Cousera Cloud Computing Capstone Task 2 Report – Stuart Chaney

Video link ---------- https://youtu.be/MUEdYT7SAn0

Approach to all questions:

Firstly I used the cleaned csv files from task 1, the first being a full representation of the on_time CSV and the seconds being just 2008 for Group 3 Question 2. Answers to the questions were done using PySpark receiving input from Kafka. Amazon's Elastic Map Reduce (EMR) was used to build the cluster including Spark, some custom bootstrapping and steps to ensure the required software was installed. This included Zookeeper and Kafka along with their configurations. The cluster consisted of a single Master running r3.2xlarge instance types as I found the extra memory was required. To optimize things I switched from Cassandra to DynamoDB for task 2 as it much more accessible when using EMR. The core instance types (cluster nodes) were running c3.xlarge instance types as I found the compute and improved network performance was beneficial. After some experimentation I found matching the executors to the amount of cluster instances worked best. My submit line (example):

```
spark-submit --packages=" org.apache.spark:spark-streaming-kafka_2.10:1.5.2,org.apache.spark:spark-streaming-kafka-assembly_2.10:1.5.2 " --master=yarn-client --driver-cores=4 --driver-memory=8g --num-executors=5 --executor-cores=4 --executor-memory=2g ./sparkApp.py
```

Days were spent on getting Kafka talking to Spark in an efficient manor, my solution ended up being to rate limit the stream using the pv command.

```
cat /mnt/tmp/sparkOntimeAllYears | pv -L 5m | bin/kafka-console-producer.sh --broker-list=localhost:9092 --topic=capstone
```

SQL Context.

I was fascinated by the fact you could stream data in and perform SQL style queries on it. Using this method makes source code far more readable providing the reader knows some SQL and **allows Spark to optimize the logic (Spark Engine handles ordering).** First I started with a standard spark context and from that I create the streaming context along with a windows size of 3 seconds (after much experimentation). Using the KafkaUtils packages I take the input stream through a simple map routine stripping off the carriage and creating my RDD. The next step splits the csv lines into the appropriate fields depending on the question and performs a reduce by key. Then it iterates through each item converting them to Rows (from the pyspark.sql package). From this I create a new SQL context DataFrame which then allows me to utilize Sparks amazing SQL module. Source code snipits:

```
ssc = StreamingContext(sc, 3)
kvs = KafkaUtils.createStream(ssc, 'x.x.x.x:2181', "spark-streaming-consumer", {'capstone': 4})
lines = kvs.map(lambda x: x[1][:-1])  #remove \n
def splitStream(csvline):
    f = csvline.split(",")
    return (f[1], f[7] + ',' + f[13]) #grab the fields I want
fields = lines.map(splitStream)
reducedFields = fields.reduceByKey(lambda a, b: b) #reduce by key
reducedFields.foreachRDD(process) #process results
```

…(snip to example process function)

```
    sqlContext = getSqlContextInstance(rdd.context)
    def mapFields(tup):
        f = tup[1].split(",")
        return (Row(Airport = tup[0], AirlineID = f[0], DepDelay = f[1]))
    rowRdd = rdd.map(mapFields)
    allDataFrame = None
    try:
        allDataFrame = sqlContext.createDataFrame(rowRdd)
        allDataFrame.registerTempTable("g2q1")
    except:
        print ".........no Data"
        return
    allDataFrame = sqlContext.sql("""
        …SQL GOES HERE (see answers section below)
```

Group 1 Questions:

Using the technique explained above I used the following SQL context queries to generate the results. I confirmed the results by comparing with task 1. I ran question 2 twice, once calculating with ArrDelay and again with ArrDel15. SQL logic is included in the table below:

Q1:

```
SELECT x.Airport,
COUNT(*) as c
FROM (SELECT Airport
FROM airportX
UNION ALL
SELECT Airport
FROM airportY) x
GROUP BY x.Airport
```

# of flights	Airport
12020931	"ORD"
11301229	"ATL"
10562404	"DFW"
7574328	"LAX"
6494512	"PHX"
6169795	"DEN"
5491596	"DTW"
5400340	"IAH"
5073589	"MSP"
5050872	"SFO"

Q2: (early arrivals)

```
SELECT AirlineID,
SUM(ArrDelay)
FROM availableflights
GROUP BY AirlineID
```

On Time perf	AirlineID
-264258.0	"19690"
60319.0	"19391"
175282.0	"19678"
328150.0	"20295"
1262537.0	"20312"
1569537.0	"20384"
1751640.0	"20436"
2713127.0	"20363"
4552841.0	"20404"
5313381.0	"19707"

Q2b:(arrivals within 15m)

```
SELECT AirlineID,
SUM(ArrDelay15)
FROM availableflights
GROUP BY AirlineID
```

On Time perf	AirlineID
4558.0	"19391"
10651.0	"20295"
14274.0	"19678"
16209.0	"19690"
40913.0	"20312"
58368.0	"20384"
62730.0	"20436"
88663.0	"20363"
143262.0	"19707"
147247.0	"20404"

Group 2 Questions:

The SQL logic for group 2 is shown below, please refer to the video for the results and queries.

Q1:

```
SELECT
    Airport,
    AirlineID,
    SUM(DepDelay) AS delay
FROM g2q1
GROUP BY Airport, AirlineID
```

Q2:

```
SELECT
    Origin,
    Destination,
    SUM(DepDelay) AS delay
FROM g2q2
GROUP BY Origin, Destination
```

Q3:

```
SELECT
    SrcDest,
    AirlineID,
    SUM(ArrDelay) AS delay
FROM g2q3
GROUP BY SrcDest, AirlineID
```

Group 3 Question 2:

SQL logic in 3 states as follows (key points in bold). I ran out of time to come up with an elegant solution so ended up streaming the data in chucks that allowed for the calculations to succeed. Writing backwards and forwards to DynamoDB was my initial solution but it turned out to slow things down too much.

```
SELECT
xflights.FlightDate as xFlightDate,
xflights.FlightNum as xFlightNum,
xflights.Origin as xOrigin,
xflights.DepTime as xDepTime,
xflights.Dest as xDest,
xflights.ArrTime as xArrTime,
xflights.ArrDelay as xArrDelay,
yflights.FlightDate as yFlightDate,
yflights.FlightNum as yFlightNum,
yflights.Origin as yOrigin,
yflights.DepTime as yDepTime,
yflights.Dest as yDest,
yflights.ArrTime as yArrTime,
yflights.ArrDelay as yArrDelay
FROM xflights
INNER JOIN yflights ON
xflights.Dest = yflights.Origin
WHERE
dayofyear(yflights.FlightDate) =
dayofyear(xflights.FlightDate)+2
```

```
SELECT
        mt.xFlightDate,
        mt.xFlightNum,
        mt.yFlightNum,
        mt.xOrigin,
        mt.xDest,
        mt.yDest,
        mt.xArrDelay,
        mt.yArrDelay
FROM availableFlights mt INNER JOIN
(   SELECT xFlightDate, xOrigin, xDest,
yDest, MIN(xArrDelay + yArrDelay)
MinyArrDelay
FROM availableFlights
GROUP BY xFlightDate, xOrigin, xDest,
yDest
) t ON mt.xFlightDate = t.xFlightDate
       AND mt.xOrigin = t.xOrigin
       AND mt.xDest = t.xDest
       AND mt.yDest = t.yDest
       AND (mt.xArrDelay +
mt.yArrDelay) = t.MinyArrDelay
```

```
SELECT
bestAvailableFlightsTmp.xFlightDate,
availableFlights.xFlightNum,
availableFlights.xOrigin,
availableFlights.xDepTime,
availableFlights.xDest,
availableFlights.xArrTime,
availableFlights.yFlightDate,
bestAvailableFlightsTmp.yFlightNum,
availableFlights.yOrigin,
availableFlights.yDepTime,
availableFlights.yDest,
availableFlights.yArrTime,
bestAvailableFlightsTmp.xArrDelay,
bestAvailableFlightsTmp.yArrDelay
FROM bestAvailableFlightsTmp
INNER JOIN availableFlights
ON bestAvailableFlightsTmp.xFlightDate =
availableFlights.xFlightDate
AND bestAvailableFlightsTmp.xOrigin =
availableFlights.xOrigin
AND bestAvailableFlightsTmp.xDest =
availableFlights.xDest
AND bestAvailableFlightsTmp.yDest =
availableFlights.yDest
AND bestAvailableFlightsTmp.xArrDelay =
availableFlights.xArrDelay
AND bestAvailableFlightsTmp.yArrDelay =
availableFlights.yArrDelay
```

Results Snipit:

***** bestAvailableFlightsTmp_df count = 60541399

xFlightDate	xFlightNum	yFlightNum	xOrigin	xDest	yDest	xArrDelay	yArrDelay
2008-01-01	1680	460	ABQ	ATL	DCA	-26.0	-4.0
2008-01-01	1680	4310	ABQ	ATL	ILM	-26.0	3.0
2008-01-01	332	1528	ABQ	DEN	LAS	-7.0	-2.0
2008-01-01	2284	3323	ABQ	DFW	MLU	-18.0	-6.0
2008-01-01	2874	767	ABQ	LAS	MHT	-10.0	-2.0
2008-01-01	624	4772	ABQ	MSP	CPR	1.0	7.0
2008-01-01	624	5692	ABQ	MSP	SDF	1.0	-19.0
2008-01-01	5476	6415	ACV	SFO	PMD	0.0	-19.0
2008-01-01	4660	4554	AEX	ATL	VLD	-24.0	4.0
2008-01-01	3206	1774	AEX	DFW	BDL	7.0	-5.0
2008-01-01	4839	1103	ALB	ATL	MLB	7.0	-8.0
2008-01-01	1101	997	ALB	MCO	PIT	15.0	-6.0
2008-01-01	1769	1569	ALB	PHL	BWI	-4.0	-17.0
2008-01-01	3432	3861	AMA	DFW	VPS	-12.0	-5.0
2008-01-01	12	538	AMA	LAS	BOI	-13.0	24.0
2008-01-01	12	380	AMA	LAS	CLE	-13.0	-12.0
2008-01-01	12	1011	AMA	LAS	PHL	-13.0	-21.0
2008-01-01	12	1450	AMA	LAS	PHL	-13.0	-21.0
2008-01-01	846	1712	ANC	MSP	GRR	-15.0	-6.0
2008-01-01	634	587	ANC	PHX	GEG	23.0	-22.0

How the different stacks contrasted:

Hadoop map reduce has maybe better documentation and support out there but it's pretty clear that Spark is becoming the preferred method. It's a number of times faster once you have it running the way you would like. It's far more important with Spark streaming to ensure your settings and logic are appropriate depending on the input stream. Fortunately Spark is very customizable and feature rich so it can be configured to suite the input stream.

I think the bottom line is Spark will replace Hadoop over time as it can achieve everything it does, only faster and more efficiently thanks to better resource allocation and better use of cheap RAM.