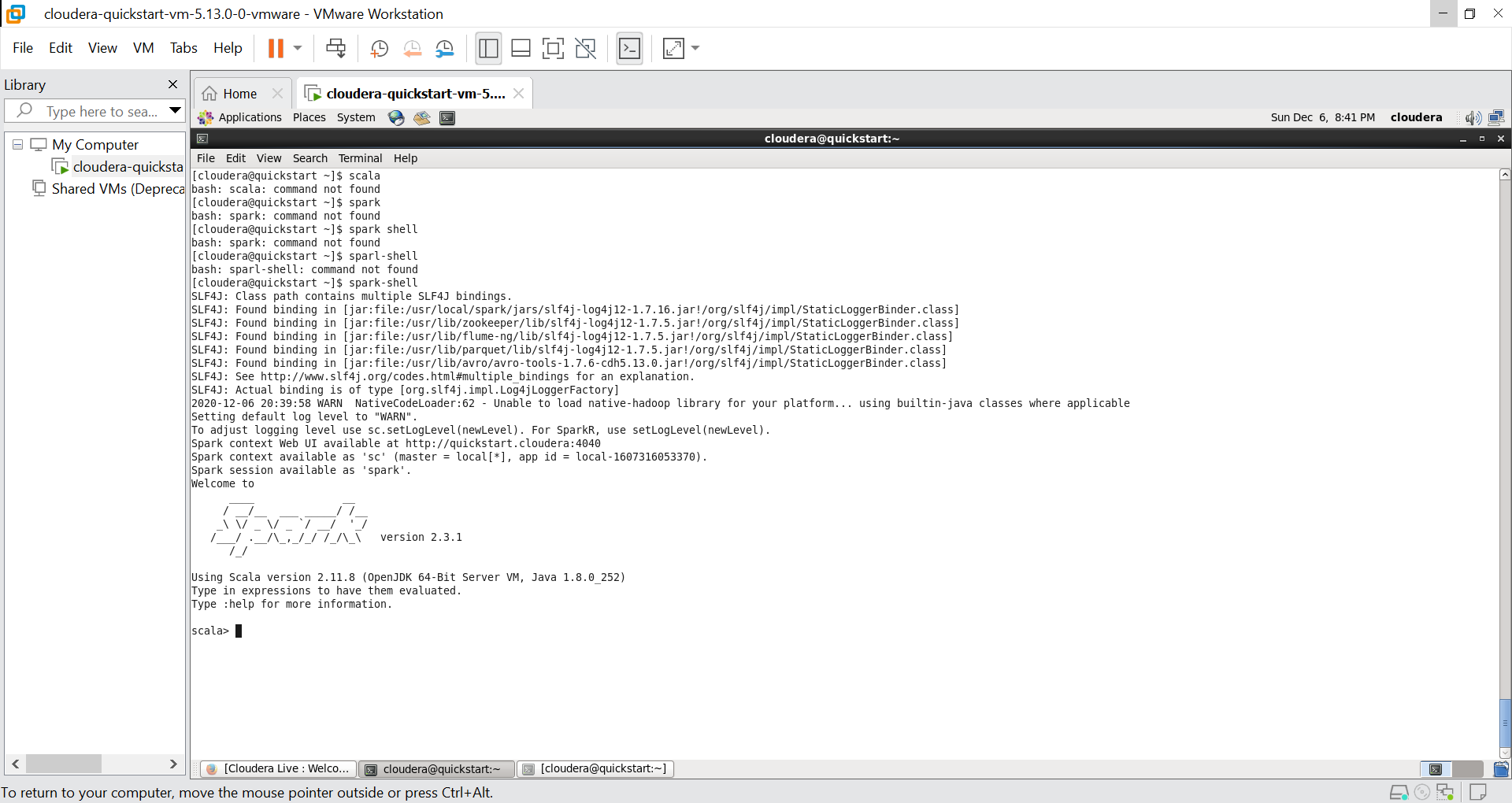
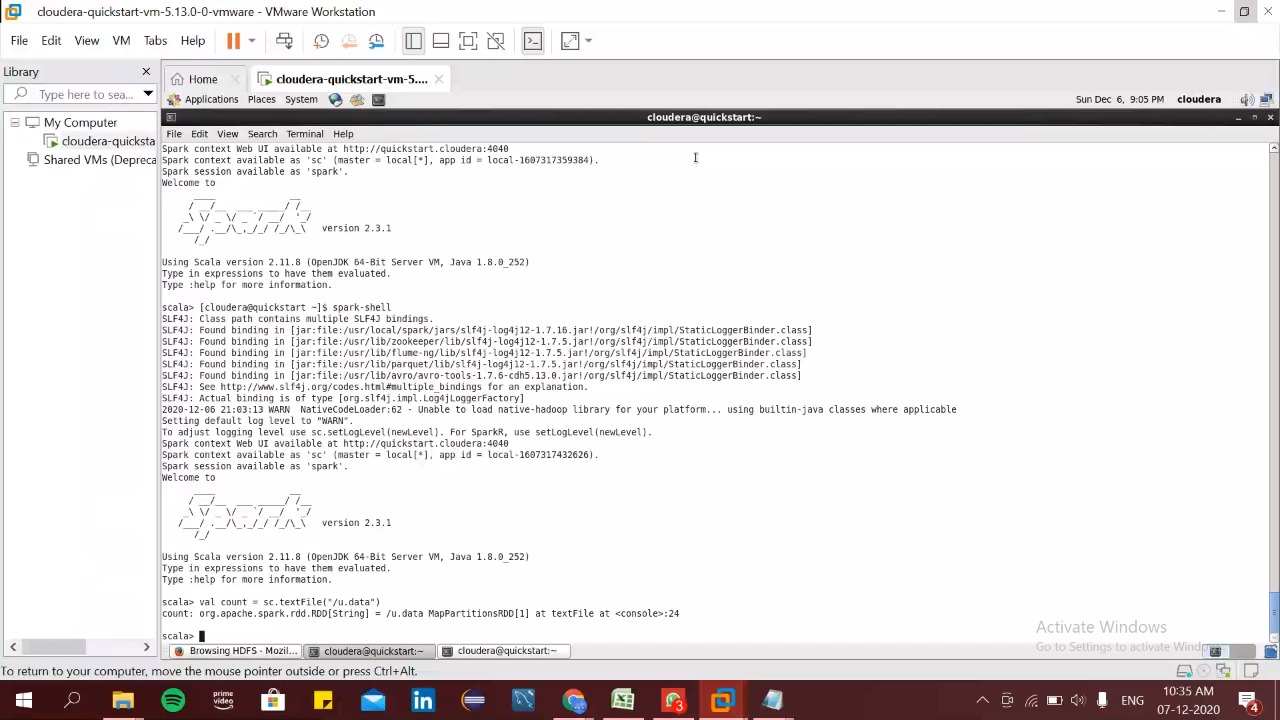
**KO: analyze data using RDDS .**

Enter to the spark shell:



Use the command “spark-shell” to enter to the spark shell

Load the data to the RDD:

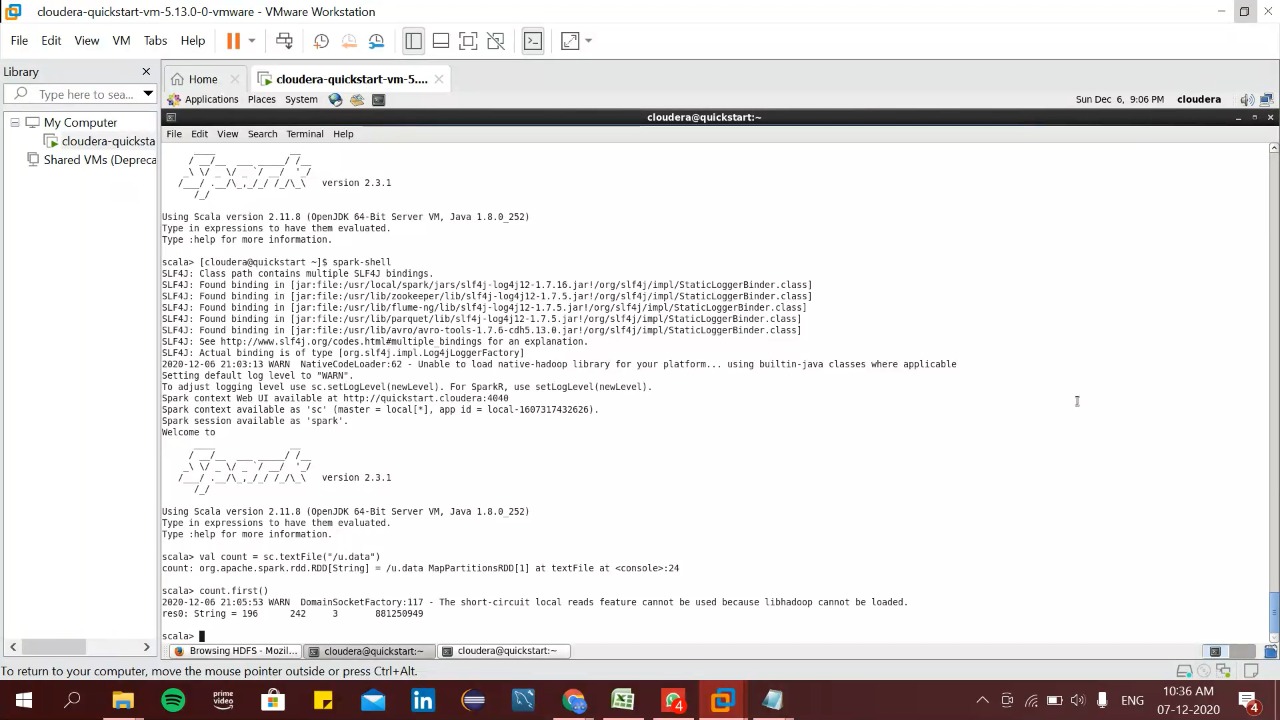


Load the data to the RDD using the command:

val count = sc.textFile("/u.data")

make sure to upload the file in the hdfs.

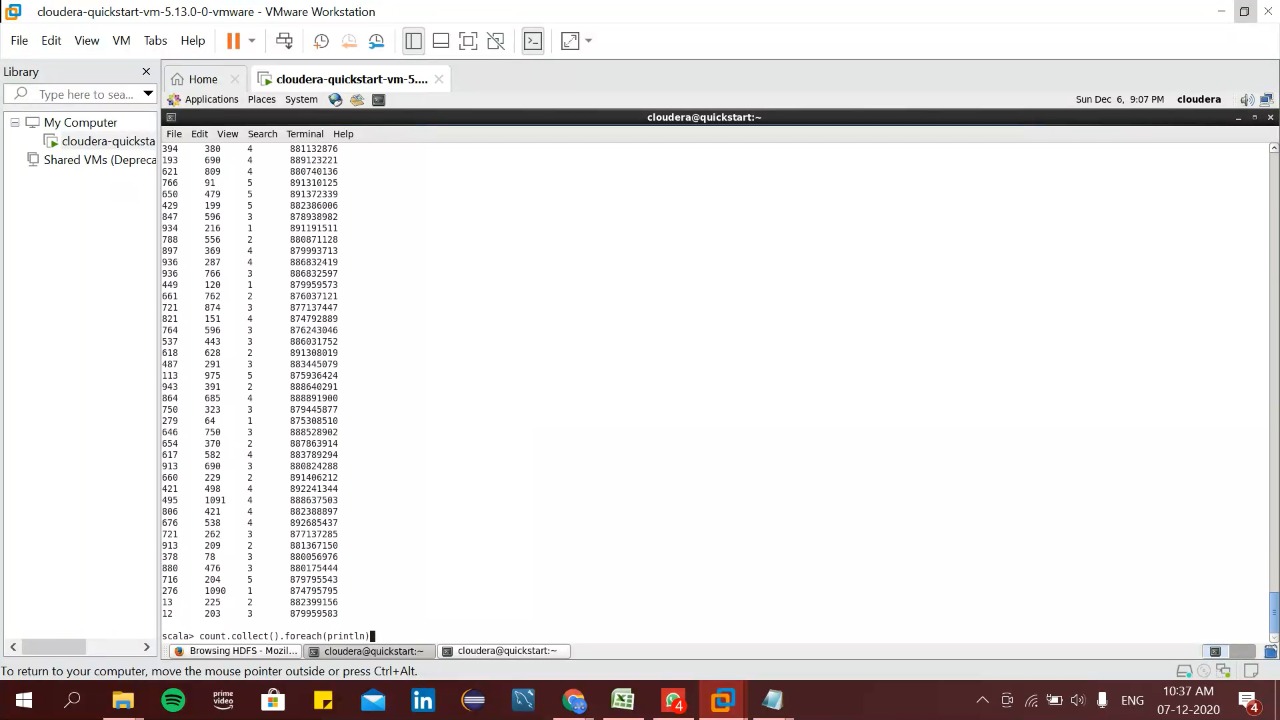
Print the first line of the RDD:



Here we can see the first record of the dataset. This can be fetched using the command:

count.first()

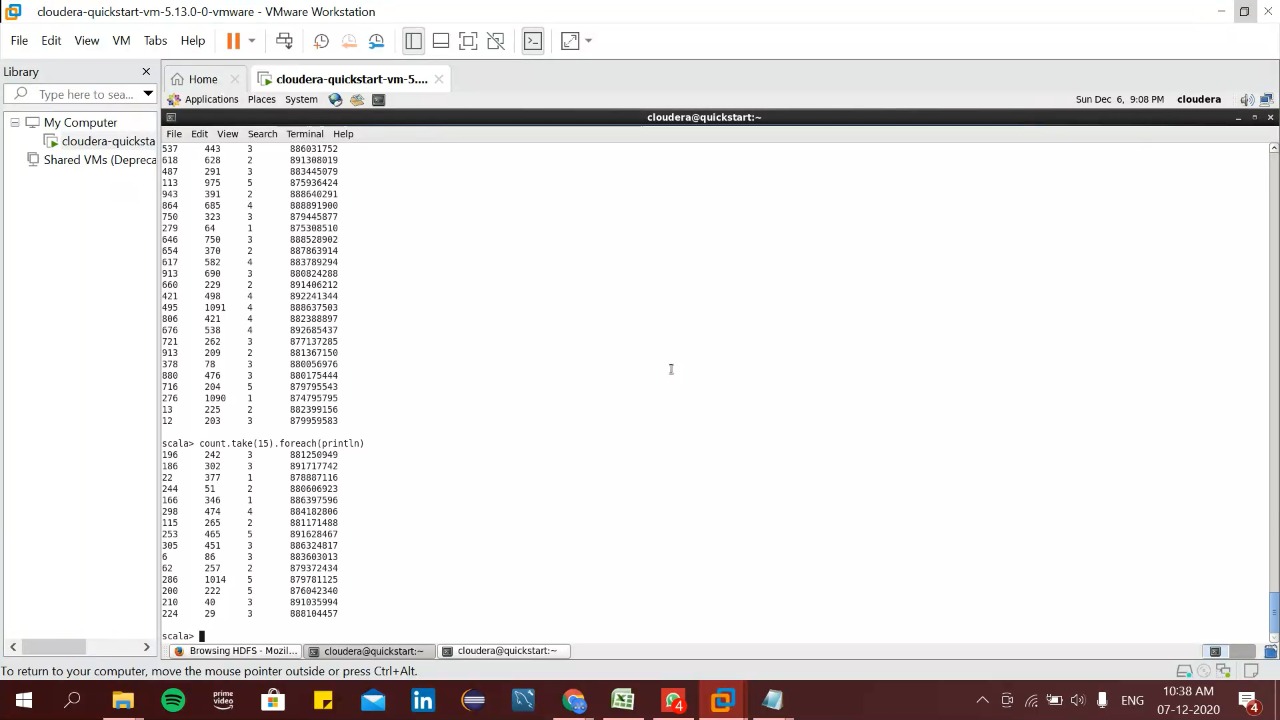
Printing each and every line of the csv



Here we have printed each and every line of the u.data file using the command:

count.collect().foreach(println)

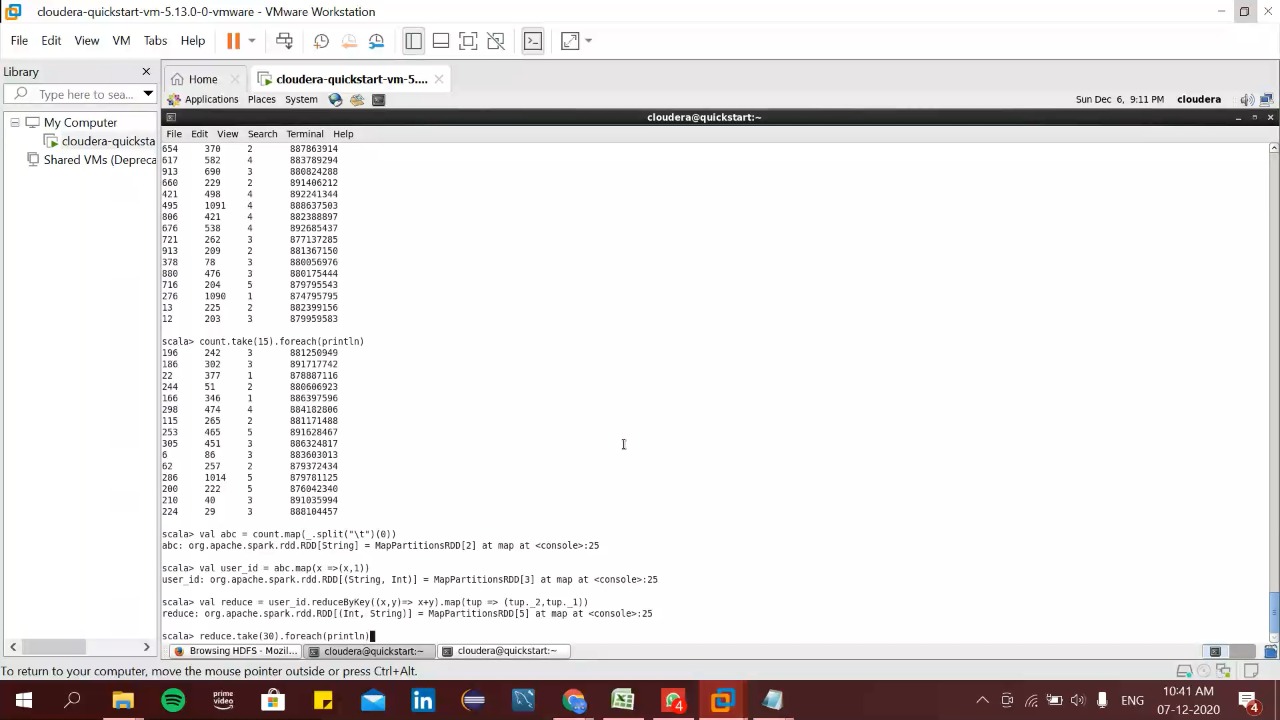
Printing the first 15 lines of the dataset:



For printing the first 15 records of the dataset we use the command:

count.take(15).foreach(println)

Counting the data based on the user\_Id



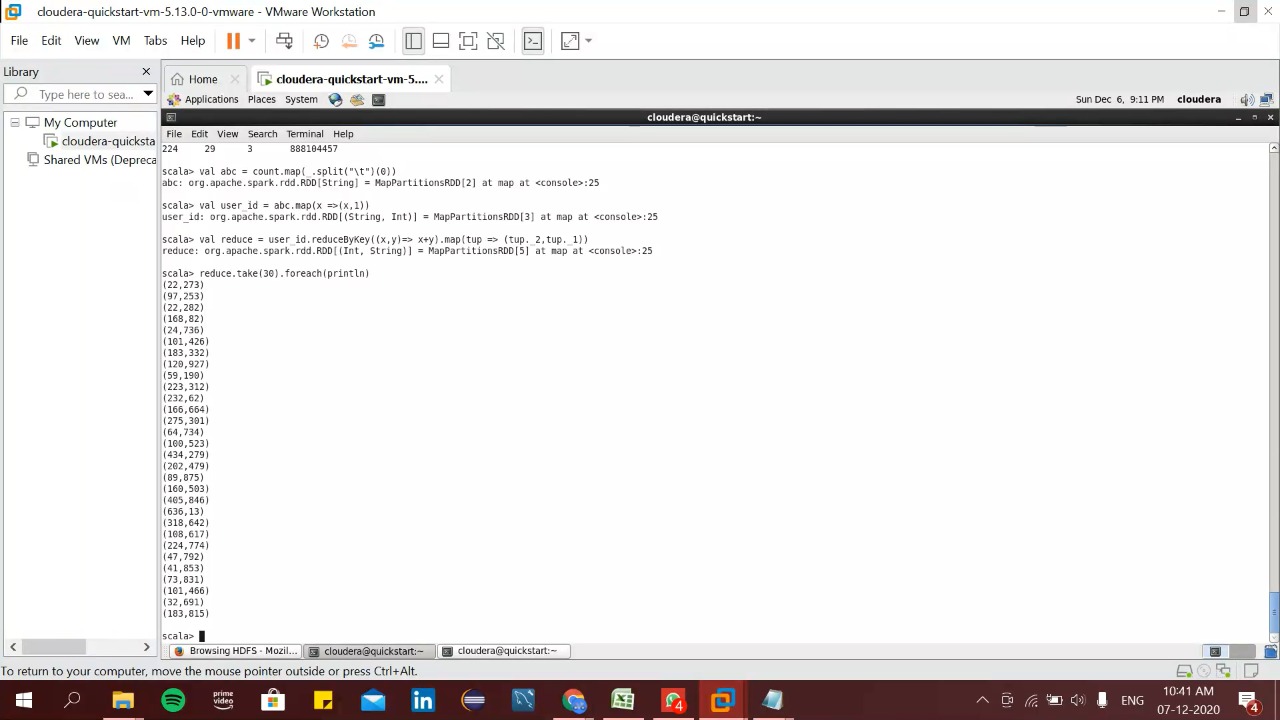
Here we divided the dataset according to the user id and count the user id using the command:

val abc = count.map(\_.split("\t")(0))

val user\_id = abc.map(x =>(x,1))

val reduce = user\_id.reduceByKey((x,y)=> x+y).map(tup => (tup.\_2,tup.\_1))

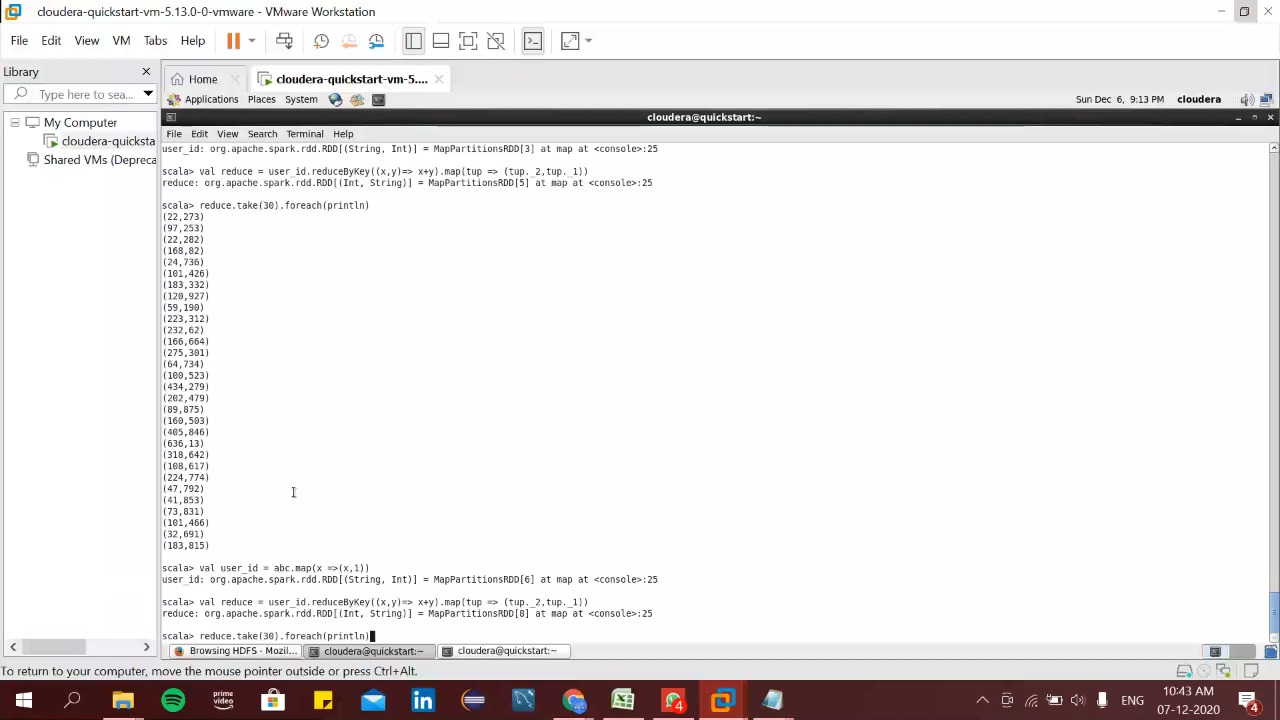
Printing the user\_id data using group by:



Here we can see the output of the group by done on user Id and get its total count by using the command:

reduce.take(30).foreach(println)

Count the total number of movie id grouped by movie id



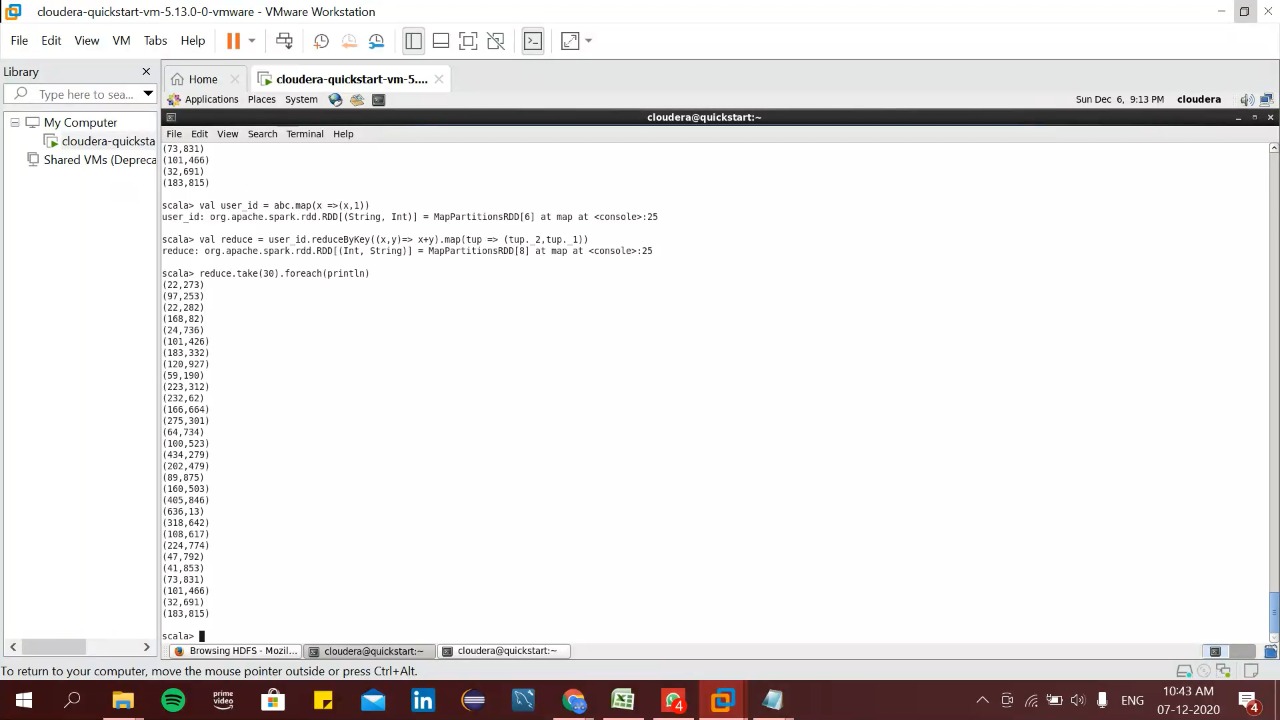
Here we do a operation to get the count of total number of movie id grouped by user id.

val user\_id = abc.map(x =>(x,1))

val reduce = user\_id.reduceByKey((x,y)=> x+y).map(tup => (tup.\_2,tup.\_1))

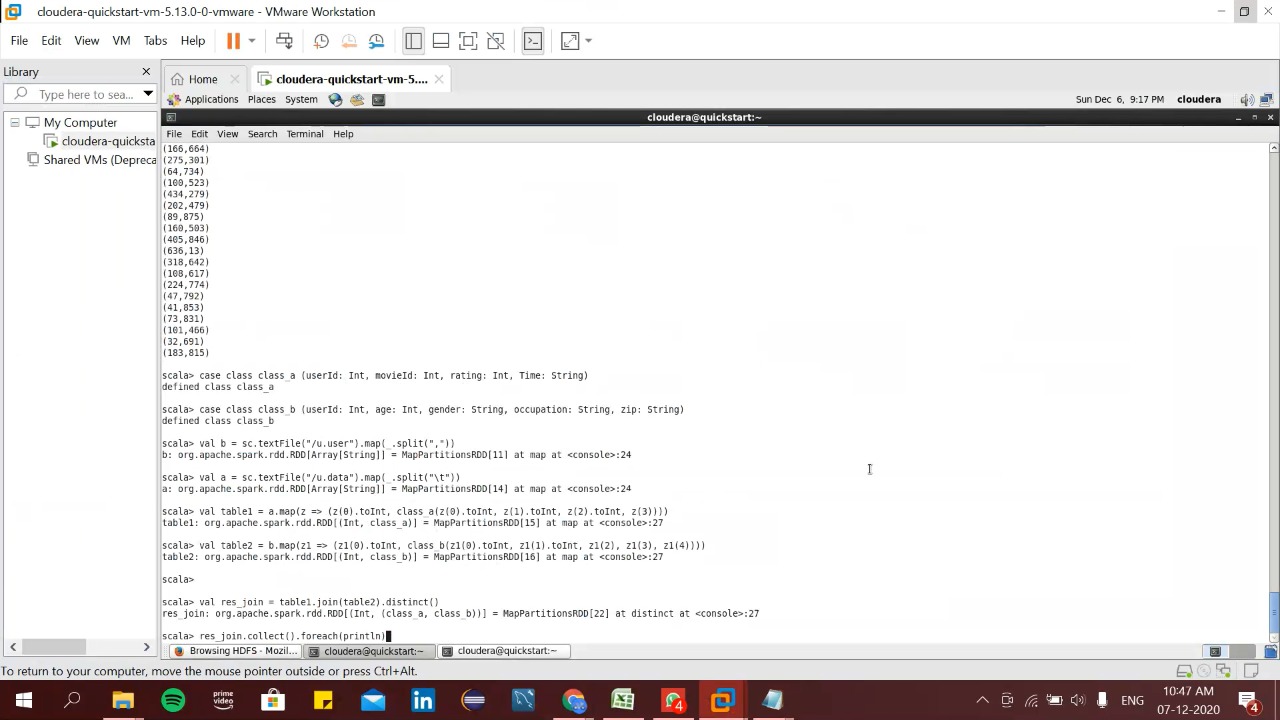
reduce.take(30).foreach(println)

Print the dataset grouped by movie id:



We get the output of the movie id and its count based on user id

Joining of two RDDs:



Here joining of 2 RDDs is done using the command:

case class class\_a (userId: Int, movieId: Int, rating: Int, Time: String)

case class class\_b (userId: Int, age: Int, gender: String, occupation: String, zip: String)

val b = sc.textFile("/u.user").map(\_.split(","))

val a = sc.textFile("/u.data").map(\_.split("\t"))

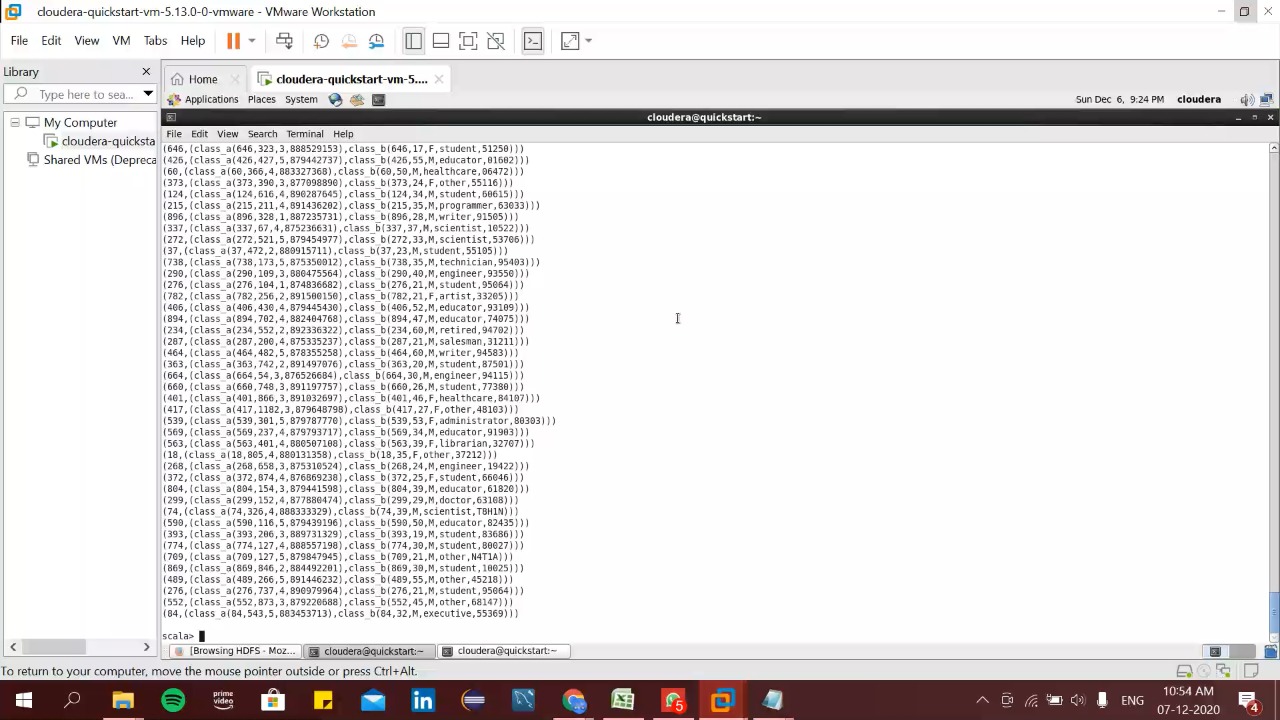
val table1 = a.map(z => (z(0).toInt, class\_a(z(0).toInt, z(1).toInt, z(2).toInt, z(3))))

val table2 = b.map(z1 => (z1(0).toInt, class\_b(z1(0).toInt, z1(1).toInt, z1(2), z1(3), z1(4))))

val res\_join = table1.join(table2).distinct()

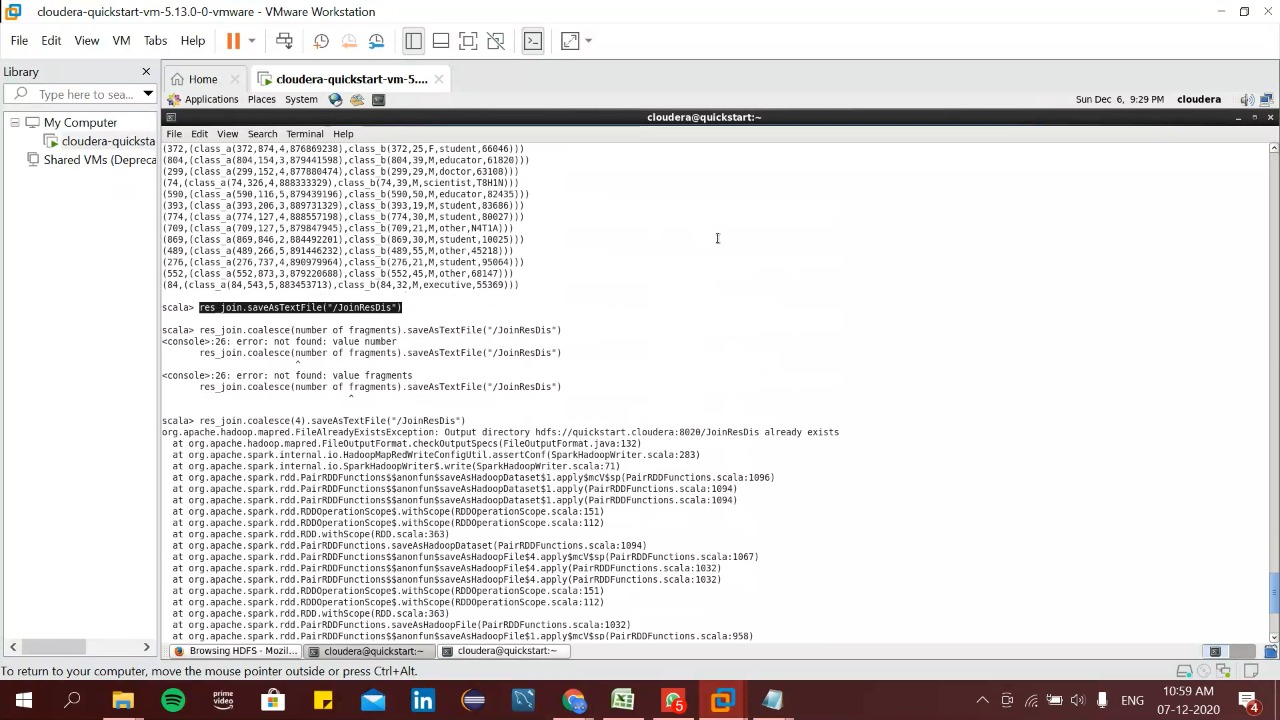
res\_join.collect().foreach(println)

Printing the result of the joining of two RDDs:



We can see the result of joining the two RDDs.

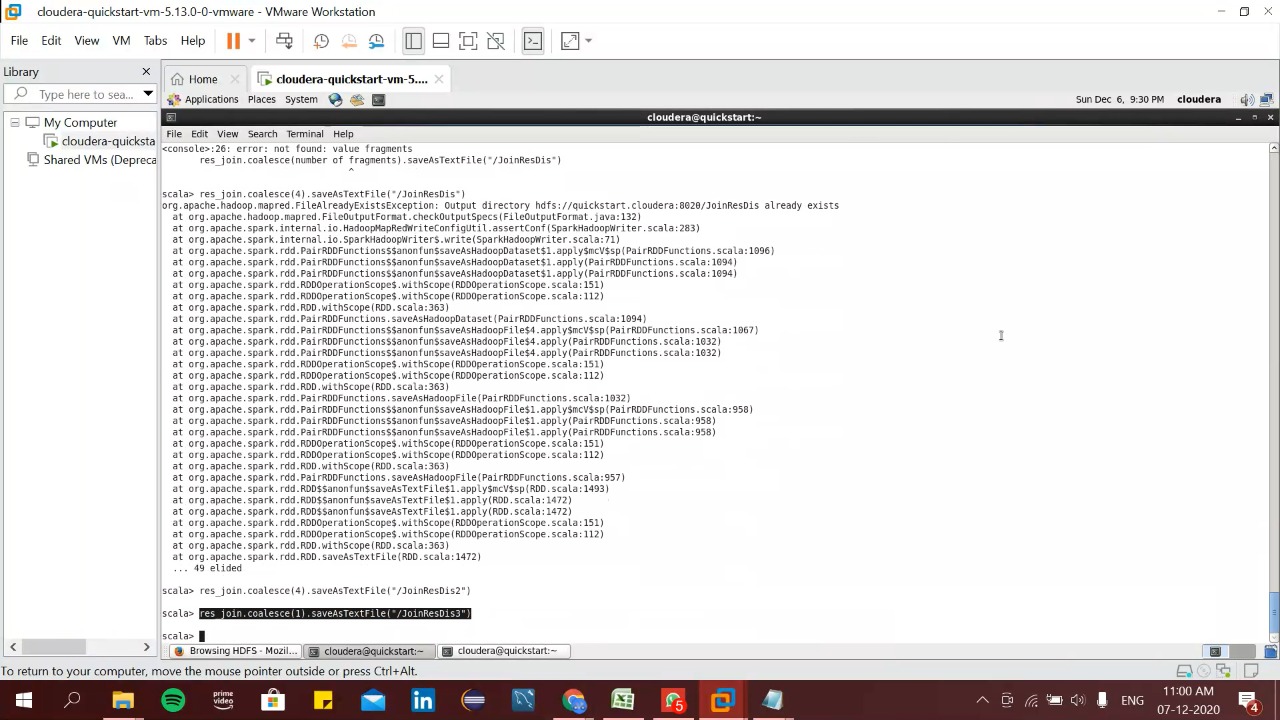
Storing the output of the joining of two RDDs in the form of textfile in hdfs:



Here we stored the result of joining 2 RDDs as a textfile in the hdfs using:

res\_join.saveAsTextFile("/JoinResDis")

Performance tuning in RDD:



Here we have done a coalesce of the RDDs to save the output of the joins in desired number of fragments.