Distributed data analytics (2016-2017)
Information Systems and Machine Learning Lab
University of Hildesheim

Exercise Sheet 5

Mario Rodríguez Ruiz

May 19, 2017

Contents

List of Figures

List of Tables

1 Explain your system

	Home System
Machine	Asus Notebook ROG G60Jx
Operating System	Windows 10 Pro 64-bit
CPU	Intel Core i7 720QM @1.60GHz
Number of cores	4
Number of threads	8
RAM	16GB @665MHz (9-9-9-24)
Programming language version Python	v3.6.1:69c0db5 64 bit
Programming language version Java	v1.8
Hadoop	v2.6.0

Table 1.1: My system

2 Analysis of Airport efficiency with Map Reduce

2.1 Computing the maximum, minimum, and average departure delay for each airport

The Figure 2.1 shows a small portion of the result obtained. The complete file is in the folder "**Exercise2**" and its name is "**ex2_output.csv**"

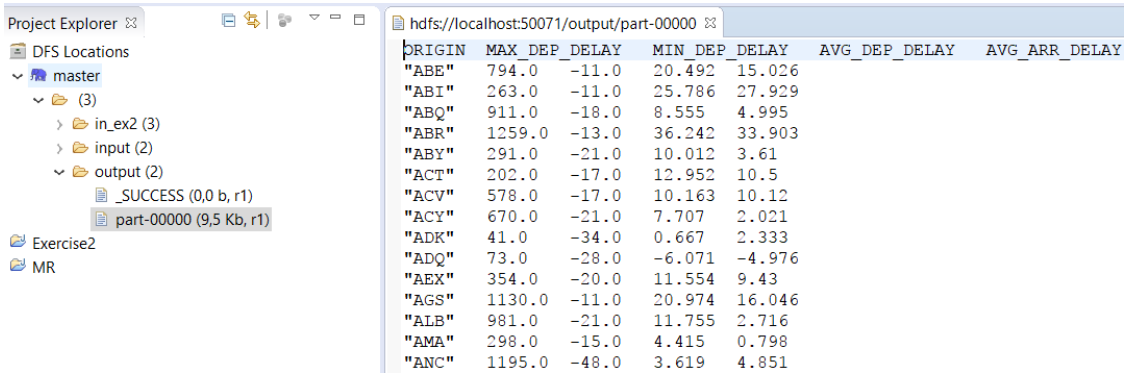

Figure 2.1: Maximum, minimum, and average departure delay for each airport

2.2 Computing a ranking list that contains top 10 airports by their average arrival delay

The Figure 2.2 shows a small portion of the result obtained. The complete file is in the folder "**Exercise2**" and its name is "**ex2_output.csv**"

At the end of the file named before two lists are printed: the first is top 10 airports by their average arrival delay and the second of the worst 10.

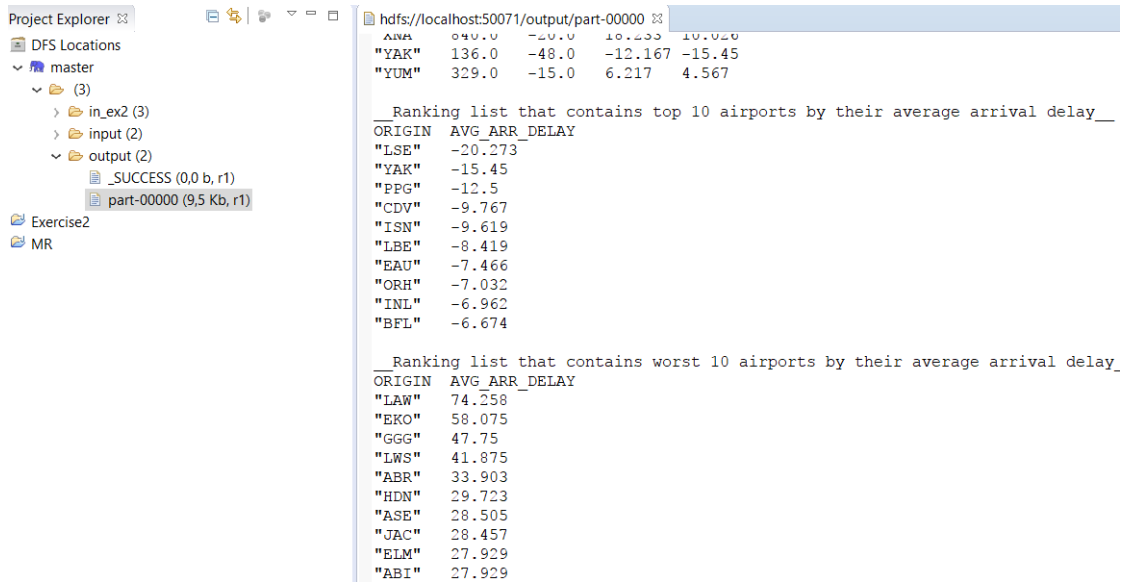

Figure 2.2: ranking list

2.3 What are your mapper.py and reduce.py solutions

2.3.1 mapper.py solution

```
1  import sys
2
3  for line in sys.stdin:
4    line = line.strip()
5    line = line.split(",")
6
7    origin = line[3]
8    dep_delay = line[6]
9    arr_delay = line[8]
10
11   print ("{}\t{}\t{}".format(origin, dep_delay, arr_delay))
```

2.3.2 reduce.py solution

```
1  import sys
2
3  origin_dep_delay = {}
4  origin_arr_delay = {}
5  max_dep_delay = {}
6  min_dep_delay = {}
7
8  # Classification of airports
9  for line in sys.stdin:
10   line = line.strip()
11   origin, dep_delay, arr_delay = line.split('\t')
```

```
12    dep_delay = float(dep_delay)
13    arr_delay = float(arr_delay)
14
15    # If already exists this origin
16    if origin in origin_dep_delay:
17      # Add new items to the lists
18      origin_dep_delay[origin].append(dep_delay)
19      origin_arr_delay[origin].append(arr_delay)
20      # Need to update the maximum?
21      if dep_delay > max_dep_delay[origin]:
22        max_dep_delay[origin] = dep_delay
23      # Need to update the minimum?
24      if dep_delay < min_dep_delay[origin]:
25        min_dep_delay[origin] = dep_delay
26      # If this origin does not yet exist
27    else:
28      # Create two new list
29      origin_dep_delay[origin] = []
30      origin_arr_delay[origin] = []
31      # Add the first items to the lists
32      origin_dep_delay[origin].append(dep_delay)
33      origin_arr_delay[origin].append(arr_delay)
34      # Initial value of the maximum and minimum
35      max_dep_delay[origin] = dep_delay
36      min_dep_delay[origin] = dep_delay
37
38  # Airports reduce
39  list_arr_delay = [] # List created to get a specific ranking
40  print("ORIGIN\tMAX_DEP_DELAY\tMIN_DEP_DELAY\tAVG_DEP_DELAY\
        tAVG_ARR_DELAY")
41  for origin in origin_dep_delay.keys():
42    # Calculation of averages
43    avg_dep_delay = sum(origin_dep_delay[origin])*1.0 / len(
          origin_dep_delay[origin])
44    avg_arr_delay = sum(origin_arr_delay[origin])*1.0 / len(
          origin_arr_delay[origin])
45    # Add new items to the list
46    list_arr_delay.append([origin, avg_arr_delay])
47    print ("{}\t{}\t{}\t{}\t{}".format(origin, max_dep_delay[
          origin], min_dep_delay[origin], round(avg_dep_delay, 3),
          round(avg_arr_delay, 3)))
48
49  # List top 10 airports
50  list_arr_delay.sort(key=lambda arr: arr[1])
51  print("\n__Ranking list that contains top 10 airports by their
        average arrival delay__")
52  print("ORIGIN\tAVG_ARR_DELAY")
53  for i in range(10):
```

```
54    print("{}\t{}".format(list_arr_delay[i][0], round(
          list_arr_delay[i][1], 3)))
55
56  # List of the 10 worst airports
57  print("\n__Ranking list that contains worst 10 airports by
        their average arrival delay__")
58  print("ORIGIN\tAVG_ARR_DELAY")
59  for i in range(len(list_arr_delay)-1, len(list_arr_delay)-11,
        -1):
60    print("{}\t{}".format(list_arr_delay[i][0], round(
          list_arr_delay[i][1], 3)))
```

2.4 Describe step-by-step how you apply them and the outputs during this procedure

First have to have Hadoop installed, procedure do in Windows 10 with the help of the document "**Install_Hadoop-2.6.0_on_Windows10.pdf**".
Once installed, start the Hadoop run using the "**start-all.cmd**" script that is located in "**[...]\hadoop-2.6.0\sbin**".

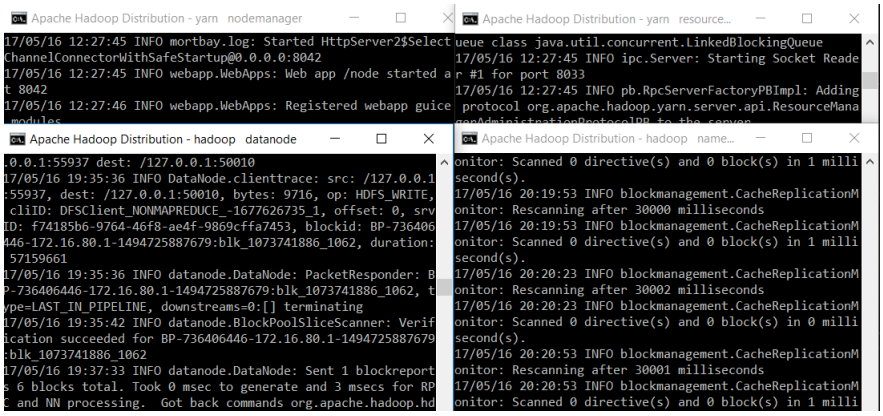

Figure 2.3: Run start-all.cmd

Now have to upload to the "server" the necessary input files for the program, in my case the file "**input0.csv**" in which are the data downloaded from "*Bureau of Transportation Statistics*" homepage. Figure 2.4 shows the command that was used for it and the contents of the folder after the upload.

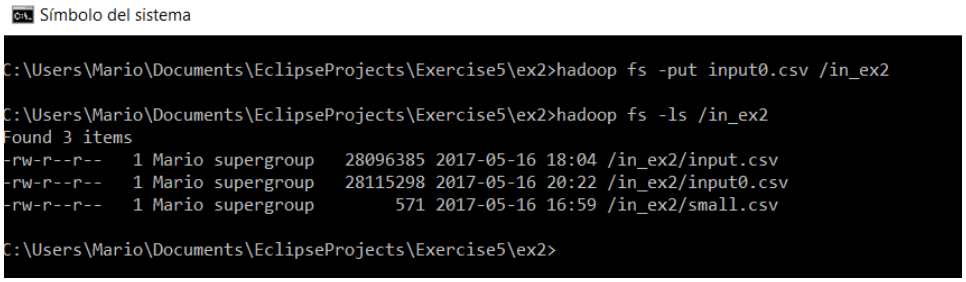

Figure 2.4: Contents HDFS folders after the upload

For a more convenient run of the program, as well as better visualization of the results I have used Eclipse. To do this I have created a new configuration in "*External Tools*" that can be seen in Figure 2.5. Once all this is configured, simply execute and visualize the results.

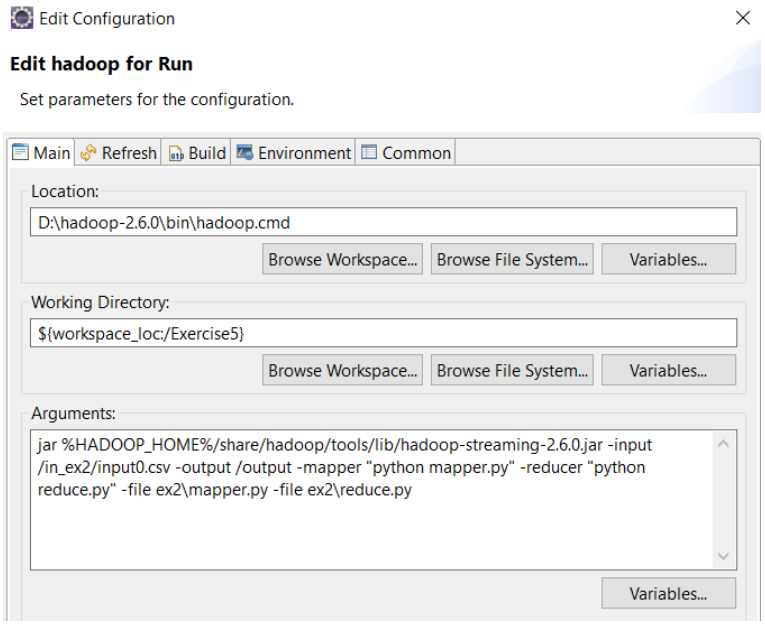

Figure 2.5: External Tools configurations in Eclipse

To view the results just go to "*DFS Locations*" (here all folders and files hosted in HDFS, Figure 2.6) and then to the folder that we indicated in the previous command to save the results (called "**output** " in my case). There are now two new files: "**part-00000**" is the file containing the result of the program execution.

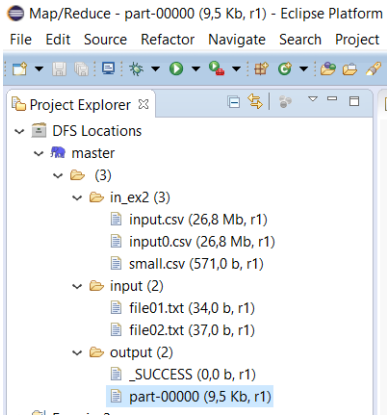

Figure 2.6: Folders and files hosted in HDFS from Eclipse

To open it there are several ways: the first is to do it directly from Eclipse by double-clicking on it. The second is to download the file (also from eclipse) by right clicking on it and choosing "*Download from DFS...*". Finally, from the command console by the command:

```
hadoop fs -get /hdfs/path /local/path
```

The complete file is in the folder "**Exercise2**" and its name is "**ex2_output.csv**" of the submission.

3 Analysis of Movie dataset using Map and Reduce

3.0.1 mapper.py solution

```
import sys

for line in sys.stdin:
  line = line.strip()
  line = line.split('::')

  movies_id = "-1"
  movies_ratings = "-1"
  title = "-1"
  movies_id_tit = "-1"
  users_id = "-1"
  gnres = "-1"

  # ratings data
  # UserID::MovieID::Rating::Timestamp
  #  ^^^       ^^^       ^^^
  # line[0] line[1]  line[2]
  if len(line) == 4:
  users_id = line[0]
  movies_id = line[1]
  movies_ratings = line[2]
  # movies data
  # MovieID::Title::Genres
  #   ^^^       ^^^    ^^^
  # line[0]  line[1] line[2]
  elif len(line) > 1:
  movies_id_tit = line[0]
  title = line[1]
  gnres = line[2]

  print ("{}\t{}\t{}\t{}\t{}\t{}".format(movies_id,
     movies_ratings, movies_id_tit, title, users_id, gnres))
```

3.0.2 Find the movie title which has the maximum average rating

reduce.py

```
import sys

movies_id_ratings = {}
movies_id_title = {}
# Create the corresponding lists and then
# find a movie according to your rating
def createListsMovRatings(movies_id, movies_ratings, id_tit,
    title):
  # If title isn't trash
  if title != "-1":
    # If yet exists this movie_id
    if id_tit not in movies_id_title:
      # Add new title to the lists with id_tit like index
      movies_id_title[id_tit] = title
  # If movies_id isn't trash
  elif movies_id != -1:
    # If already exists this movie_id
    if movies_id in movies_id_ratings:
      # Add new rating to the list of this movie_id
      movies_id_ratings[movies_id].append(movies_ratings)
    # If this origin does not yet exist
    else:
      # Create two new rating list with movies_id like index
      movies_id_ratings[movies_id] = []
      # Add the first rating to the lists
      movies_id_ratings[movies_id].append(movies_ratings)

# Movie title which has the maximum average rating
def movieMaxAvgRating():
  result = [0, 0]
  # Calculation of ratings averages for each movie
  # Storing averages in a list along with your ID
  for movie_id in movies_id_ratings.keys():
    avg_movie_ratings = sum(movies_id_ratings[movie_id])*1.0 /
        len(movies_id_ratings[movie_id])
    # Only update if there is a higher average than there is
    if(avg_movie_ratings > result[1]):
      # Update result with new maximum
      result = [movie_id, avg_movie_ratings]

  return result

# -------------------- Inputs processing
    ----------------------------
for line in sys.stdin:
```

```
43    line = line.strip()
44    movies_id, movies_ratings, id_tit, title = line.split('\t')
45    # Convert inputs to numeric values
46    movies_id = int(movies_id)
47    movies_ratings = float(movies_ratings)
48    id_tit = int(id_tit)
49
50    # Create the corresponding lists and then find
51    # a movie according to your ratings (for PART 1)
52    createListsMovRatings(movies_id, movies_ratings, id_tit,
          title)
53
54 list_avg_movies_ratings = movieMaxAvgRating()
55
56 if(len(list_avg_movies_ratings)>0):
57    print("\n__Movie title which has the maximum average
          rating__")
58    id_best = list_avg_movies_ratings[0]
59    print("TITLE\tMOVIE_ID\tAVG_MOVIE_RATING")
60    print("{}\t{}\t{}".format(movies_id_title[id_best], id_best,
          round(list_avg_movies_ratings[1], 3)))
```

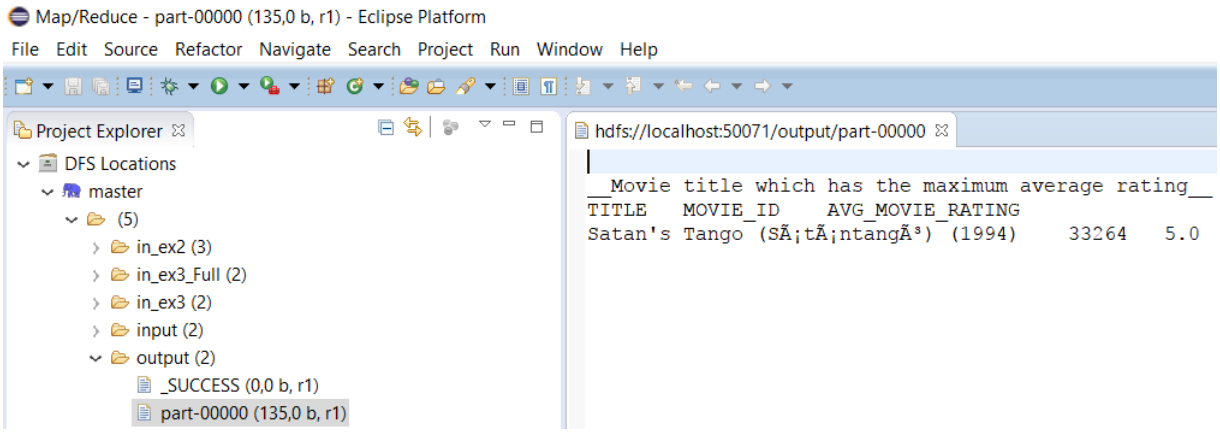

Figure 3.1: Execution of task 1

Mappers	Min	Seg	Time(s)
1	3	33,234	213,234
2	3	30,067	210,067
4	3	26,165	206,165
16	3	20,364	200,364

Table 3.1: Combine results of task 1

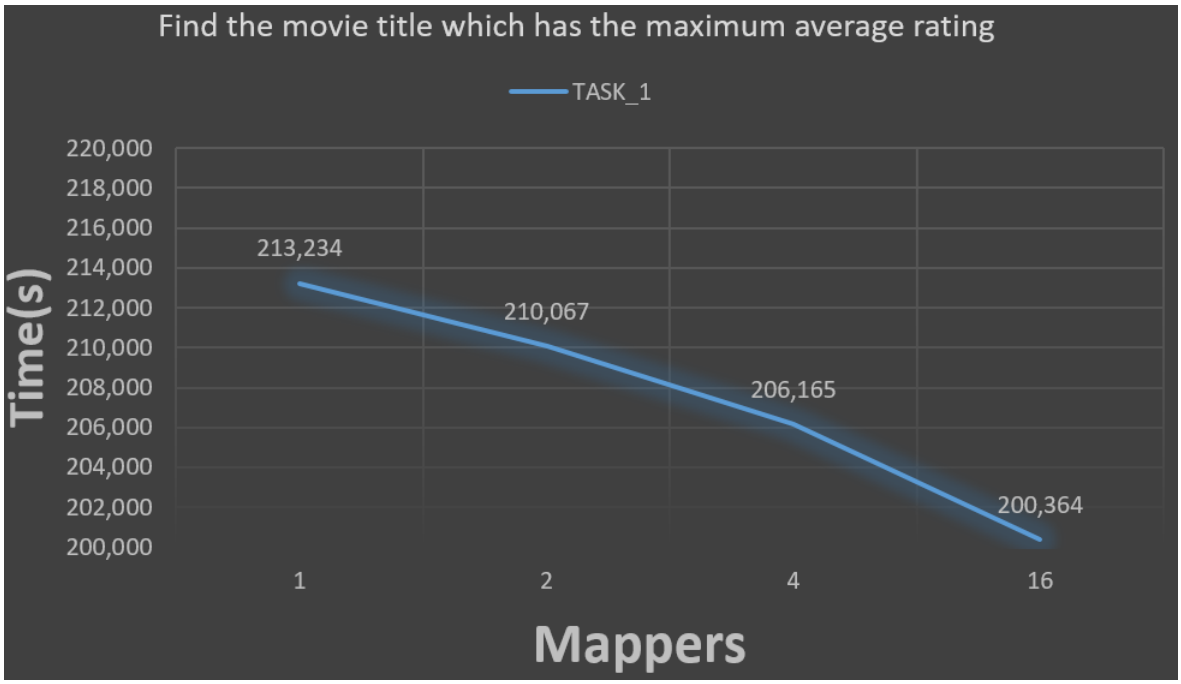

Figure 3.2: Combine results of task 1

3.0.3 Find the user who has assign lowest average rating among all the users who rated more than 40 times

reduce.py

```
import sys

users_ratings = {}

# List of users with min_times occurrences
def userLowestAvgRating(min_times):
  list_users_avg_ratings = [] # List created to get a specific
      ranking
  # Calculation of ratings averages for each user_id
  # Storing averages in a list along with your ID
  for user_id in users_ratings.keys():
    times = len(users_ratings[user_id])
    # Only add if its number of occurrences is greater than or
        equal to min_times
    if(times >= min_times):
      avg_user_ratings = sum(users_ratings[user_id])*1.0 /
          times
      # Add new user_id along with your número de apariciones
          and average to the list
      list_users_avg_ratings.append([user_id, times,
          avg_user_ratings])

  return list_users_avg_ratings
```

```
19
20 # -------------------- Inputs processing
       ------------------------------
21 for line in sys.stdin:
22   line = line.strip()
23   movies_id, movies_ratings, id_tit, title, user_id = line.
       split('\t')
24   # Convert inputs to numeric values
25   movies_ratings = float(movies_ratings)
26   user_id = int(user_id)
27
28   # Create the corresponding lists and then find
29   # a user according to your ratings (for PART 2)
30   if user_id != -1:
31     createListUsers(user_id, movies_ratings)
32
33 min_times = 40
34 list_users_avg_ratings = userLowestAvgRating(min_times)
35
36 if(len(list_users_avg_ratings)>0):
37   # Sort the list based on the ratings average
38   list_users_avg_ratings.sort(key=lambda arr: arr[2])
39
40   print("\n__User who has assign lowest average rating among
       all the users who rated more than {} times__".format(
       min_times))
41   id_best = list_users_avg_ratings[0][0]
42   times_best = list_users_avg_ratings[0][1]
43   avg_user_ratings = round(list_users_avg_ratings[0][2], 3)
44   print("USER_ID\tTIMES\tAVG_USER_RATING")
45   print("{}\t{}\t{}".format(id_best, times_best,
       avg_user_ratings))
```

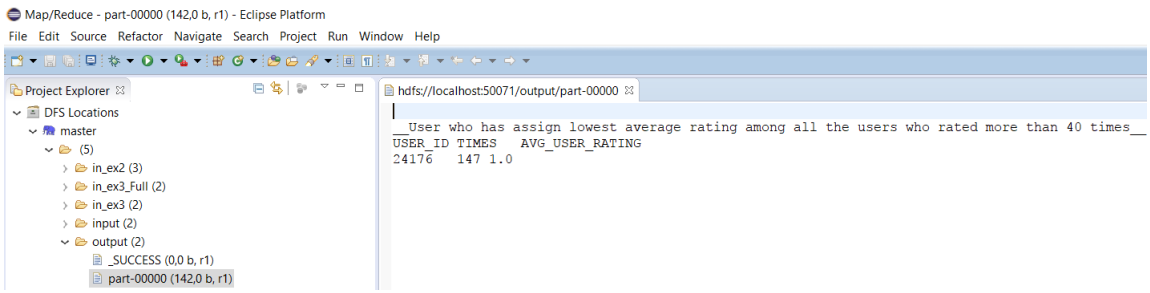

Figure 3.3: Execution of Task2

Mappers	Min	Seg	Time(s)
1	3	37,098	217,098
2	3	34,041	214,041
4	3	30,406	210,406
16	3	25,096	205,096

Table 3.2: Combine results of Task2

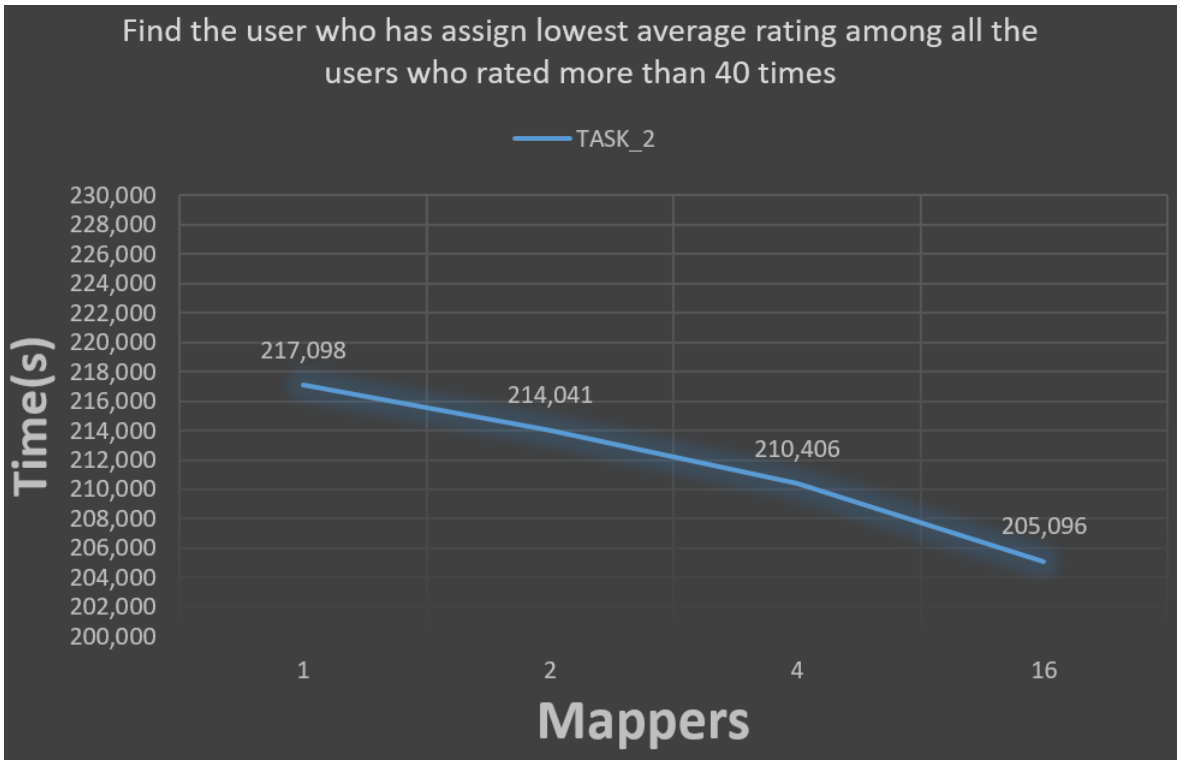

Figure 3.4: Combine results of Task2

3.0.4 Find the highest average rated movie genre

reduce.py

```
1  import sys
2
3  movies_id_ratings = {}
4  movies_id_gnres = {}
5
6  def highestAvgRatedMovGenre():
7    maxi = 0
8    result = "-1"
9    listGenres = {}
10   # Calculation of ratings averages for each genre
11   for movie_id in movies_id_ratings.keys():
12     avg_movie_ratings = sum(movies_id_ratings[movie_id])*1.0 /
         len(movies_id_ratings[movie_id])
```

```
13      # Go down the gnre list of each movie
14      for i in movies_id_gnres[movie_id]:
15        # If it does not exist the genre is created with a new
              value, if it does not exist it is updated
16        if i in listGenres:
17          listGenres[i] = (listGenres[i] + avg_movie_ratings)/2
18        else:
19          listGenres[i] = avg_movie_ratings
20      # Go down the gnre list to find max average rating
21    for i in listGenres:
22      if(listGenres[i] > maxi):
23        maxi = listGenres[i]
24        result = [i, maxi]
25    return result
26
27  # -------------------- Inputs processing
        ----------------------------
28  for line in sys.stdin:
29    line = line.strip()
30    movies_id, movies_ratings, id_tit, title, user_id, gnres =
          line.split('\t')
31    # Convert inputs to numeric values
32    movies_id = int(movies_id)
33    movies_ratings = float(movies_ratings)
34    id_tit = int(id_tit)
35    user_id = int(user_id)
36
37    # Create the corresponding lists and then find
38    # a gnre according to movies ratings (for PART 3)
39    createListsGnresRatings(movies_id, movies_ratings, id_tit,
          gnres)
40
41  print("\n__Find the highest average rated movie genre__")
42  par3 = highestAvgRatedMovGenre()
43  print("GNRE\tAVG_GNR_RATING")
44  print("{}\t{}".format(par3[0], par3[1]))
```

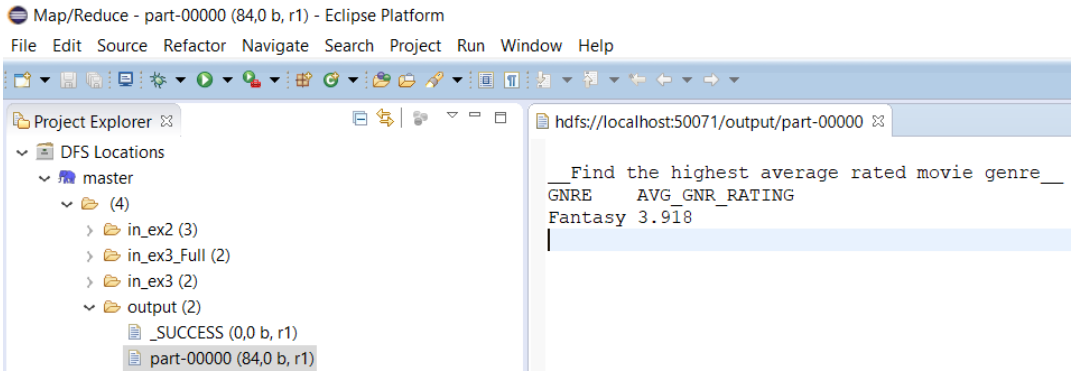

Figure 3.5: Execution of Task3

Mappers	Min	Seg	Time(s)
1	3	17,944	197,944
2	3	15,968	195,968
4	3	13,771	193,771
16	3	5,362	185,362
100	3	4,299	184,299

Table 3.3: Combine results of Task3

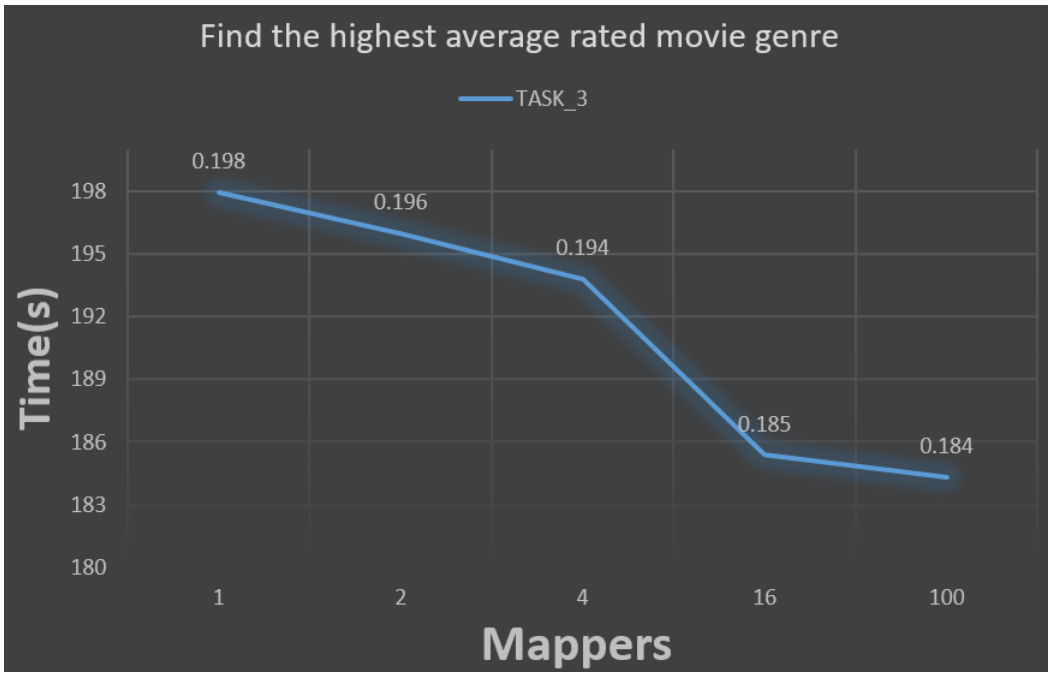

Figure 3.6: Combine results of Task3

Note: all measurements have been made with "**time**" under the console "**cygwin64**".