گزارش تمرین شماره یک BIG DATA

تاریخ تحویل : ۹۸/۸/۱۰ نمره : ۱/۵

---

۱. فایل Shakespeare.txt شامل کلیه نوشته هاي شکسپیر مي باشد. با استفاده از SPARK برنامه اي بنویسید که

الف: تعداد کل کلمات این متن را مشخص نمایید.

ب:تعداد کلمات بدون تکرار چقدر است؟

ج : ده کلمه اي که بیشترین تعداد تکرار را دارند کدامند و هر کدام چند بار تکرار شده اند؟

نتایج:

it has 1050323 word

part-b: teadad kalamat bedoone tekrar = 8785

part-c: 10 kalamat por tekrar shamel :

(27843,the)

(26847,and)

(22538,i)

(19882,to)

(18307,of)

(14800,a)

(13928,you)

(12490,my)

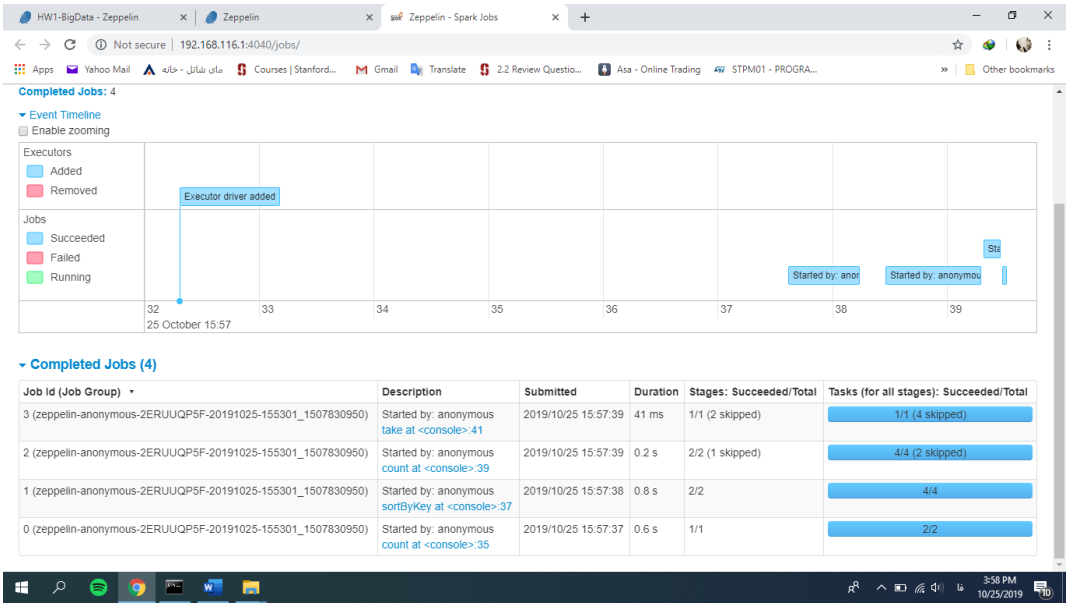
(11563,that)

کد:

```
import org.apache.spark._
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext._
import org.apache.spark.rdd.RDD._
//HW1-1
val line = sc.textFile("C:/BigData/HW1/Shakespeare.txt")
val word = line.flatMap(w => w.split("[^a-zA-Z]+") )
//part a
println("it has "+word.map(x=> 1).count()  +"  word" )
//part b and c
val NumList = word.map( x => (x.toLowerCase,1)
).reduceByKey((x,y) => x+y ).map(t =>
(t._2,t._1)).sortByKey(false)

val part_b =NumList.filter(t=>t._1==1).count()

val part_c =NumList.take(10)
//report
println("part-b: teadad kalamat bedoone tekrar = " +
part_b.toString)
println("part-c:  10 kalamat por tekrar shamel : " )
part_c.foreach(println)
```

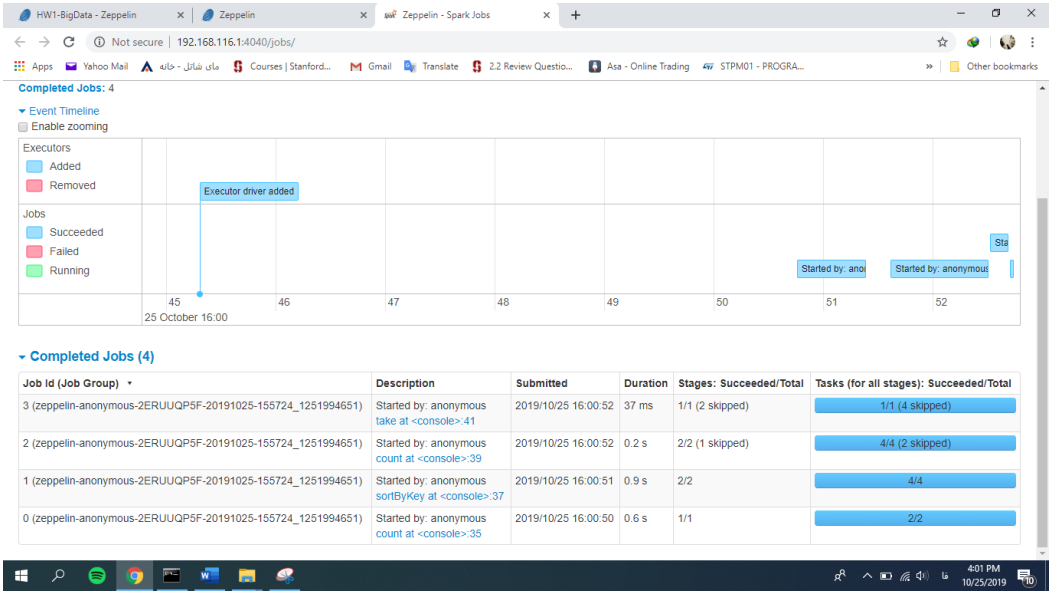
نتایج با یک هسته فعال ۲۵ ثانیه

User: hosse
Total Uptime: 25 s
Scheduling Mode: FIFO



نتایج با چهار هسته فعال ۱۳ ثانیه

User: hosse
Total Uptime: 13 s
Scheduling Mode: FIFO

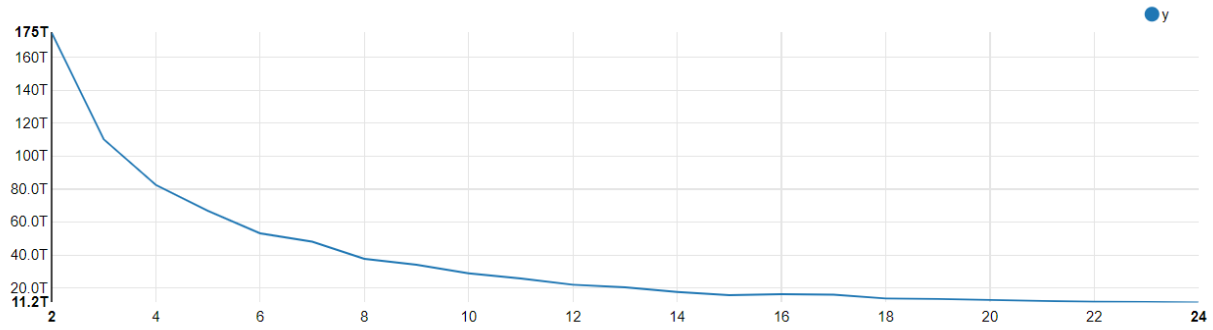


۲. سه فايل C1,C2,C3 هر كدام حاوي اطلاعات دو ستوني متني مي باشند كه هر سطر ان مختصات يک نقطه دو بعدي را نشان مي دهد.

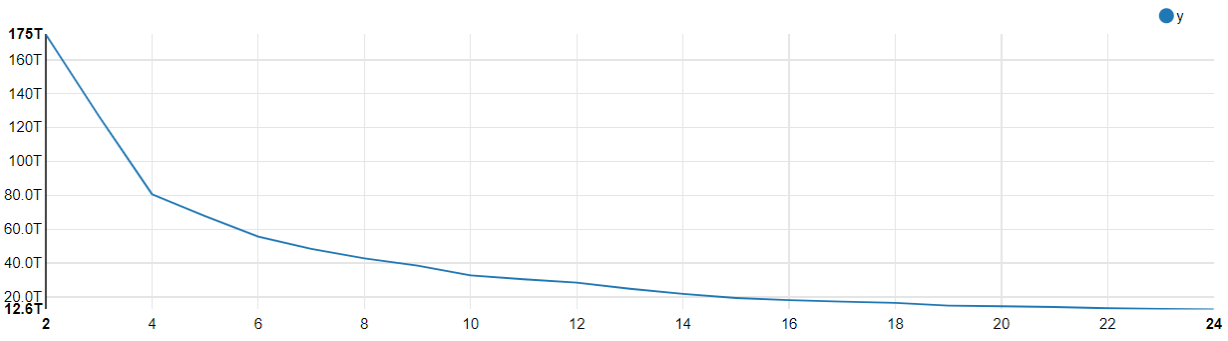
با استفاده از SPARK دو برنامه بنويسيد كه با استفاده از دو روش خوشه بندي K-means++ و Bisecting K-means

الف : براي هر فايل داده نمودار هزينه كلاستر را براي k=2 to25 رسم نماييد.

C1=>K-means++:



C1=> Bisecting K-means



تعداد كلاستر بهينه ۱۴ است كه مراكز كلاستر آن در روش اول:

```
Cluster Center 0: [397571.4303178484,499977.12469437654]

Cluster Center 1: [661286.3097345133,264471.66076696164]
```

```
Cluster Center 2: [281100.28695652174,661698.2130434783]
Cluster Center 3: [532478.7464788733,398416.54694835684]
Cluster Center 4: [647607.7537091988,744965.2225519288]
Cluster Center 5: [507263.57547169813,222529.19496855346]
Cluster Center 6: [316805.37394247035,303224.1912013536]
Cluster Center 7: [325990.4903846154,774807.1826923077]
Cluster Center 8: [596425.2573839662,577422.0654008439]
Cluster Center 9: [503860.3166666667,776553.9133333333]
Cluster Center 10: [762643.5092592592,579700.8055555555]
Cluster Center 11: [738130.5721649484,430456.7345360825]
Cluster Center 12: [243735.86649214663,484872.4869109948]
Cluster Center 13: [464465.51587301586,638022.0026455026]
```

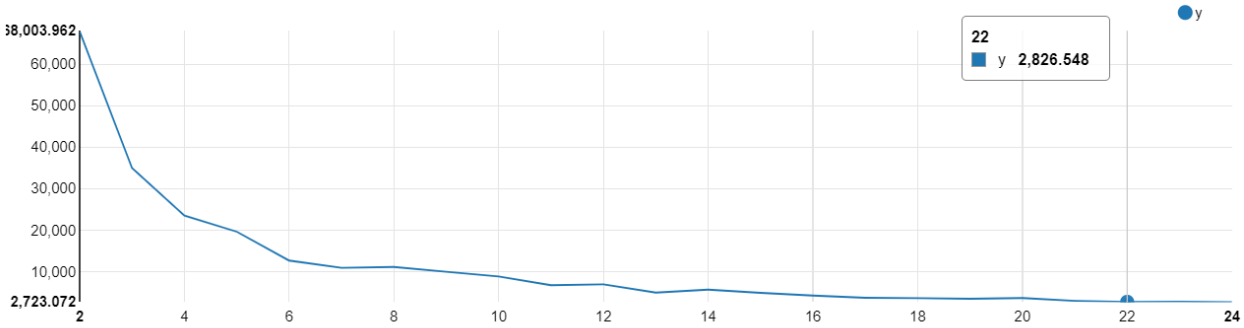
و مراکز کلاستر با روش دوم :

```
Cluster Center 0: [283810.72559366754,280306.2401055409]
Cluster Center 1: [374425.225,354668.4791666667]
Cluster Center 2: [243381.004784689,502410.6937799043]
Cluster Center 3: [392407.52407932014,490464.99716713885]
Cluster Center 4: [521697.50929368025,206893.73605947953]
Cluster Center 5: [664449.490909091,268696.8333333333]
Cluster Center 6: [522690.720379147,369583.518957346]
Cluster Center 7: [309501.76510067115,710736.1174496644]
Cluster Center 8: [461762.1937639198,613833.5812917594]
Cluster Center 9: [519704.51378446113,735828.9047619047]
Cluster Center 10: [634584.1309090909,760803.8545454545]
Cluster Center 11: [583317.0131578947,520427.98245614034]
Cluster Center 12: [681590.469273743,629862.153631285]
Cluster Center 13: [753761.6134020619,474915.9845360825]
```

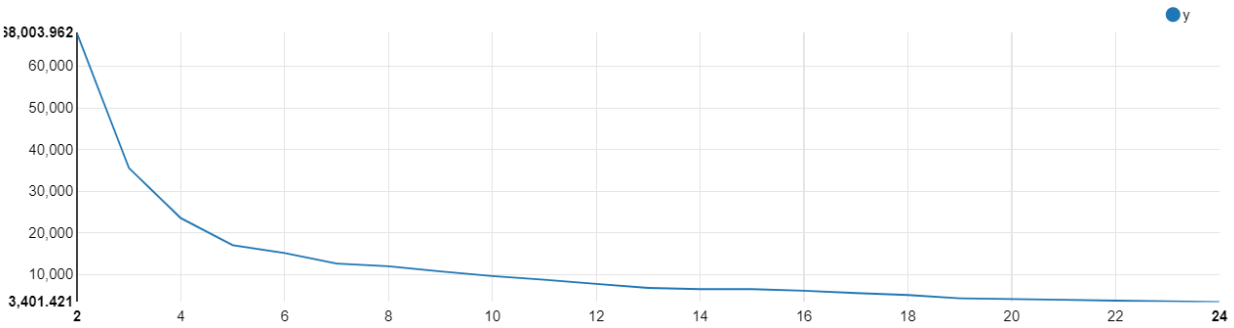
با بررسی جزئی میتوان دریافت که نوع دسته بندی این دو متفاوت است ، پس برای یافتن دسته بندی بهتر ، باید از معیار خطا استفاده کرد:

روش اول 1.862989265480894E13 و روش دوم 2.183727843514322E13 همان طور که مشاهده میشود K-means++ بهتر دسته بندی کرده است. ولی به طور کلی میزان خطا بسیار زیاد است و این به معنای این است که این داده ها قابل تفکیک و جدا پذیری نیستند.

C2=>K-means++:



C2=> Bisecting K-means



تعداد کلاستر بهینه ۱۳ است که مراکز کلاستر آن در روش اول:

```
Cluster Center 0: [14.73571428571428,7.128571428571425]

Cluster Center 1: [11.341999999999999,19.377]

Cluster Center 2: [32.59913793103449,25.075]

Cluster Center 3: [6.730555555555555,3.6722222222222216]
```

```
Cluster Center 4: [8.063953488372093,11.425581395348836]

Cluster Center 5: [32.75833333333333,11.11574074074074]

Cluster Center 6: [32.773571428571415,19.704285714285707]

Cluster Center 7: [21.543333333333326,23.003333333333337]

Cluster Center 8: [20.744186046511626,7.058527131782945]

Cluster Center 9: [33.55799999999999,6.286000000000001]

Cluster Center 10: [6.5219512195121965,21.206097560975607]

Cluster Center 11: [12.094736842105265,26.657894736842103]

Cluster Center 12: [7.279268292682925,25.623170731707315]
```

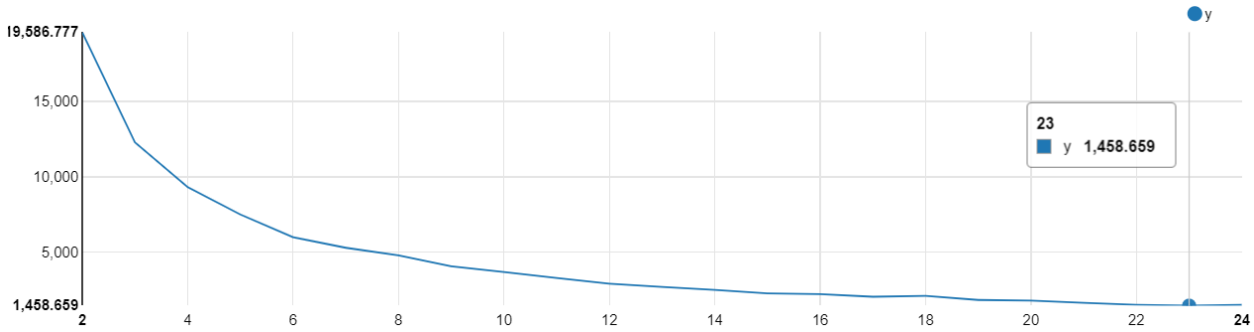
و مراکز کلاستر با روش دوم :

```
Cluster Center 0: [8.904385964912283,4.002631578947367]

Cluster Center 1: [10.698684210526318,9.91118421052632]

Cluster Center 2: [16.863425925925924,7.026388888888891]

Cluster Center 3: [21.43791208791209,7.066483516483516]

Cluster Center 4: [10.555660377358487,18.38867924528302]

Cluster Center 5: [6.466666666666666,22.781666666666673]

Cluster Center 6: [10.958064516129033,26.23870967741935]

Cluster Center 7: [19.509999999999998,22.86666666666667]

Cluster Center 8: [32.49561403508771,6.538596491228072]

Cluster Center 9: [33.180769230769236,11.344230769230771]

Cluster Center 10: [22.56,23.07166666666667]

Cluster Center 11: [32.771014492753615,19.76304347826086]

Cluster Center 12: [32.59913793103449,25.075]
```

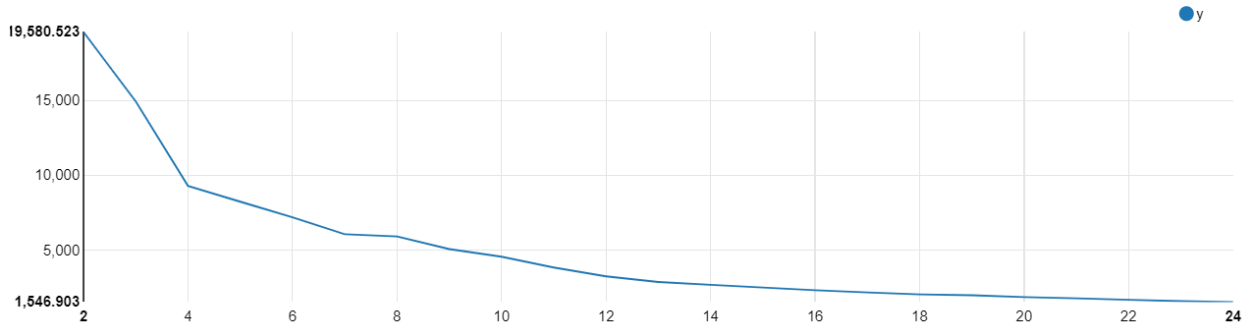
با بررسی جزئی میتوان دریافت که نوع دسته بندی این دو متفاوت است ، پس برای یافتن دسته بندی بهتر ، باید از معیار خطا استفاده کرد:

روش اول 5722.202057896133 و روش دوم 6806.000689768103 پس همان طور که مشاهده میشود K-means++ دسته بندی بهتری انجام داده است.

C3=>K-means++:



C3=> Bisecting K-means



تعداد کلاستر بهینه ۱۲ است که مراکز کلاستر آن در روش اول:

```
Cluster Center 0: [14.940384615384614,19.01153846153846]

Cluster Center 1: [16.59,5.395999999999999]

Cluster Center 2: [6.3547619047619035,19.304761904761904]

Cluster Center 3: [28.003999999999998,8.262]

Cluster Center 4: [20.971739130434777,9.99782608695652]

Cluster Center 5: [28.974999999999998,20.923214285714284]

Cluster Center 6: [22.429166666666667,28.0375]

Cluster Center 7: [14.331081081081082,12.795945945945947]
```

```
Cluster Center 8: [12.419230769230769,24.805769230769233]

Cluster Center 9: [6.535,11.5025]

Cluster Center 10: [23.771052631578947,15.77631578947368]

Cluster Center 11: [20.394736842105257,22.19736842105263]
```

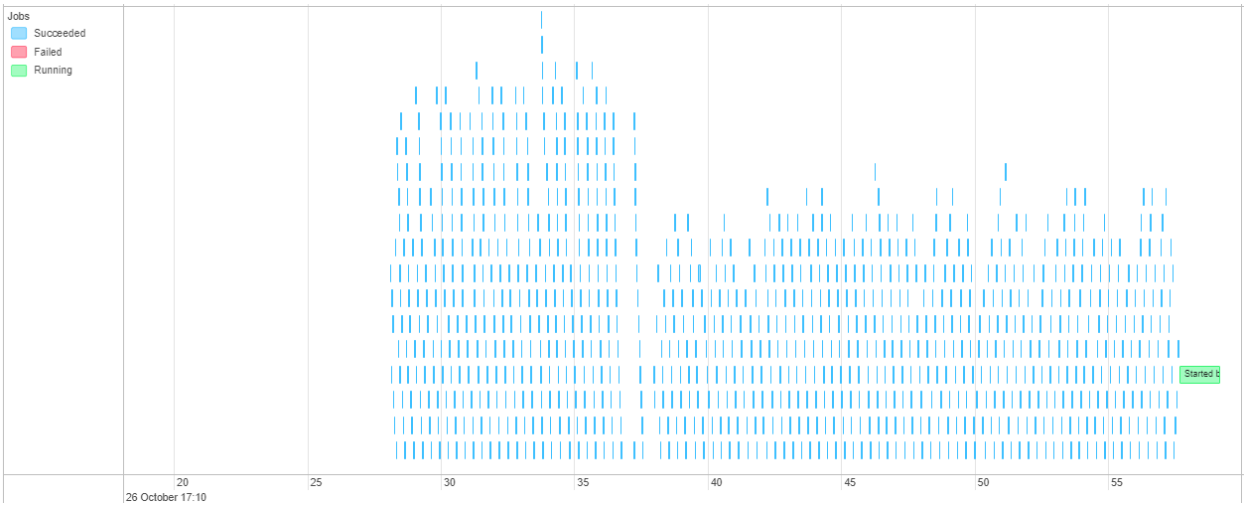
و مراکز کلاستر با روش دوم :

```
Cluster Center 0: [9.004999999999999,11.416666666666666]

Cluster Center 1: [16.16379310344827,12.210344827586205]

Cluster Center 2: [16.345000000000002,5.645]

Cluster Center 3: [6.084090909090909,18.29318181818182]

Cluster Center 4: [14.293103448275861,18.944827586206895]

Cluster Center 5: [12.462499999999999,24.88392857142857]

Cluster Center 6: [23.17058823529411,5.808823529411764]

Cluster Center 7: [22.282812500000002,13.8078125]

Cluster Center 8: [28.606249999999996,12.017187499999999]

Cluster Center 9: [21.352,20.892]

Cluster Center 10: [21.764285714285716,28.054761904761907]

Cluster Center 11: [28.409259259259255,22.40185185185185]
```

با بررسی جزئی میتوان دریافت که نوع دسته بندی این دو متفاوت است ، پس برای یافتن دسته بندی بهتر ، باید از معیار خطا استفاده کرد:

روش اول 3011.059168902624 و روش دوم 3270.611434653738 پس همان طور که مشاهده میشود K-means++ دسته بندی بهتری انجام داده است ولی نتایج تقریبا مشابه است.

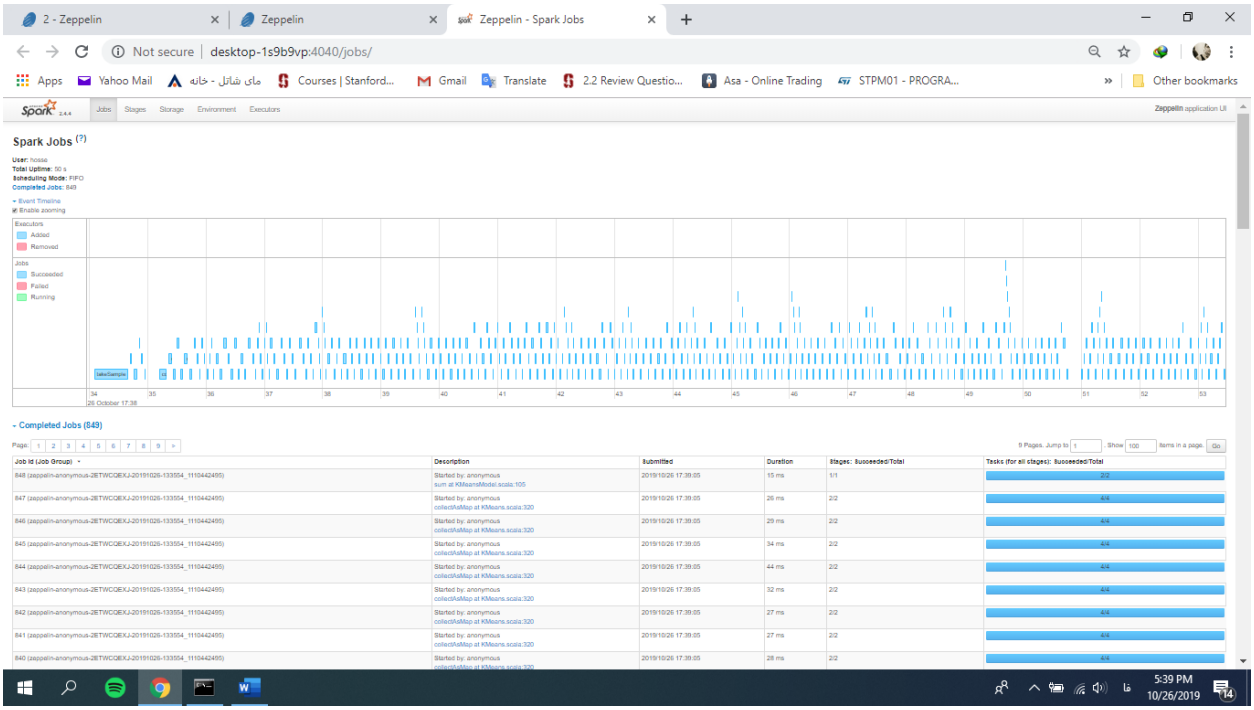
نمودار کلی پردازش برای ۲ روش و ۳ دیتاست به صورت همزمان بر روی چهار هسته :



نمودار پردازش ، روش K-means++ بر روی C1 با ۱۰۰۰ تکرار، با چهار هسته

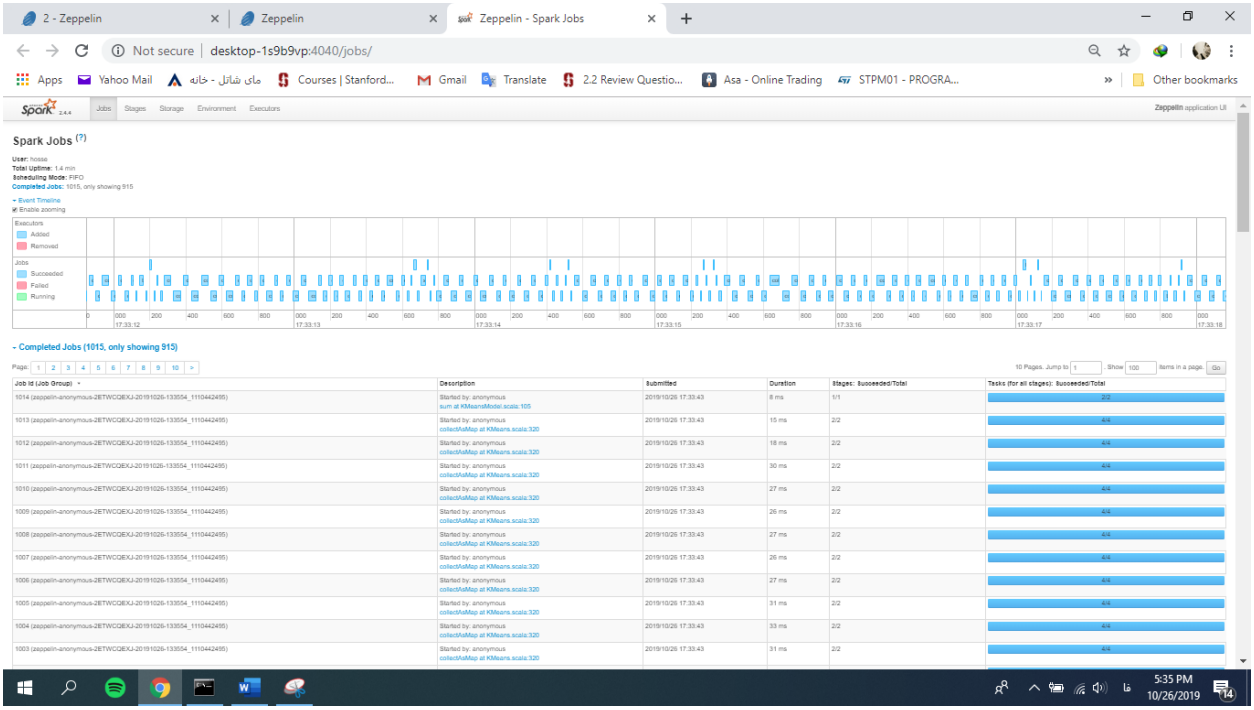
User: hosse
Total Uptime: 50 s
Scheduling Mode: FIFO

و زمان ۵۰ ثانیه



نمودار پردازش ، روش K-means++ بر روی C1 با ۱۰۰۰ تکرار، با یک هسته و زمان۱/۴ دقیقه

User: hosse
Total Uptime: 1.4 min
Scheduling Mode: FIFO



کد بخش دوم :

```
import org.apache.spark._
import org.apache.spark.SparkConf
import org.apache.spark.SparkContext._
import org.apache.spark.rdd.RDD._

import org.apache.spark.mllib.linalg.Vectors

import org.apache.spark.mllib.clustering.{KMeans,
KMeansModel}
import org.apache.spark.mllib.clustering.BisectingKMeans
def parser(line:String)={
   val num =line.split("\t")
   val f1=num(0).toDouble
   val f2=num(1).toDouble
   Vectors.dense(f1,f2)
}
val c1 = sc.textFile("C:/BigData/HW1/C1.txt").map(t=>
Vectors.dense( t.split("\\D+")(1).toFloat , t.split("\\D+")(2).toFloat
) )
val c2 = sc.textFile("C:/BigData/HW1/C2.txt").map(parser)
val c3 = sc.textFile("C:/BigData/HW1/C3.txt").map(parser)
```

```
//KMeans
//for c1
val numIterations = 1000
println("%table\nx \ty")
(2 to 24).map(numClusters=>(numClusters,KMeans.train(c1,
numClusters, numIterations).computeCost(c1)))
                    .foreach{case (x,y) => println(x + "\t" + y)}
val numClusters = 14
val numIterations = 1000

val clusters = KMeans.train(c1, numClusters, numIterations)
val res=clusters.computeCost(c1)

clusters.clusterCenters.zipWithIndex.foreach { case (center, idx)
=>  println(s"Cluster Center ${idx}: ${center}") }
//KMeans
//for c2
println("%table\nx \ty")
(2 to 24).map(numClusters=>(numClusters,KMeans.train(c2,
numClusters, numIterations).computeCost(c2)))
                    .foreach{case (x,y) => println(x + "\t" + y)}
val numClusters = 13
val numIterations = 1000
```

```
val clusters = KMeans.train(c2, numClusters, numIterations)
val res=clusters.computeCost(c2)

clusters.clusterCenters.zipWithIndex.foreach { case (center, idx)
=>  println(s"Cluster Center ${idx}: ${center}") }
//KMeans
//for c3
println("%table\nx \ty")
(2 to 24).map(numClusters=>(numClusters,KMeans.train(c3,
numClusters, numIterations).computeCost(c3)))
                    .foreach{case (x,y) => println(x + "\t" + y)}
val numClusters = 12
val numIterations = 1000

val clusters = KMeans.train(c3, numClusters, numIterations)
val res=clusters.computeCost(c3)

clusters.clusterCenters.zipWithIndex.foreach { case (center, idx)
=>  println(s"Cluster Center ${idx}: ${center}") }
//BisectingKMeans
//for c1
println("%table\nx \ty")
```

```
(2 to 24).map(numClusters=>(numClusters,new
BisectingKMeans().setK(numClusters).run(c1).computeCost(c1)))
                    .foreach{case (x,y) => println(x + "\t" + y)}


val numClusters = 14
val bkm = new BisectingKMeans().setK(numClusters).run(c1)
val result =bkm.computeCost(c1)

bkm.clusterCenters.zipWithIndex.foreach { case (center, idx) =>
println(s"Cluster Center ${idx}: ${center}") }
//BisectingKMeans
//for c2
println("%table\nx \ty")
(2 to 24).map(numClusters=>(numClusters,new
BisectingKMeans().setK(numClusters).run(c2).computeCost(c2)))
                    .foreach{case (x,y) => println(x + "\t" + y)}
```

```
val numClusters = 13

val bkm = new BisectingKMeans().setK(numClusters).run(c2)

val result =bkm.computeCost(c2)


bkm.clusterCenters.zipWithIndex.foreach { case (center, idx) =>
println(s"Cluster Center ${idx}: ${center}") }

//BisectingKMeans

//for c3

println("%table\nx \ty")

(2 to 24).map(numClusters=>(numClusters,new
BisectingKMeans().setK(numClusters).run(c3).computeCost(c3)))

                    .foreach{case (x,y) => println(x + "\t" + y)}


val numClusters = 12

val bkm = new BisectingKMeans().setK(numClusters).run(c3)

val result =bkm.computeCost(c3)
```

```
bkm.clusterCenters.zipWithIndex.foreach { case (center, idx) =>
println(s"Cluster Center ${idx}: ${center}") }
```

توضیحات:

لازم به ذکراست دو فایل ،

HW1_1.json

و

HW1_2.json

در ضمیمه قرار گرفت که

Zeppelin noteهستند

که میتوان در Zeppelin ایمپورت شده و اجرا شوند.

حسین غلامی ۹۷۱۲۳۰۲۱

تکلیف یک کاوش دادگان انبوه