KTH ROYAL INSTITUTE OF TECHNOLOGY
STOCKHOLM

Lab 2 - Report

Author
Emil STÅHL

Author
Selemawit FSHA

October 10, 2021

Review Questions 4 - ID2221

Emil Ståhl and Selemawit Fsha Nguse

October 3, 2021

1 Task 1

Source code

```
 val messages = KafkaUtils.createDirectStream[String, String, StringDecoder, StringDecoder](
    ssc, kafkaConf, topics
  )
  val values = messages.map(x => x._2.split(","))
  val pairs = values.map(x => (x(0), x(1).toDouble))
  pairs.print()

  // measure the average value for each key in a stateful manner
  def mappingFunc(key: String, value: Option[Double], state: State[Double]): (String, Doub
    val sum = value.getOrElse(0.0) + state.getOption().getOrElse(0.0)
    val avg = sum / 2
    state.update(avg)
    (key, avg)
  }
  val stateDstream = pairs.mapWithState(StateSpec.function(mappingFunc _))
  stateDstream.foreachRDD(rdd => {
    val values = rdd.map(x => x).collect()
    values.foreach(println)
  })
----------------------
```

Output

```
(b,5.805191786672103)
(h,13.921922704496573)
(x,6.7641222449056295)
(l,15.324197196282146)
(b,5.902595893336052)
(b,9.451297946668026)
(j,16.099794931979616)
(v,13.921797333508856)
(h,17.960961352248287)
(z,7.980199680353475)
(x,11.882061122452814)
(z,6.490099840176738)
(f,8.911182920236467)
(t,16.578138589284322)
(l,9.662098598141073)
(p,15.458718636502585)
(t,8.789069294642161)
(p,19.72935931825129)
(h,10.480480676124143)
(r,8.144959490687564)
(x,17.441030561226405)
(r,10.57247974534378)
(f,9.455591460118233)
(t,13.89453464732108)
(p,20.364679659125645)
(t,14.94726732366054)
(p,14.182339829562823)
(j,9.549897465989808)
(j,10.774948732994904)
(v,17.460898666754428)
(h,6.740240338062072)
(r,12.78623987267189)
(p,17.59116991478141)
(l,13.831049299070536)
(l,18.915524649535268)
(j,14.887474366497452)
(b,8.725648973334014)
(x,16.220515280613203)
(x,8.610257640306601)
(b,8.862824486667007)
(v,16.730449333377216)
(v,12.365224666688608)
(t,8.97363366183027)
(r,7.393119936335945)
(d,12.330191443338375)
(j,7.943737183248726)
(t,10.486816830915135)
(t,17.743408415457566)
(t,10.371704207728783)
```

```
(g,13.161243030610622)
(o,11.088976598039846)
(u,9.353066786592406)
(y,8.121941653755151)
(e,13.172844021129688)
(u,17.176533393296204)
(w,14.740246000144277)
(s,15.938893364978075)
(w,10.370123000072137)
(k,6.912570943850135)
(e,15.586422010564844)
```

```
(m,18.94425796305368)
(a,9.897134342579209)
(q,9.837227850937385)
(q,12.418613925468692)
(q,6.709306962734346)
(a,16.948567171289604)
(m,18.97212898152684)
(o,6.044488299019923)
(m,21.486064490763418)
(e,18.793211005282423)
(u,17.088266696648102)
(w,15.185061500036069)
(a,10.974283585644802)
(a,7.987141792822401)
(o,10.522244149509962)
(k,14.456285471925067)
(e,11.396605502641211)
(a,12.4935708964112)
(i,12.194124552992333)
(m,14.243032245381709)
(i,15.597062276496167)
(i,12.798531138248084)
(q,15.854653481367173)
(a,17.7467854482056)
(a,19.3733927241028)
(q,15.427326740683586)
(e,7.698302751320606)
(i,7.399265569124042)
(q,13.213663370341793)
(e,4.849151375660303)
```

2 Task 2

In this task we make use of the stateful operation *groupBy()*, the idea is that spark structured streaming keep internal state data as required to compute whole average implicitly.[1]

Source code

```
var df = spark
    .readStream
    .format("kafka")
    .option("kafka.bootstrap.servers", "localhost:9092")
    .option("subscribe", "avg")
    .load()

  // convert the valuer column to string withColumn function
  df = df.withColumn("value",col("value").cast(StringType))

  // Split by , and create two columns
  val value = df.select(
    split(col("value"),",").getItem(0).as("Letter"),
    split(col("value"),",").getItem(1).cast(DoubleType).as("Count")
  )

  val letterCountAvg = value
    .groupBy(col("Letter")).avg("Count")
    .toDF("Letter", "AverageCount")

  val query = letterCountAvg
    .orderBy(desc("AverageCount"))
    .writeStream
    .format("console")
    .outputMode(Complete)
    .start()
-------------------------
```

---

[1]https://spark.apache.org/docs/3.1.2/structured-streaming-programming-guide.html

Output

```
------------------------------------------
Batch: 16
------------------------------------------
+------+------------------+
|Letter|      AverageCount|
+------+------------------+
|     p|12.658812868146805|
|     u|12.637505607895918|
|     j| 12.61829512575117|
|     l|12.582994120307553|
|     m|12.580305069537909|
|     q|12.579287305122495|
|     n|12.577969174977335|
|     g|12.561109837193305|
|     o|  12.5574686940966|
|     b|12.554435483870968|
|     t|12.549298813376483|
|     i|12.542870036101084|
|     y| 12.53919609149456|
|     s|12.533725667722951|
|     r|12.508352144469526|
|     x|12.490719782707107|
|     z|12.478696741854636|
|     w| 12.46781685467817|
|     k| 12.46251673360107|
|     c|12.456904332129964|
+------+------------------+
only showing top 20 rows
```

```
|     b|  12.50632911392405|
|     e|12.503533568904594|
|     j|12.442250740375123|
|     v|12.437043795620438|
+------+------------------+
only showing top 20 rows


-----------------------------------------
Batch: 4
-----------------------------------------
+------+------------------+
|Letter|      AverageCount|
+------+------------------+
|     y|  12.79801559177888|
|     n|12.782543265613244|
|     o|12.764532744665194|
|     m|12.732436472346786|
|     p|12.732075471698113|
|     x|12.720170454545455|
|     r|12.716763005780347|
|     z|12.685435435435435|
|     u|12.648708487084871|
|     g|12.613752743233357|
|     c|12.587583148558759|
|     s|12.586538461538462|
|     t|12.534313725490197|
|     b|12.532188841201716|
|     k|12.520958083832335|
|     w|12.470459518599561|
|     v|12.445396145610278|
|     e|12.438280166435506|
|     i|   12.4304932735426|
|     l|12.402616279069768|
+------+------------------+
only showing top 20 rows


-----------------------------------------
Batch: 5
-----------------------------------------
+------+------------------+
|Letter|      AverageCount|
+------+------------------+
|     o|  12.88126159554731|
|     n|12.773098680075424|
|     p|12.761146496815286|
|     y|12.741451709658069|
|     r|  12.72123076923077|
|     m|12.677399380804953|
|     x|12.665675193337298|
|     u|  12.66564039408867|
|     z|12.648802017654477|
|     v|12.576716417910447|
|     i|  12.57285803627267|
|     c|12.570453134698944|
|     t|12.556321839080459|
|     g|12.546683046683047|
|     b|  12.52397868561279|
|     s|12.501532801961986|
|     q|  12.45476923076923|
|     j|12.452876376988984|
|     w|12.440361445783132|
|     f|12.438888888888888|
+------+------------------+
only showing top 20 rows
```

6

3   Task 3

```scala
import org.apache.spark._
import org.apache.spark.graphx._
import org.apache.spark.rdd.RDD

val vertices = Array(
  (1L, ("Alice", 28)),
  (2L, ("Bob", 27)),
  (3L, ("Charlie", 65)),
  (4L, ("David", 42)),
  (5L, ("Ed", 55)),
  (7L, ("Alex", 55)),
  (6L, ("Fran", 50))
)

val edges = Array(
  Edge(4L, 1L, 1),
  Edge(2L, 1L, 2),
  Edge(5L, 2L, 2),
  Edge(7L, 5L, 3),
  Edge(5L, 6L, 3),
  Edge(3L, 6L, 3),
  Edge(3L, 2L, 4),
  Edge(7L, 6L, 4),
  Edge(2L, 1L, 7),
  Edge(5L, 3L, 8)
)

var vertexRDD = spark.sparkContext.parallelize(vertices)
var edgeRDD = spark.sparkContext.parallelize(edges)
var graph = Graph(vertexRDD, edgeRDD)
case class User(name: String, age: Int, inDeg: Int, outDeg: Int)
```

3.1   Display the names of the users that are at least 30 years old

```scala
println("1. Users that are at least 30 years old: ")
graph
  .vertices
  .filter { case (id, (name, age)) => age > 30 }
  .collect
  .foreach { case (id, (name, age)) => println(s"$name is $age")}
 ----------------

Charlie is 65
```

```
David is 42
Ed is 55
Fran is 50
Alex is 55
```

3.2   Display who likes who.

```
println("2. Who likes who: ")
for (triplet <- graph.triplets.collect){
  println(s"${triplet.srcAttr._1} likes ${triplet.dstAttr._1}")
}
------------------

David likes Alice
Bob likes Alice
Ed likes Bob
Ed likes Fran
Alex likes Ed
Charlie likes Fran
Charlie likes Bob
Alex likes Fran
Bob likes Alice
Ed likes Charlie
```

3.3   If someone likes someone else more than 5 times than that relationship is getting pretty serious, so now display the lovers.

```
println("3. Someone likes someone else more than 5 times: ")
for (triplet <- graph.triplets.filter(t => t.attr > 5).collect) {
  println(s"${triplet.srcAttr._1} loves ${triplet.dstAttr._1}")
}
---------------------

Bob loves Alice
Ed loves Charlie
```

3.4   Print the number of people who like each user

```
Alice is liked by 3 people.
Bob is liked by 2 people.
Charlie is liked by 1 people.
David is liked by 0 people.
Ed is liked by 1 people.
Fran is liked by 3 people.
Alex is liked by 0 people.
```

3.5 Print the names of the users who are liked by the same number of people they like

```
val initialUserGraph = graph.mapVertices{ case (id, (name, age)) => User(name, age, 0, 0) }
val userGraph = initialUserGraph.outerJoinVertices(initialUserGraph.inDegrees) {
  case (id, user, inDegOpt) => User(user.name, user.age, inDegOpt.getOrElse(0), user.outDeg)
}.outerJoinVertices(initialUserGraph.outDegrees) {
  case (id, user, outDegOpt) => User(user.name, user.age, user.inDeg, outDegOpt.getOrElse(0)
}

println("4. Number of people who like each user: ")
for ((id, property) <- userGraph.vertices.collect) {
  println(s"${property.name} is liked by ${property.inDeg} people.")
}
-----------------------
Bob
```

3.6 Find the oldest follower of each user

```
println("5. Names of the users who are liked by the same number of people they like: ")
userGraph.vertices.filter {
  case (id, user) => user.inDeg == user.outDeg
}.collect.foreach {
  case (id, property) => println(property.name)
}
-------------

David is the oldest follower of Alice.
Charlie is the oldest follower of Bob.
Ed is the oldest follower of Charlie.
```