**Bài thực hành số 03. Lập trình MapReduce trên Spark**

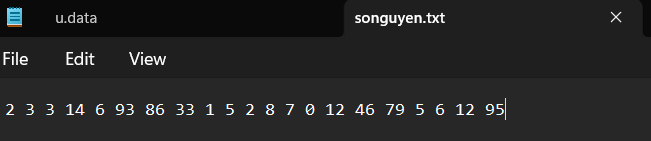
**Link bài làm trên Databricks:** <https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/345358637122346/3220407689531383/5017236833160125/latest.html>

**Bài 1. Tìm số lớn nhất**

Cho tệp songuyen.txt chứa các số nguyên, giữa các số cách nhau bởi khoảng

trống. Viết chương trình tìm số lớn nhất trong tệp.

file songuyen.txt:



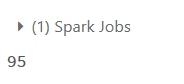
rdd1 = sc.textFile('/FileStore/Bigdata\_task03/songuyen-1.txt')

rdd1\_m1 = rdd1.flatMap(lambda x: x.split(' '))

rdd1\_m2 = rdd1\_m1.map(lambda x: int(x))

result1 = rdd1\_m2.max()

print(result1)



**Bài 2. Đếm số lấn xuất hiện một xâu**

Viết chương trình đếm số lần xuất hiện của xâu x trong một tệp.

Sử dụng tệp README.md đếm xâu ‘Spark’:

rdd2 = sc.textFile('/FileStore/Bigdata\_task03/README.md')

rdd2\_m1 = rdd2.map(lambda line: line.count('Spark'))

result2 = rdd2\_m1.reduce(lambda x,y: x+y)

print(result2)



**Bài 3. Bigram**

Bigram (hay 2-gram) là những cặp từ liên tiếp trong một câu trích ra từ một văn

bản. Ví dụ, câu “Trường Đại học Quy Nhơn nằm bên bờ biển.” thì có các bigram sau :(xét các từ đơn giản là cách nhau bởi khoảng cách): “Trường Đại”, “Đại học”, “học Quy”, “Quy Nhơn”, “Nhơn nằm”, “nằm bên”, “bên bờ”, “bờ biển”.

Cho trước một tệp văn bản HXH.txt, hãy thống kê số lần xuất hiện từng cặp từ

trong văn bản, sắp thứ tự giảm theo số lần xuất hiện rồi ghi ra thư mục HXH.

rdd3 = sc.textFile('/FileStore/Bigdata\_task03/HXH.txt')

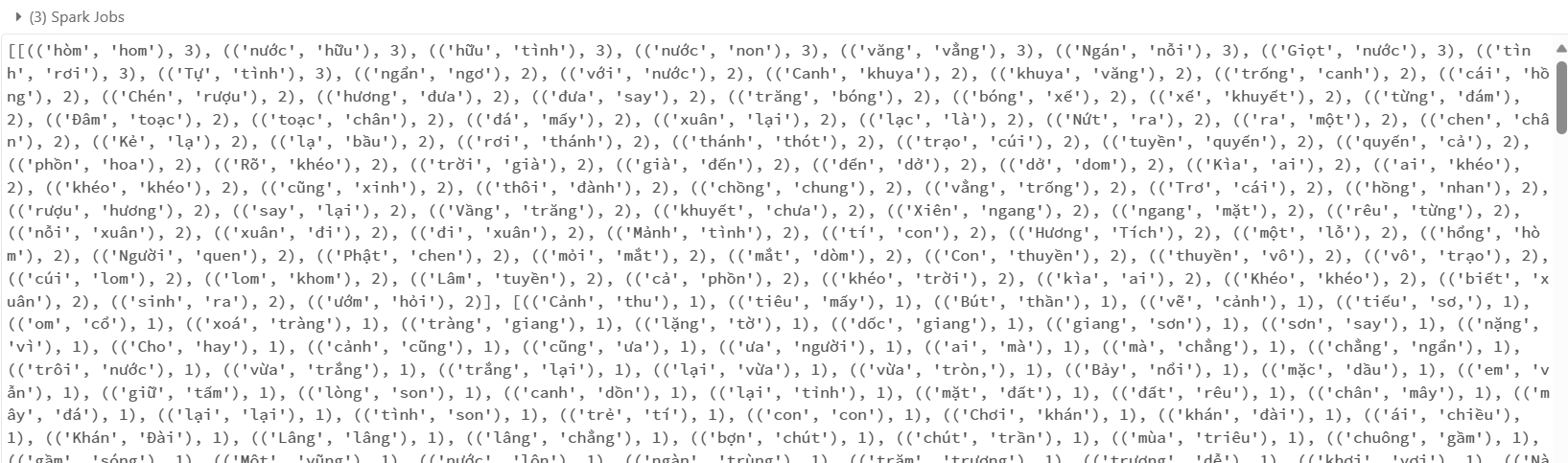
rdd3\_m1 = rdd3.flatMap(lambda line: line.split(".")) \

        .map(lambda line: line.strip().split(" ")) \

        .flatMap(lambda xs: (tuple(x) for x in zip(xs, xs[1:])))

rdd3\_m2 = rdd3\_m1.map(lambda x: (x,1)).reduceByKey(lambda x,y: x+y).sortBy(lambda x: x[1], ascending=False)

print(rdd3\_m2.glom().collect())



**Bài 4. Rating movies:**

- 4.1 Đếm số bộ phim rating của mỗi người dùng.

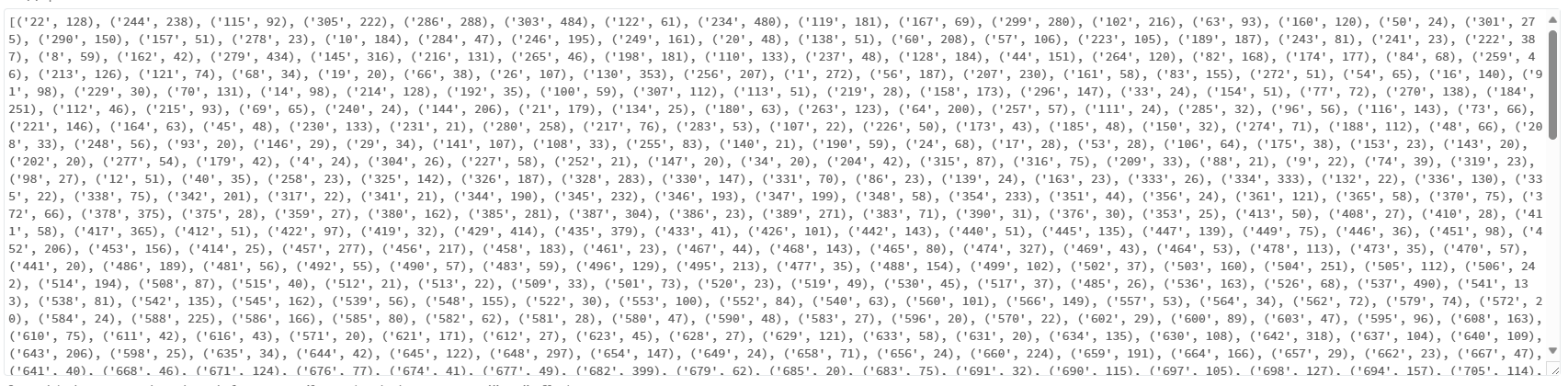
rdd4 = sc.textFile('/FileStore/Bigdata\_task03/u.data')

rdd41\_m1 = rdd4.map(lambda r: r.split('\t'))

rdd42\_m2 = rdd41\_m1.map(lambda r: (r[0], 1))

result = rdd42\_m2.reduceByKey(lambda x, y: x + y)

print(result.collect())



- 4.2 Tính trung bình rating của mỗi người dùng.

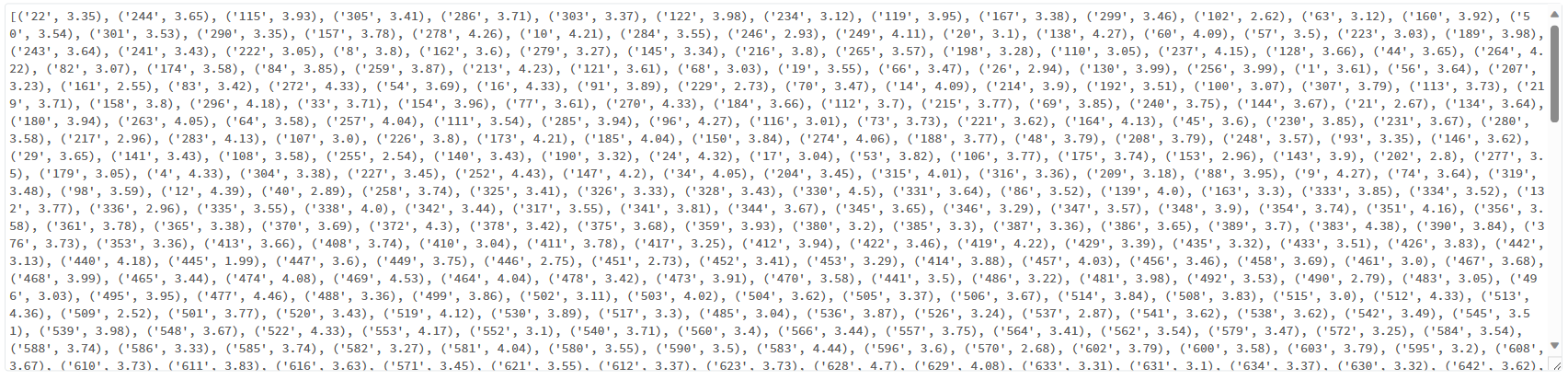
def fmean(l):

    return round(sum(l)/len(l), 2)

rdd42\_m1 = rdd4.map(lambda r: r.split('\t')).map(lambda r: (r[0], int(r[2])))

rdd42\_m2 = rdd42\_m1.groupByKey().mapValues(fmean)

print(rdd42\_m2.collect())



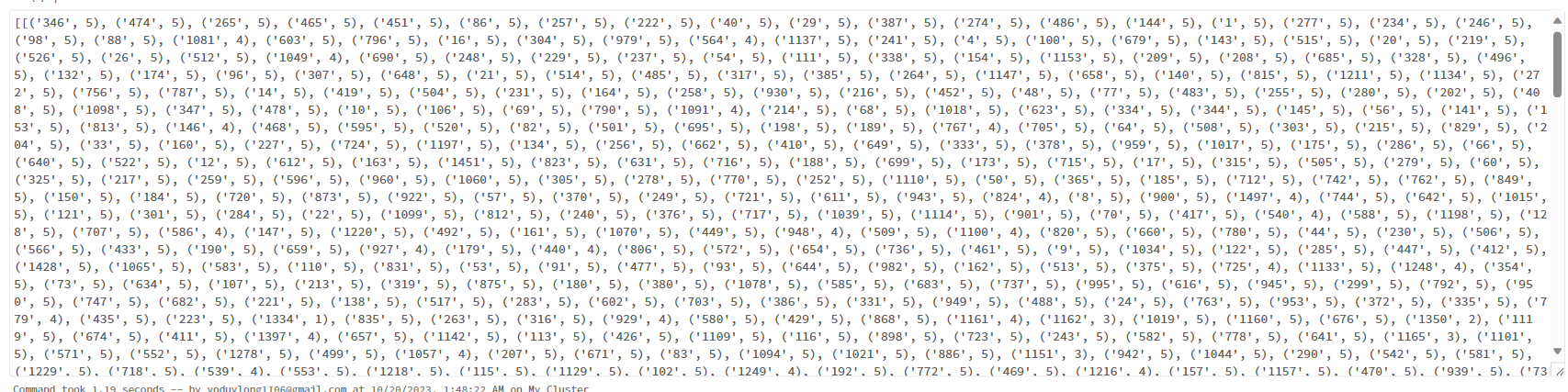
- 4.3 Tính trung bình, cao nhất, thấp nhất rating cho mỗi bộ phim.

Min

rdd43\_m1 = rdd4.map(lambda r: r.split('\t')).map(lambda r: (r[1], int(r[2])))

rdd43\_max = rdd43\_m1.groupByKey().mapValues(max)

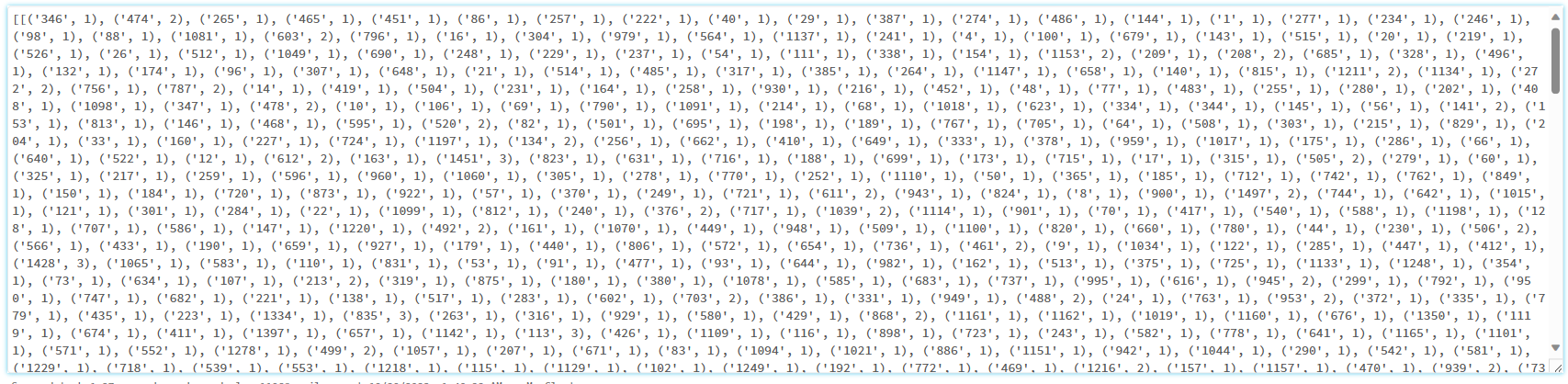
print(rdd43\_max.glom().collect())



Max

rdd43\_min = rdd43\_m1.groupByKey().mapValues(min)

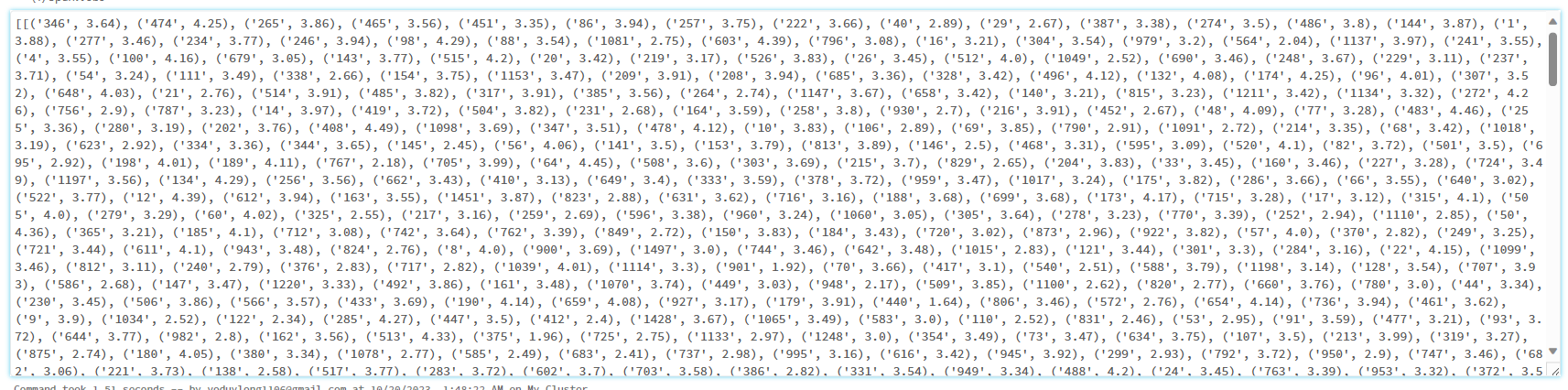
print(rdd43\_min.glom().collect())



AVG

rdd43\_avg = rdd43\_m1.groupByKey().mapValues(fmean)

print(rdd43\_avg.glom().collect())



- 4.4 Tính những bộ phim rating chung của từng cặp người dùng.

rdd44\_m1 = rdd4.map(lambda r: r.split('\t')).map(lambda x: (x[0], x[1]))

rdd44\_m2 = rdd44\_m1.groupByKey().mapValues(set)

cartesian\_rdd = rdd44\_m2.cartesian(rdd44\_m2)

filter\_rdd = cartesian\_rdd.filter(lambda x: x[0][0] != x[1][0])

result = filter\_rdd.map(lambda x: ((x[0][0], x[1][0]), x[0][1] & x[1][1]))

print(result.collect())

