroup         22 2019-09-29 10:27 input/word/file1.txt
bash-4.1# bin/hdfs dfs -cat input/word/file1.txt
Hello World Bye World
bash-4.1# bin/hdfs dfs -cat output/word/out.txt/*
(Bye,1)
(Hello,1)
(World,2)
bash-4.1# █
```

2.1



```
        double [] [] centroid_1 = new double [0] [2];
        double [] [] centroid_2 = new double [0] [2];
        boolean to_check=false;
        int iteration = 5;

    for (int i=0 ;i<iteration;i++){


          Configuration conf = new Configuration();
     conf.set("Centroids-file", args[0]);
     System.out.println(conf.get("Centroids-file"));

     Job job = Job.getInstance(conf, "KMeans");
     job.setJarByClass(KMeans.class);
     job.setMapperClass(KMMapper.class);
     //job.setCombinerClass(KMCombiner.class);
     job.setReducerClass(KMReducer.class);
     job.setOutputKeyClass(IntWritable.class);
     job.setOutputValueClass(Text.class);
     FileInputFormat.addInputPath(job, new Path(args[1]));
     FileOutputFormat.setOutputPath(job, new Path(args[2]));

      System.exit(job.waitForCompletion(true) ? 0 : 1);
      centroid_1=KMeans.loadCentroids(args[2]+"/part-r-00000",conf);
      centroid_2=KMeans.loadCentroids(args[0],conf);
      to_check = KMeans.converge(centroid_1,centroid_2, 0.002);
    }
  }
  }
```

Used For loop for iteration, adding input path of file and setting output path for file. After that updating values of centroid with output and comparing convergence with previous centroids such that it stops if converge() is true. Yes I have successfully implemented iteration for given source code



```
[cloudera@quickstart ~]$ hdfs dfs -ls /user/greesh/output
Found 2 items
-rw-r--r--   1 cloudera supergroup          0 2019-10-07 13:22 /user/greesh/output/_SUCCESS
-rw-r--r--   1 cloudera supergroup         50 2019-10-07 13:22 /user/greesh/output/part-r-00000
[cloudera@quickstart ~]$ hdfs dfs -cat  /user/greesh/output
cat: `/user/greesh/output': Is a directory
[cloudera@quickstart ~]$ hdfs dfs -cat  /user/greesh/output/part-r-00000
0       0.75,0.75
1       7.666666666666667,7.666666666666667
[cloudera@quickstart ~]$ yum install python pip
```

2.2

Combiner class encapsulates the output from map with same key and output of combiner class will sent to actual reduce class as input. Several repeated keys are genrated by maps this basically used to sum-up these large output records from map to less records. It does not have interface instead it uses reducer interface and reduce method in combiner is called on each map output key. It's input and output key-values should match with reducer class. Combiner increases overall performance of Mapreduce, it also decreases time for data transmission. The big O representation of data passed to reducer phase without combiner is O(NRKM) and with combiner is O(NRK/M)

2.3

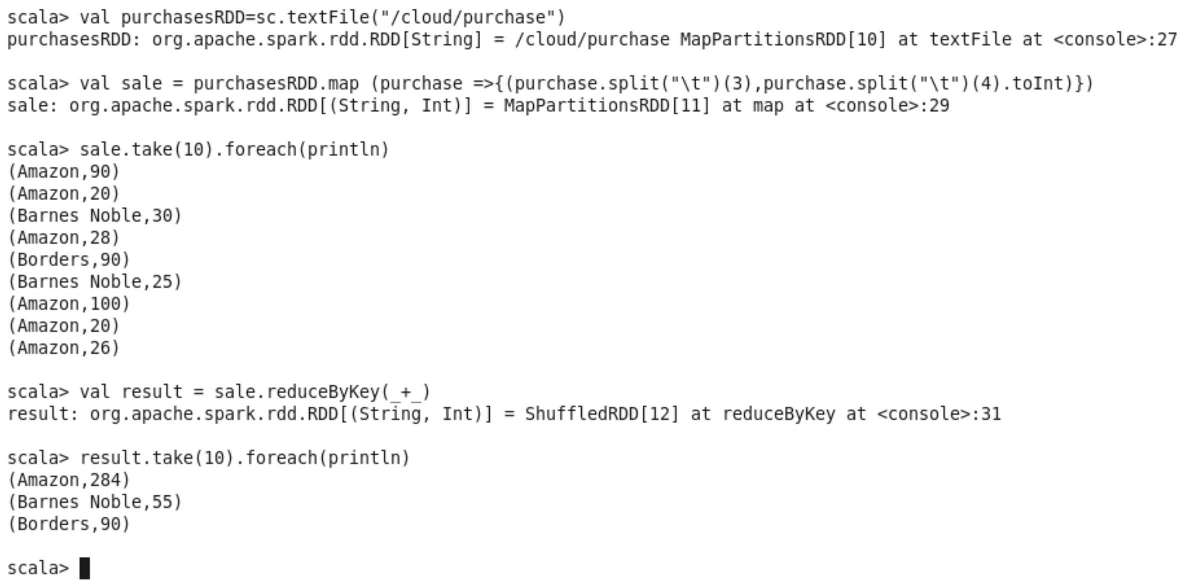
```
def mapper(a,b,centroid_array):

    min_distance = []
    min_distance = np.append(euclidian_dist(centroid0[0],
centroid0[1], a, b)) //distance between two centroids and data
points are caluculated and stored in array
    print(np.argmin(min_distance), "\t", a, b) // min distance of
both is calculated and label is assigned


def read_centroids(fname):
    data = None
    with open(fname, "r") as fd: // opens file and reads data
        data = fd.read()

    return data


def split_centroids(centroids_raw):
    centroids = centroids_raw.split("\r\n")    // splits by line
    centroid0 = centroids[0].split("\t")[1].split(",") // splits
line into words by delimiter , and puts first element into
centroid 0
    centroid1 = centroids[1].split("\t")[1].split(",")
// splits line into words by delimiter , and puts second element
into centroid 1

    return centroid0, centroid1


def euclidian_dist(centroid_a, centroid_b, a, b):
    centroid_a = float(centroid_a)
    centroid0_b = float(centroid_b)
    a = float(a)
    b = float(b)
    return ((centroid_a - a) ** 2 + (centroid_b - b) ** 2) ** 0.5
//distance between centroids and data is caluculated and returned
```

Project1 Greeshmika korrapati
U00932594

I have two methods split_centroids and read_centroids for reading and splitting centroids.I am getting data file as system standard input and splitting lines and storing values in x,y.I have Euclidian_distance calucuting distance between centroids and data points from input file. After calculating distance finding out minimum distance using numpy.argmin and assigning label for the cluster.

```
if __name__ == "__main__":
    for line in sys.stdin:
        data1, data2 = line.split("\t")[1].split("\n")
[0].split(",") // data file is splitted by lines and words and
first word assigned to data1 and second word to data2
        centroid0, centroid1 =
split_centroids(read_centroids(sys.argv[1]))
//both split_centroids and read_centroids functions are called
        centroid_array=[]
        centroid_array=np.append(centroid0,centroid1) resultant
centroid0,centroid1 is stored in centroid_array
        mapper(data1,data2,centroid_array) // mapper function is
called
```

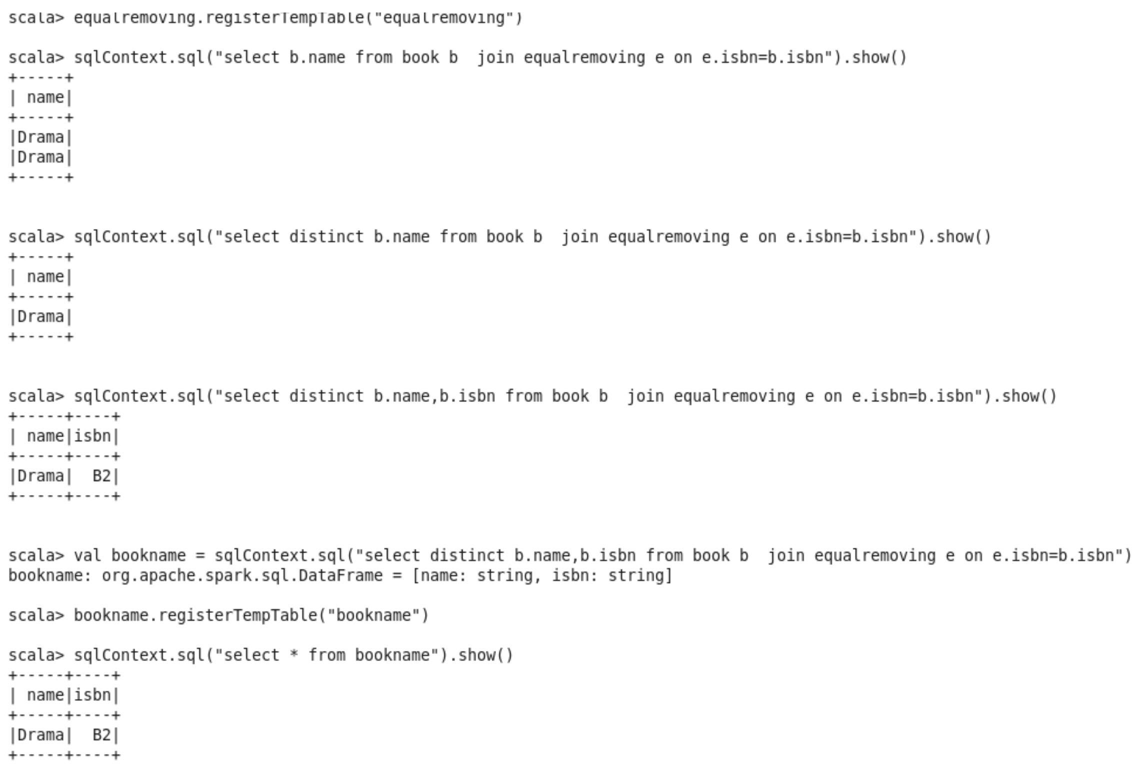
3.1



```
scala> val purchasesRDD=sc.textFile("/cloud/purchase")
purchasesRDD: org.apache.spark.rdd.RDD[String] = /cloud/purchase MapPartitionsRDD[10] at textFile at <console>:27

scala> val sale = purchasesRDD.map (purchase =>{(purchase.split("\t")(3),purchase.split("\t")(4).toInt)})
sale: org.apache.spark.rdd.RDD[(String, Int)] = MapPartitionsRDD[11] at map at <console>:29

scala> sale.take(10).foreach(println)
(Amazon,90)
(Amazon,20)
(Barnes Noble,30)
(Amazon,28)
(Borders,90)
(Barnes Noble,25)
(Amazon,100)
(Amazon,20)
(Amazon,26)

scala> val result = sale.reduceByKey(_+_)
result: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[12] at reduceByKey at <console>:31

scala> result.take(10).foreach(println)
(Amazon,284)
(Barnes Noble,55)
(Borders,90)

scala> 
```

The total sales of Amazon is 284, Barnes Noble is 55, Borders is 90.Intially loaded file into purchaseRDD then splited by \t (since dataset is seperated by \t) and extracted seller and sales column into sale and then used reduceByKey to get total sales of each seller".

3.2



```
scala> equalremoving.registerTempTable("equalremoving")

scala> sqlContext.sql("select b.name from book b  join equalremoving e on e.isbn=b.isbn").show()
+-----+
| name|
+-----+
|Drama|
|Drama|
+-----+


scala> sqlContext.sql("select distinct b.name from book b  join equalremoving e on e.isbn=b.isbn").show()
+-----+
| name|
+-----+
|Drama|
+-----+


scala> sqlContext.sql("select distinct b.name,b.isbn from book b  join equalremoving e on e.isbn=b.isbn").show()
+-----+----+
| name|isbn|
+-----+----+
|Drama|  B2|
+-----+----+


scala> val bookname = sqlContext.sql("select distinct b.name,b.isbn from book b  join equalremoving e on e.isbn=b.isbn")
bookname: org.apache.spark.sql.DataFrame = [name: string, isbn: string]

scala> bookname.registerTempTable("bookname")

scala> sqlContext.sql("select * from bookname").show()
+-----+----+
| name|isbn|
+-----+----+
|Drama|  B2|
+-----+----+
```

Drama is the name of book that amazon sells for lowest price compared to all other sellers.