Week 14 Scala – PySpark equivalent programs

Week 14 is based on optimization where we need to allocate resources and hence, we are using shell prompt on cluster to write programs.

Generalized changes that are required in every program

1. To start cmd prompt for PySpark. We PySpark instead of scala-shell.
2. Remove all val, var keyword as python does not have val and var types.
3. Anonymous functions are replaced with lambda in python.
4. Comment is given using # in python instead of // in scala

Note

1. Best practice is to use your own itversity hdfs location in the program for input and output files. You can also use Linux root as shown in video.
2. There are be many ways to get the output for particular problem, we are showcasing one way.
3. Changes are highlighted in yellow.

Problem Statement: We are creating array as one source and second source is a file. We are going to do join on both the sources.

Solution:

Scala Spark Program	PySpark Program
spark2-shell --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 6 --executor-cores 2 --executor-memory 3G --conf spark.ui.port=4063	PySpark --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 6 --executor-cores 2 --executor-memory 3G --conf spark.ui.port=4063
val rdd1 = sc.textFile("bigLogNew.txt") rdd1.getNumPartitions val rdd2 = rdd1.map(x => (x.split(":")(0),x.split(":")(1)) val a = Array(("ERROR",0),("WARN",1)) val rdd3 = sc.parallelize(a) val rdd4 = rdd2.join(rdd3) rdd4.saveAsTextFile("joinResults1")	rdd1=sc.textFile("/user/itv000001/bigLog.txt") rdd1.getNumPartitions rdd2 = rdd1.map(lambda x : (x.split(":")[0],x.split(":")[1])) a = {"ERROR":0, "WARN":1} rdd3 = sc.parallelize(a) rdd4 = rdd2.join(rdd3) rdd4.saveAsTextFile("/user/itv000001/joinResult1")

Specific changes that are required in above program

1. Replace () with [] in python for accessing array
2. Array in scala is changed to dict (Dictionary) in python. Variations are possible, we have showed one way.

Problem Statement: We are creating array as one source and second source is a file. We are going to do join on both the sources. In above program we did normal join and, in this program, we will use broadcast join.

Solution:

Scala Spark Program	PySpark Program
spark2-shell --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 6 --executor-cores 2 --executor-memory 3G --conf spark.ui.port=4063	PySpark --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 6 --executor-cores 2 --executor-memory 3G --conf spark.ui.port=4063

val a = Array(("ERROR",0),("WARN",1))
val rdd3 = sc.parallelize(a)
val keyMap = a.toMap
val bcast = sc.broadcast(keyMap)
val rdd1 = sc.textFile("bigLogNew.txt")
val rdd2 = rdd1.map(x => (x.split(":")(0),x.split(":")(1)))
val rdd4 = rdd2.map(x => (x._1,x._2,bcast.value(x._1)))
rdd4.saveAsTextFile("joinresults2")
	a = {"ERROR":0, "WARN":1}
	# not required
	# not required
	bcast = sc.broadcast(a)
	rdd1=sc.textFile("/user/itv000001/bigLog.txt")
	rdd2 = rdd1.map(lambda x : (x.split(":")[0],x.split(":")[1]))
	rdd4 = rdd2.map(lambda x : (x[0], x[1], bcast.value[x[0]]))
	rdd4.saveAsTextFile("/user/itv000001/joinResult2")

Specific changes that are required in above program

1. Array in scala is changed to dict (Dictionary) in python. Variations are possible, we have showed one way.
2. Second and third line of scala program is not required, hence fourth line broadcast method accepts a as input
3. Replace () with [] in python for accessing array
4. Tuples in python is 0 index and accessed with []

Problem Statement: Join using two data frames

Solution:

Scala Spark Program	PySpark Program
spark2-shell --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 21	PySpark --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 21
val customerDF = spark.read.format("csv").option("header",true).option("inferSchema",true) .option("path","customers.csv").load val orderDF = spark.read.format("csv").option("header",true).option("inferSchema",true) .option("path","orders.csv").load spark.conf.set("spark.sql.autoBroadcastJoinThreshold",-1) val joinedDF = customerDF.join(orderDF,customerDF("customer_id") === orderDF("order_customer_id")) joinedDF.write.csv("output1")	customerDF = spark.read.format("csv").option("header",True)\ .option("inferSchema",True) .option("path","/user/itv000173/customers.csv").load() orderDF = spark.read.format("csv").option("header",True).\ option("inferSchema",True) .option("path","/user/itv000173/orders.csv").load() spark.conf.set("spark.sql.autoBroadcastJoinThreshold",-1) joinedDF = customerDF.join(orderDF,customerDF["customer_id"] == orderDF["order_customer_id"]) joinedDF.write.csv("/user/itv000173/output1")

Specific changes that are required in above program

1. Replace true with True
2. Replace load with load()
3. Replace === with ==

Problem Statement: Join using two data frames and providing orders schema

Solution:

Scala Spark Program	PySpark Program
spark2-shell --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 21	PySpark --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 21
import org.apache.spark.sql.types._ val ordersSchema = StructType(List(StructField("order_id",IntegerType,true), StructField("order_date",TimestampType,true), StructField("order_customer_id",IntegerType,true), StructField("order_status",StringType,true))) val customerDF = spark.read.format("csv").option("header",true) .option("inferSchema",true).option("path","custom ers.csv").load val orderDF = spark.read.format("csv").schema(ordersSchema) .option("header",true).option("path","orders.csv ").load val joinedDF = customerDF.join(orderDF,customerDF("customer_id") === orderDF("order_customer_id")) joined.write.csv("output21")	from PySpark.sql.types import StructType, StructField, IntegerType, TimestampType, StringType ordersSchema = StructType([StructField("order_id",IntegerType(),True), StructField("order_date",TimestampType(),True), StructField("order_customer_id",IntegerType(),True), StructField("order_status",StringType(),True)]) customerDF = spark.read.format("csv").option("header",True).option("inferSchema",True).\ option("path","/user/itv000173/customers.csv").load() orderDF = spark.read.format("csv").schema(ordersSchema).option("header",True).\ option("path","/user/itv000173/orders.csv").load() joinedDF = customerDF.join(orderDF,customerDF.customer_id == orderDF.order_customer_id) joinedDF.write.csv("/user/itv000173/output21")

//----------call take so that it will bring data to driver and may raise OOM joinedDF.take(1000000) //----------increase driver memory and call take again , now no OOM error spark2-shell --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 21 --driver-memory 4G joinedDF.take(1000000)	#----------call take so that it will bring data to driver and may raise OOM joinedDF.take(1000000) #----------increase driver memory and call take again , now no OOM error PySpark --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 21 --driver-memory 4G joinedDF.take(1000000)

Specific changes that are required in above program

1. Replace appropriate imports in python.
2. Replace List() with []
3. Replace true with True
4. Replace load with load(), same for IntegerType(), TimestampType(), StringType()
5. Replace === with ==
6. Replace customerDF("customer_id") === orderDF("order_customer_id") with customerDF.customer_id == orderDF.order_customer_id)

Problem Statement: try repartition and coalesce

Solution:

Scala Spark Program	PySpark Program
val rdd1 = sc.textFile("bigLogFinal.txt") rdd1.getNumPartitions val rdd2 = rdd1.repartition(6) rdd2.count val rdd2 = rdd1.coalesce(6) rdd2.count	rdd1 = sc.textFile("bigLogFinal.txt") rdd1.getNumPartitions rdd2 = rdd1.repartition(6) rdd2.count rdd2 = rdd1.coalesce(6) rdd2.count

Specific changes that are required in above program

1. Remove all val keyword

Problem Statement: Execute Code in production. Create jar and execute using spark-submit in cluster mode. Program is same as week13 except few changes mentioned in video

Solution:

Scala Spark Program	PySpark Program
spark2-submit \ --class LogLevelGrouping \ --master yarn \ --deploy-mode cluster \ --executor-memory 3G \ --num-executors 4 \ wordcount.jar bigLogNew.txt	spark2-submit \ ~~--class LogLevelGrouping \~~ --master yarn \ --deploy-mode cluster \ --executor-memory 3G \ --num-executors 4 \ LogLevelGrouping.py bigLogNew.txt

Specific changes that are required in above program

1. –class is not required, remove it
2. In python we execute python file directly

Problem Statement: Execute Code in production. Create jar and execute using spark-submit in local mode. Program is same as week13 except few changes mentioned in video

Solution:

Scala Spark Program	PySpark Program
spark2-submit \ --class LogLevelGrouping \ --master yarn \ --executor-memory 3G \ --num-executors 4 \ wordcount.jar bigLogNew.txt	spark2-submit \ ~~--class LogLevelGrouping \~~ --master yarn \ --executor-memory 3G \ --num-executors 4 \ LogLevelGrouping.py bigLogNew.txt

Specific changes that are required in above program

1. –class is not required, remove it
2. In python we execute python file directly

Problem Statement: spark sql

Solution:

Scala Spark Program	PySpark Program
spark2-shell --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 11 --conf spark.ui.port=4063	PySpark --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 11 --conf spark.ui.port=4063
val orderDF = spark.read.format("csv").option("inferSchema",true) .option("header",true).option("path","orders.csv").load orderDF.createOrReplaceTempView("orders") spark.sql("select * from orders").show spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(date_format(order_date,'M')) monthnum from orders group by order_customer_id, orderdt order by cast(monthnum as int)").show //----------change the cast from order by spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(cast(date_format(order_date,'M') as int)) monthnum from orders group by order_customer_id, orderdt order by monthnum").show	orderDF = spark.read.format("csv").option("inferSchema",True)\ .option("header",True).option("path","/user/itv000001/orders.csv").load() orderDF.createOrReplaceTempView("orders") spark.sql("select * from orders").show() spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(date_format(order_date,'M')) monthnum from orders group by order_customer_id, orderdt order by cast(monthnum as int)").show() //----------change the cast from order by spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(date_format(order_date,'M')) monthnum from orders group by order_customer_id, orderdt order by cast(monthnum as int)").show()

Specific changes that are required in above program

1. Replace true with True
2. Replace load with load()

3. Replace show with show()
4. Spark sql is same in scala and python

Problem Statement: just add .explain to spark sql from above program

Solution:

Scala Spark Program	PySpark Program
spark2-shell --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 11 --conf spark.ui.port=4063	PySpark --conf spark.dynamicAllocation.enabled=false --master yarn --num-executors 11 --conf spark.ui.port=4063

spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(date_format(order_date,'M')) monthnum from orders group by order_customer_id, orderdt order by cast(monthnum as int)").explain
// It took 3.9 minutes to complete this query - sort aggregate
 	
spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(cast(date_format(order_date,'M') as int)) monthnum from orders group by order_customer_id, orderdt order by monthnum").explain
// It took 1.2 minutes to complete this query - hash aggregate
	spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(date_format(order_date,'M')) monthnum from orders group by order_customer_id, orderdt order by cast(monthnum as int)").explain()
	// It took 3.9 minutes to complete this query - sort aggregate
	
	spark.sql("select order_customer_id, date_format(order_date, 'MMMM') orderdt, count(1) cnt, first(cast(date_format(order_date,'M') as int)) monthnum from orders group by order_customer_id, orderdt order by monthnum").explain()
	// It took 1.2 minutes to complete this query - hash aggregate

Specific changes that are required in above program

1. Replace explain with explain()
2. Spark sql is same in scala and python

Problem Statement: Connecting to external resources

Solution:

Scala Spark Program	PySpark Program
spark-shell --driver-class-path /usr/share/java/mysql-connector-java.jar	PySpark -jars /usr/share/java/mysql-connector-java.jar
val connection_url ="jdbc:mysql://cxln2.c.thelab-240901.internal/retail_db" val mysql_props = new java.util.Properties mysql_props.setProperty("user","sqoopuser") mysql_props.setProperty("password","NHkkP876rp") val orderDF = spark.read.jdbc(connection_url,"orders",mysql_props) orderDF.show()	connection_url ="**jdbc:mysql://cxln2.c.thelab-240901.internal/retail_db**" orderDF = spark.read \ .jdbc(connection_url, **"orders"**, properties={**"user"**: **"sqoopuser"**, **"password"**: **"NHkkP876rp"**}) orderDF.show()

Specific changes that are required in above program

1. Giving properties is different