

Ronaldlee Ejalu
CSC 555 Mining Big Data
Assignment 6

Due Wednesday, 11/17

Suggested Reading: Hadoop: The Definitive Guide Ch19; Mining of Massive Datasets: Ch9

1)

a) Solve 9.3.1-a (normalize the ratings based on a threshold), 9.3.1-e

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>A</i>	4	5		5	1		3	2
<i>B</i>		3	4	3	1	2	1	
<i>C</i>	2		1	3		4	5	3

Figure 9.8: A utility matrix for exercises

Exercise 9.3.1: Figure 9.8 is a utility matrix, representing the ratings, on a 1–5 star scale, of eight items, *a* through *h*, by three users *A*, *B*, and *C*. Compute the following from the data of this matrix.

- (a) Treating the utility matrix as boolean, compute the Jaccard distance between each pair of users.

Treating the Utility matrix as boolean:

Replace :

- 1, 2 => No rating
- 3, 4, 5 => 1

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>
<i>A</i>	1	1		1			1	
<i>B</i>		1	1	1				
<i>C</i>				1		1	1	1

Computing the Jaccard distance between each pair of users:

A<->B Jaccard Similarity of 2/5 (distance of 3/5).

B<->C Jaccard Similarity of 1/6 (distance of 5/6).

A<->C Jaccard Similarity of 2/6 (distance of 4/6).

- (e) Normalize the matrix by subtracting from each nonblank entry the average value for its user.

Average value for user A: $20/6 = 3.33$

Average value for user B: $14/6 = 2.33$

Average value for c: $18/6 = 3$

	a	b	c	d	e	f	g	h
A	4-3.33	5-3.33		5-3.33	1-3.33		3-3.33	2-3.33
B		3-2.33	4-2.33	3-2.33	1-2.33	2-2.33	1-2.33	
C	2-3		1-3	3-3		4-3	5-3	3-3

	a	b	c	d	e	f	g	h
A	0.67	1.67		1.67	-2.33		-0.33	-1.33
B		0.67	1.67	0.67	-1.33	-0.33	-1.33	
C	-1		-2	0		1	2	0

- b) Describe a strategy that is used to make a utility matrix less sparse
To unite the attributes, we use clustering since the utility matrix is less sparse; we use hierarchy clustering to group things together.

	a	b	c	d	e	f	g	h
A	4	5		5	1		3	2
B		3	4	3	1	2	1	
C	2		1	3		4	5	3

	d	g
A	4.7	2
B	3.3	1.3
C	3	4

2)

- a) Describe at least one mechanism that ensures that data is not lost in Spark when a failure occurs.

Spark is equipped with Resilient Distributed Dataset (RDD), which acts as central abstraction in Spark. This means that Spark can automatically reconstruct a lost partition by recomputing it from the RDDs that it was computed from. Spark's Directed Acyclic Graph (DAG) engine processes a number of operators and translates them into a single job for the user. The moment there is any failure, DAG will be re-executed from the nearest node of failure to recompute the Resilient Distributed Dataset (RDD).

Also, Spark is enabled with write a head logs where it saves data to fault-tolerant storage before it is processed; these logs have to be explicitly saved.

Furthermore, if you were to lose everything, you can go back to the source, for example, could be HDFS or amazon S3, which are guaranteed to be reliable storage

solutions and recompute whatever you were doing. This is the one special mechanism that is completely unique to spark.

- b) Describe at least one mechanism that ensures that data is not lost in Storm when a failure occurs.

Apache storm is fault-tolerant: when workers die, Apache Storm will automatically restart them. If a node dies, the worker will be started on another node. Nimbus and the Supervisors, which are Apache storm daemons will be re-started like nothing happened since they are designed to be stateless and fail-fast.

- c) From a resource managing perspective, which Hadoop nodes should be chosen to run Spark tasks?

The resource manager (Cluster Manager): This is an external service that acquires resources on the cluster. A spark cluster can run in either yarn cluster or yarn client mode. Spark needs as much RAM because certain components expect certain types of resources and RAM is the biggest need since processing is done in memory.

The Job tracker which coordinates all the jobs run in a Spark cluster.

- d) Implement (in python only, without actual Storm) a solution that would compute streaming queries average for a specified window. For example, to compute a 4-value windowed average that moves 2 tuples at a time, you can use the following line (make sure that your code supports other sizes as well).
You are **not** allowed to read the entire input before producing output, because the input stream is infinite. Instead, you should compute and print the output as the data arrives.

```
cat mydata | python storm.py 4 2
```

Outside of Linux, you can simulate receiving data from standard input as follows:

```
fd = open('mydata', 'r')
sys.stdin = fd
for line in sys.stdin:
    # Your code goes here.
```

Assuming that mydata file contains (one value per line, no error checking is necessary)

```
5
3
5
11
6
3
5
3
7
```

Your command above should output the following three averages (representing an average of (5,3,5,11), (5,11,6,3), (5,3,5,3)):

6
6.25
4

```
ec2-user@ip-172-31-16-126:~  
[ec2-user@ip-172-31-16-126 ~]$ cat mydata | python storm.py  
Current window: [5, 3, 5, 11]  
  
Current Window Average 6  
Current window: [3, 5, 11, 6]  
  
Current Window Average 6  
Current window: [5, 11, 6, 3]  
  
Current Window Average 6  
Current window: [11, 6, 3, 5]  
  
Current Window Average 6  
Current window: [6, 3, 5, 3]  
  
Current Window Average 4  
Current window: [3, 5, 3, 7]  
  
Current Window Average 4  
[ec2-user@ip-172-31-16-126 ~]$
```

Code used:

```

import sys

fd = open('mydata', 'r')
sys.stdin = fd

wSize = 4

movingTupleSize = 1
window = []

for line in sys.stdin:
    line = line.strip()
    window.append(int(line))

    if len(window) > wSize - 1:
        print('Current window: {}'.format(window))
        avg = sum(window)/len(window)
        print('\nCurrent Window Average {}'.format(avg))

        #print('\n')

        for i in range(movingTupleSize):
            window.pop(0)

```

3) Download and setup Mahout

a) Run KMeans clustering using Mahout:

cd

(download mahout zip package)

wget <http://cdmgesarprd01.dpu.depaul.edu/CSC555/apache-mahout-distribution-0.11.2.zip>

(Unzip the file)

unzip apache-mahout-distribution-0.11.2.zip

set the environment variables (as always, you can put these commands in ~/.bashrc to automatically set these variables, and run one-time source ~/.bashrc to refresh now)

```
export MAHOUT_HOME=/home/ec2-user/apache-mahout-distribution-0.11.2
```

```
export PATH=/home/ec2-user/apache-mahout-distribution-0.11.2/bin:$PATH
```

be absolutely sure you set Hadoop home variable (if you haven't yet):

Create a new numeric file with 360,000 rows and 10 columns, separated by space – you can generate numeric data as you prefer, but submit the code/method that you have used.

The python code used to generate the file:

```
#!/usr/bin/python
```

```
import numpy as np
```

```
from numpy import savetxt
```

```
import random
```

```
arr = np.random.randint(50, size = (360000,10))
```

```
np.savetxt('numericGeneratedFile.csv', arr, fmt= '%i', delimiter = ' ')
```

Using mahout kmeans clustering (e.g., Slide #4 from Module 9), cluster the data into 8 clusters with up to 10 iterations.

Submit the command line you used and a screenshot of the last page of output.

This is the command line I used:

```
bin/mahout org.apache.mahout.clustering.syntheticcontrol.kmeans.Job --maxIter 10 --numClusters 8 --t1 5 --t2 3 --input numericGeneratedFile --output kmeansRes
```

The screenshot of the last page:

```
ec2-user@ip-172-31-16-126:~/apache-mahout-distribution-0.11.2
```

```
1.0 : [distance=951.1992367167166] : [9.0,40.0,40.0,6.0,16.0,31.0,16.0,8.0,39.0,6.0]
1.0 : [distance=1245.3856316747006] : [13.0,20.0,24.0,15.0,27.0,45.0,4.0,5.0,27.0,18.0]
1.0 : [distance=2324.228975327951] : [8.0,3.0,22.0,6.0,21.0,29.0,42.0,47.0,41.0,20.0]
1.0 : [distance=544.7709479064542] : [31.0,29.0,21.0,21.0,16.0,30.0,24.0,32.0,45.0,9.0]
1.0 : [distance=1562.6507356110124] : [8.0,33.0,48.0,34.0,10.0,31.0,5.0,13.0,19.0,7.0]
1.0 : [distance=1829.3513990342763] : [29.0,47.0,31.0,16.0,46.0,3.0,19.0,7.0,33.0,12.0]
1.0 : [distance=1020.4226064646173] : [25.0,44.0,40.0,14.0,41.0,32.0,21.0,17.0,25.0,13.0]
1.0 : [distance=2761.1465781585557] : [{"0":2.0}, {"1":26.0}, {"2":13.0}, {"3":48.0}, {"5":40.0}, {"6":4.0}, {"7":9.0}, {"8":38.0}, {"9":13.0}]
1.0 : [distance=1627.0536104451567] : [{"0":22.0}, {"1":30.0}, {"2":15.0}, {"3":13.0}, {"4":41.0}, {"5":9.0}, {"6":19.0}, {"7":3.0}, {"8":42.0}]
1.0 : [distance=1818.1914255712072] : [{"1":43.0}, {"2":38.0}, {"3":2.0}, {"4":27.0}, {"5":31.0}, {"6":16.0}, {"7":42.0}, {"8":46.0}, {"9":37.0}]
1.0 : [distance=1572.5430399011493] : [23.0,46.0,28.0,19.0,11.0,42.0,6.0,5.0,45.0,32.0]
1.0 : [distance=1505.3848797949995] : [29.0,29.0,8.0,21.0,6.0,3.0,13.0,19.0,40.0,1.0]
1.0 : [distance=847.416237601281] : [14.0,25.0,32.0,3.0,21.0,5.0,10.0,34.0,36.0,27.0]
1.0 : [distance=1580.4348134526754] : [46.0,20.0,33.0,31.0,25.0,32.0,13.0,41.0,48.0,6.0]
1.0 : [distance=1459.5234910289673] : [5.0,3.0,29.0,2.0,26.0,14.0,3.0,32.0,33.0,19.0]
1.0 : [distance=1323.2355653323739] : [17.0,37.0,27.0,1.0,13.0,15.0,14.0,48.0,17.0,19.0]
1.0 : [distance=2082.3968214137567] : [28.0,40.0,29.0,23.0,2.0,48.0,2.0,37.0,27.0,38.0]
1.0 : [distance=854.0783340187954] : [16.0,13.0,43.0,18.0,11.0,34.0,17.0,34.0,37.0,4.0]
1.0 : [distance=1376.6974406087993] : [40.0,22.0,27.0,6.0,43.0,22.0,2.0,33.0,45.0,10.0]
1.0 : [distance=1867.1854105336115] : [{"1":47.0}, {"2":26.0}, {"3":20.0}, {"4":47.0}, {"5":18.0}, {"6":10.0}, {"7":43.0}, {"8":35.0}, {"9":8.0}]
1.0 : [distance=1119.019023978991] : [16.0,26.0,37.0,3.0,11.0,33.0,6.0,49.0,44.0,14.0]
1.0 : [distance=2006.5753707282147] : [40.0,30.0,42.0,23.0,44.0,35.0,4.0,5.0,46.0,26.0]
1.0 : [distance=1806.9954061107892] : [34.0,21.0,10.0,24.0,30.0,25.0,4.0,19.0,43.0,7.0]
1.0 : [distance=653.6277811660766] : [17.0,38.0,36.0,9.0,10.0,38.0,8.0,20.0,47.0,13.0]
1.0 : [distance=556.0963791315789] : [21.0,26.0,36.0,20.0,10.0,13.0,1.0,28.0,38.0,8.0]
1.0 : [distance=2159.1685595827057] : [7.0,46.0,40.0,36.0,22.0,39.0,10.0,25.0,38.0,45.0]
1.0 : [distance=1806.9954061107892] : [27.0,23.0,33.0,43.0,42.0,39.0,5.0,18.0,43.0,6.0]
1.0 : [distance=1673.9989001399808] : [40.0,27.0,49.0,6.0,18.0,24.0,7.0,47.0,19.0,14.0]
1.0 : [distance=2053.116502970586] : [47.0,18.0,37.0,39.0,15.0,22.0,5.0,10.0,48.0,23.0]
1.0 : [distance=2335.8879757702325] : [14.0,10.0,22.0,44.0,14.0,48.0,4.0,14.0,36.0,1.0]
1.0 : [distance=1465.0077457835014] : [{"0":14.0}, {"1":1.0}, {"2":29.0}, {"3":9.0}, {"5":16.0}, {"6":15.0}, {"7":33.0}, {"8":36.0}, {"9":12.0}]
1.0 : [distance=944.592663961299] : [{"0":15.0}, {"1":33.0}, {"2":17.0}, {"3":9.0}, {"4":29.0}, {"5":30.0}, {"6":24.0}, {"7":12.0}, {"8":24.0}]
1.0 : [distance=1273.2377767432536] : [28.0,12.0,41.0,2.0,38.0,21.0,3.0,19.0,27.0,7.0]
1.0 : [distance=801.5268523735049] : [37.0,27.0,31.0,22.0,14.0,22.0,10.0,19.0,49.0,1.0]
1.0 : [distance=2622.6943004053483] : [7.0,46.0,10.0,21.0,33.0,38.0,21.0,49.0,11.0,1.0]
1.0 : [distance=1336.9851893925243] : [14.0,12.0,20.0,10.0,15.0,8.0,18.0,46.0,26.0,18.0]
1.0 : [distance=1247.3356095605905] : [26.0,23.0,12.0,10.0,4.0,47.0,15.0,26.0,36.0,19.0]
1.0 : [distance=1281.681562678681] : [4.0,34.0,23.0,15.0,28.0,39.0,16.0,29.0,48.0,37.0]
1.0 : [distance=3049.922296867091] : [36.0,39.0,42.0,3.0,7.0,3.0,48.0,3.0,34.0,1.0]
1.0 : [distance=819.2139819621843] : [20.0,37.0,32.0,8.0,26.0,11.0,2.0,39.0,40.0,3.0]
1.0 : [distance=1643.7859855004408] : [7.0,47.0,27.0,7.0,43.0,33.0,9.0,7.0,26.0,23.0]
1.0 : [distance=1722.393150471693] : [3.0,24.0,4.0,10.0,16.0,47.0,8.0,28.0,30.0,6.0]
1.0 : [distance=2519.6163260577177] : [23.0,47.0,48.0,6.0,19.0,1.0,44.0,9.0,46.0,20.0]
1.0 : [distance=1550.0867373801411] : [37.0,26.0,49.0,2.0,13.0,12.0,12.0,43.0,46.0,6.0]
1.0 : [distance=1494.6706383089331] : [37.0,40.0,18.0,15.0,7.0,26.0,29.0,48.0,38.0,19.0]
1.0 : [distance=1680.6033229617424] : [40.0,29.0,23.0,29.0,29.0,45.0,9.0,47.0,39.0,10.0]
1.0 : [distance=555.7180509582031] : [14.0,21.0,32.0,19.0,16.0,32.0,17.0,8.0,39.0,8.0]
1.0 : [distance=1772.6151318958418] : [10.0,17.0,36.0,2.0,15.0,5.0,38.0,39.0,46.0,21.0]
1.0 : [distance=1056.1368479506837] : [23.0,42.0,48.0,20.0,11.0,13.0,10.0,8.0,33.0,11.0]
1.0 : [distance=2470.102305712737] : [9.0,44.0,3.0,29.0,3.0,48.0,4.0,15.0,35.0,3.0]
1.0 : [distance=2621.230125261609] : [{"1":40.0}, {"2":49.0}, {"3":15.0}, {"4":21.0}, {"5":22.0}, {"6":49.0}, {"7":47.0}, {"8":40.0}, {"9":7.0}]
1.0 : [distance=2010.016060688411] : [{"0":3.0}, {"2":41.0}, {"3":11.0}, {"4":1.0}, {"5":32.0}, {"6":22.0}, {"7":35.0}, {"8":43.0}, {"9":21.0}]
1.0 : [distance=1558.2984578578053] : [19.0,26.0,46.0,11.0,44.0,4.0,15.0,42.0,33.0,10.0]
1.0 : [distance=2146.2518855446706] : [{"1":39.0}, {"2":35.0}, {"3":34.0}, {"4":38.0}, {"5":45.0}, {"6":15.0}, {"7":48.0}, {"8":32.0}, {"9":15.0}]
1.0 : [distance=1702.0796166371056] : [47.0,28.0,8.0,19.0,14.0,13.0,7.0,33.0,34.0,1.0]
1.0 : [distance=2026.4875334880544] : [7.0,44.0,14.0,2.0,18.0,46.0,13.0,49.0,45.0,6.0]
1.0 : [distance=902.8054901444029] : [27.0,39.0,18.0,3.0,17.0,19.0,18.0,10.0,44.0,3.0]
1.0 : [distance=1737.9761668361334] : [38.0,25.0,14.0,12.0,7.0,5.0,29.0,9.0,43.0,7.0]
1.0 : [distance=1562.2463570174696] : [9.0,24.0,16.0,23.0,24.0,12.0,1.0,2.0,25.0,27.0]
1.0 : [distance=1657.310311020121] : [7.0,42.0,47.0,10.0,17.0,19.0,19.0,45.0,42.0,37.0]
21/11/12 23:22:31 INFO ClusterDumper: Wrote 8 clusters
21/11/12 23:22:31 INFO MahoutDriver: Program took 321649 ms (Minutes: 5.360816666666667)
```

Don't forget that Mahout requires JobHistoryServer to be running.

b) Run a recommender on the MoveLens dataset.

(Create a directory for movie lens dataset)

```
mkdir Movielens
```

```
cd Movielens
```

```
wget http://cdmgesarprd01.dpu.depaul.edu/CSC555/ml-lm.zip
```

(Unzip the dataset, this one happens to be compressed with Zip rather than GZip)

```
unzip ml-lm.zip
```

```
cd ..
```

Take a look at the data file:

```
more Movielens/ml-lm/ratings.dat
```

(you can press q or Ctrl-C to exit, **more** command shows the first few lines worth of text. Each line contains user ID, movie ID, user rating and the timestamp of the rating, as already discussed in class)

The next step is to use aa Linux command to convert :: separated file into a comma-separated file. First part (cat) will simply output the file. Second part substitutes , for :: and third part of the command extracts just 3 attributes relevant to us (no timestamp)

```
cat MovieLens/ml-1m/ratings.dat | sed -e s/::/,/g | cut -d, -f1,2,3 > MovieLens/ml-1m/ratings.csv
```

(NOTE: if you wanted to extract all 4 columns from the original data set, you could run the same command with “1,2,3,4” instead of “1,2,3”).

Create a movielens directory and copy the articles over to HDFS into that directory:

```
$HADOOP_HOME/bin/hadoop fs -mkdir movielens  
$HADOOP_HOME/bin/hadoop fs -put MovieLens/ml-1m/ratings.csv movielens
```

Split the data set into the 88% training set and 12% evaluation set. In this case we are using Hadoop to perform the split. Naturally, you can change the percentages here to any other value instead of 0.88/0.12.

```
bin/mahout splitDataset -input movielens/ratings.csv -output ml_dataset -trainingPercentage 0.88 -probePercentage 0.12 -tempDir dataset/tmp
```

Verify and report the file sizes of the input ratings.csv file and the two sampled files (the two files are in the /user/ec2-user/ml_dataset/trainingSet/ and /user/ec2-user/ml_dataset/probeSet directories on HDFS side). Do the sampled file sizes add up to the original input file size?

The trainingSet has 9.7 Mega bytes as shown below:

```
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$ hadoop fs -ls -h /user/ec2-user/ml_dataset/trainingSet  
Found 2 items  
-rw-r--r--  2 ec2-user supergroup          0 2021-11-13 01:43 /user/ec2-user/ml_dataset/trainingSet/_SUCCESS  
-rw-r--r--  2 ec2-user supergroup      9.7 M 2021-11-13 01:43 /user/ec2-user/ml_dataset/trainingSet/part-m-00000  
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$
```

The evaluation set had 1.3 Mega bytes as shown below:

```
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$ hadoop fs -ls -h /user/ec2-user/ml_dataset/probeSet  
Found 2 items  
-rw-r--r--  2 ec2-user supergroup          0 2021-11-13 01:43 /user/ec2-user/ml_dataset/probeSet/_SUCCESS  
-rw-r--r--  2 ec2-user supergroup      1.3 M 2021-11-13 01:43 /user/ec2-user/ml_dataset/probeSet/part-m-00000  
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$
```

Yes, the sampled file sizes add up to the original input file size, which is 11 Mega bytes as shown below:

```
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$ hadoop fs -ls -h /user/ec2-user/movielens  
Found 1 items  
-rw-r--r--  2 ec2-user supergroup      11.0 M 2021-11-13 01:38 /user/ec2-user/movielens/ratings.csv  
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$
```


Factorize the rating matrix based on the training set. As always, this is a single line command, be sure to run it as such. The --numfeatures value configures the set of “hidden” variables or the dimension size to use in matrix factorization. --numIterations sets how many passes to perform; we expect a better match with more iterations

```
time bin/mahout parallelALS -input ml_dataset/trainingSet/ -output als/out -tempDir als/tmp -numfeatures 20 -numIterations 3 -lambda 0.065
```

Measure the prediction against the training set:

```
bin/mahout evaluatefactorization -input ml_dataset/probeSet/ -output als/rmse/ -userfeatures als/out/U/ -itemfeatures als/out/M/ -tempDir als/tmp
```

What is the resulting RMSE value? (rmse.txt file is in HDFS /user/ec2-user/als/rmse/)

The RMSE value is 0.89 as shown below:

```
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$ hadoop fs -cat /user/ec2-user/als/rmse/rmse.txt
0.8891858415523369[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$
```

Finally, let's generate some predictions:

```
bin/mahout recommendfactorized -input als/out/userRatings/ -output recommendations/ -userfeatures als/out/U/ -itemfeatures als/out/M/ -numRecommendations 6 -maxRating 5
```

Look at recommendations/part-m-00000 and report the first 10 rows by running the following command. These are top-6 recommendations (note that --numRecommendation setting in the previous command) for each user. Each recommendation consists of movieID and the estimated rating that the user might give to that movie.

```
$HADOOP_HOME/bin/hadoop fs -cat recommendations/part-m-00000 | head
```

```
ec2-user@ip-172-31-16-126:~/apache-mahout-distribution-0.11.2
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$ hadoop fs -cat /user/ec2-user/recommendations/part-m-00000|head
1      [572:5.0,116:4.632903,920:4.4050493,858:4.3894143,318:4.3665304,1250:4.3165193]
2      [572:4.6067853,527:4.2614627,919:4.2367635,1250:4.149129,953:4.1396813,2762:4.1279674]
3      [572:4.5856404,2571:4.506012,318:4.450691,3147:4.449339,110:4.4222116,2762:4.4103694]
4      [751:5.0,2937:5.0,2238:5.0,912:5.0,2935:5.0,3808:5.0]
5      [1361:4.2457714,1002:4.1753144,2503:4.1362376,2330:4.1230917,681:4.1168957,503:4.108034]
6      [572:5.0,2197:5.0,2056:5.0,2156:4.688849,3025:4.5539856,2996:4.5538306]
7      [1036:4.719098,3147:4.6444182,318:4.6067114,2762:4.5929804,1198:4.570129,593:4.5206714]
8      [858:4.6923656,1221:4.522899,50:4.509332,1198:4.47153,318:4.466984,2019:4.460055]
9      [260:4.415432,1196:4.3759565,1198:4.340964,296:4.2567687,858:4.2307487,2905:4.201628]
10     [2197:5.0,572:5.0,2156:4.6770167,1812:4.5642877,2562:4.5392637,1046:4.5242934]
cat: Unable to write to output stream.
[ec2-user@ip-172-31-16-126 apache-mahout-distribution-0.11.2]$
```

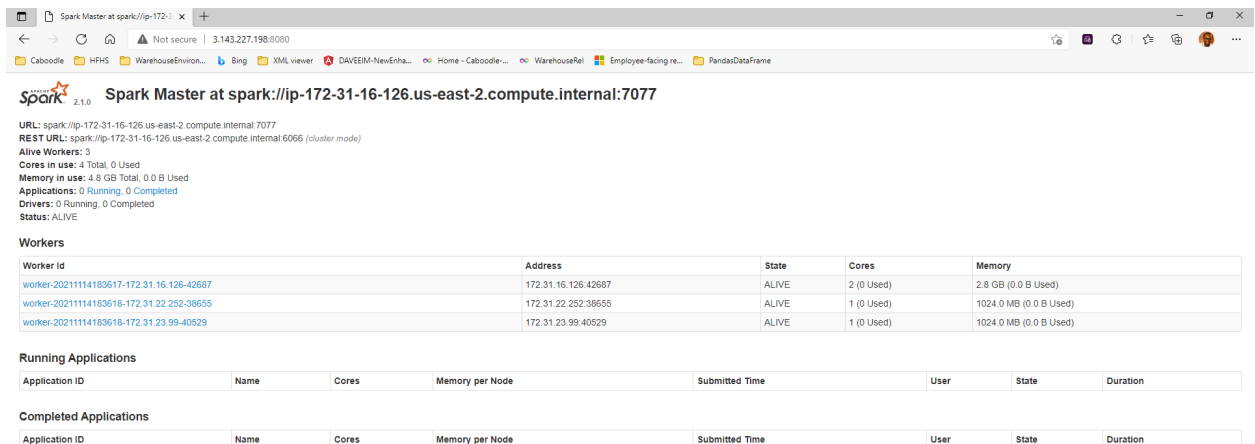
What is the top movie recommendation (movie ID) for users 4 and 6?

User 4: 751

User 6: 572

- 4) Set up a stand-alone 3-node Spark cluster. Note that you can use your existing 3-node cluster. You need to add a text file with workers to the conf directory (conf/workers) directory in the spark folder and copy spark directory to each node in the cluster in the same way as you copied Hadoop to each node in the cluster.

Cluster view status can be seen at the browser page at port 8080 (you would need to modify your firewall settings to open port 8080 exactly as you have with port 50700).



The screenshot shows the Spark Master web interface at `spark://ip-172-31-16-126.us-east-2.compute.internal:7077`. The interface displays the following information:

- URL:** `spark://ip-172-31-16-126.us-east-2.compute.internal:7077`
- REST URL:** `spark://ip-172-31-16-126.us-east-2.compute.internal:6066 (cluster mode)`
- Alive Workers:** 3
- Cores in use:** 4 Total, 0 Used
- Memory in use:** 4.8 GB Total, 0.0 B Used
- Applications:** 0 Running, 0 Completed
- Drivers:** 0 Running, 0 Completed
- Status:** ALIVE

Workers Table:

Worker id	Address	State	Cores	Memory
worker-20211114183617-172.31.16.126-42687	172.31.16.126:42687	ALIVE	2 (0 Used)	2.8 GB (0.0 B Used)
worker-20211114183618-172.31.22.252-38655	172.31.22.252:38655	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20211114183618-172.31.23.99-40529	172.31.23.99:40529	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

Running Applications Table:

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

Completed Applications Table:

Application ID	Name	Cores	Memory per Node	Submitted Time	User	State	Duration
----------------	------	-------	-----------------	----------------	------	-------	----------

You can find a spark binary of the right version here:

<http://dbgroup.cdm.depaul.edu/Courses/CSC555/spark-2.1.0-bin-hadoop2.6.tar>

Execute wordcount in Spark for the bioproject.xml file we used previously. Submit the code you used and the screenshot with some output (you can either use the output from Spark terminal or write your output file to HDFS).

Code used:

```
text_file = sc.textFile("hdfs://ec2-3-143-227-198.us-east-2.compute.amazonaws.com/data/bioproject.xml")
word_file = text_file.flatMap(lambda line: line.split(" "))
wordCounts = word_file.map(lambda word: (word, 1)).reduceByKey(lambda a,b:a + b)
wordCounts.saveAsTextFile("hdfs://ec2-3-143-227-198.us-east-2.compute.amazonaws.com/data/wordCntResults")
```

