Coursera Cloud Computing Capstone Project
Task 2: Stream Processing with Spark Streaming

Introduction

The report describes the work conducted for and the results of Task 2 (Data Extraction, Batch Processing with Hadoop) of the Coursera Cloud Computing Capstone Project. The objective of the capstone project is to use cloud computing resources to solve real-world challenges using relatively large data sets applying the knowledge and skills developed from prior courses in the Coursera Cloud Computing certification program. Work for Task 1 of the project included extracting and cleaning the aviation data set and storing the results in HDFS, among others. The goal of the Task 2 is to process streaming data from the data cleaned in Task 1, to answer a set of questions using Spark Streaming, and to store the results in DynamoDB. The data used in the project is sourced from the US Department of Transportation, specifically airline on-time data available from the public snapshot (snap-37668b5e).

Systems Integration

Amazon Web Services (AWS) were used in this project to provision a Hadoop cluster with a name node (master) and two worker nodes (slaves). Several Hadoop components (software stack) were used to host and process the provided data and to execute queries against the proceed data (clean data) to generate results for Task 2 questions using Spark Streaming. Figure 1 depicts the systems integration details for Task 2 and further described in steps 1 to 10 below:

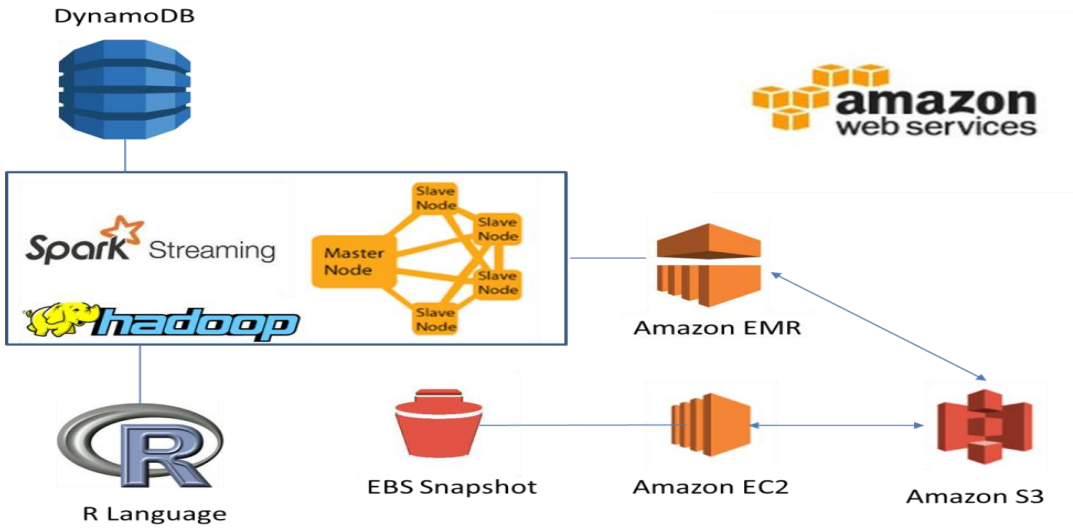


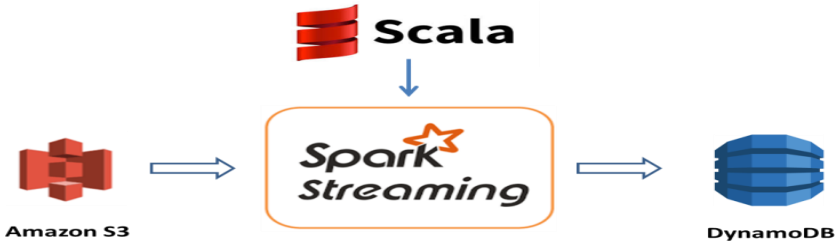
Figure 1: The Systems Integration for Task 2.

(1) A virtual server Amazon EC2 instance was created in Amazon Elastic Cluster (EC2).
(2) In the EC2 instance, a new EC2 volume was restored from snapshot (snap-37668b5e) then attached to the instance.
(3) The aviation folder of the restored snapshot was copied into an S3 bucket.

(4) Amazon Elastic Map Reduce (EMR) Hadoop m3.xlarge three instances cluster was provisioned with a name node and two worker nodes. The cluster includes Ganglia 3.7.2, Hadoop 2.7.1, Hive 1.0.0, Hue 3.7.1, Mahout 0.11.0, Pig 0.14.0, and Spark 1.6.0.
(5) The aviation data in S3 bucket was copied into HDFS, unzipped, processed, and cleaned using R.
(6) The cleaned data is copied into S3.
(7) Spark streaming is used to directly access data (using EMRFS) in S3 without copying it into HDFS, thus streaming data into the application.
(8) Scala API is used to execute quires submitted to the Spark master node (server) on the data streamed from S3.
(9) The queries results sets are stored into HDFS and DynamoDB.
(10) The results sets are read into R, which is used to generate the graphs (analytical presentation).

Approach and Algorithms to Answer Questions

The solution approach involves using Apache Spark in Amazon EMR by utilizing the Spark Streaming server on the Spark main (master) node to execute queries on data streamed from S3. The application interface was accomplished using Scala (spark-shell), which employees Scala native support in Spark. Figure 2 depicts the solution approach.



For each of the questions, the algorithm involves importing Spark Streaming classes and initiating a StreamingContext object. This is the main entry point for all streaming functionality:

```
import org.apache.spark._
import org.apache.spark.streaming._
import org.apache.spark.streaming.StreamingContext._
val conf = new SparkConf().setMaster("local[2]").setAppName("Cloud_Task2")
val ssc = new StreamingContext(conf, Seconds(1))
```

The cleaned data files in S3 can be directly linked to the streaming object:

```
val parquetFile = hiveContext.parquetFile("s3://us-east-1&bucket=ebs-
cloudcomputingcapstone&prefix=aviation/airline_ontime/”)
```

Registering the Parquet files as tables which then can be used as source for the queries using SQL like statements using DataFrames and SQL operations on streaming data. This involves creating a SQLContext using the SparkContext that the StreamingContext is using by creating a lazily instantiated singleton instance of SQLContext where each RDD is converted to a DataFrame, registered as a

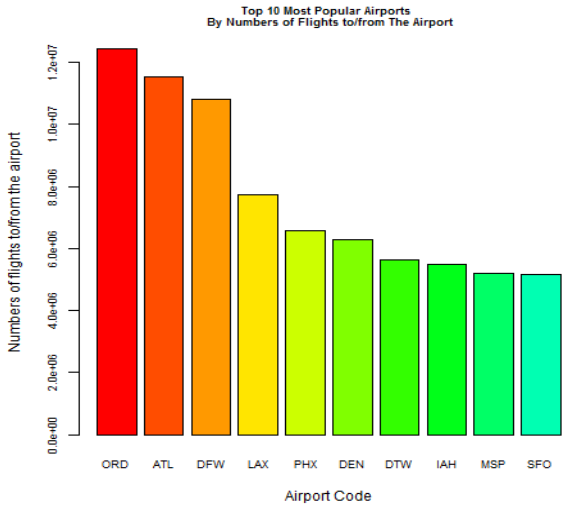
temporary table and then queried using SQL. For example below is a query for ranking the top 10 airlines by on-time arrival performance:

```
parquetFile.registerTempTable("airline_ontime")
val topAirlines = sqlContext.sql("select UniqueCarrier,
sum(arrdelay)/count(arrdelay) as avgDelay
from airline_ontime where cancelled = 0 group by UniqueCarrier order by
avgDelay asc limit 10")
topAirlines.rdd.saveAsTextFile(s"$OutputLocation/topAirlines")
```

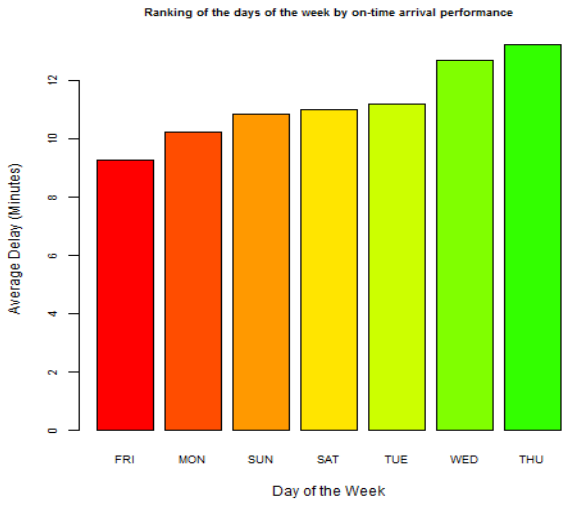
Results

Group 1

Question 1.1: Rank the top 10 most popular airports by numbers of flights to/from the airport.



Question 1.3: Rank the days of the week by on-time arrival performance.



Group 2

Question 2.1: For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.

		Airport Code				
		SRQ	**CMH**	**JFK**	**SEA**	**BOS**
Top-10 carriers RANK	1	US	NW	AA	AS	PA
	2	DL	AA	UA	PS	ML
	3	AA	ML	PA	AA	AA
	4	NW	WN	DL	AL	DL
	5	TW	DL	AL	HP	NW
	6	UA	TW	EA	NW	EA
	7	EA	PI	US	DL	CO
	8	ML	EA	TW	US	AL
	9	CO	US	NW	CO	US
	10	XE	AL	HP	UA	TW

*Unique Carriers Code in white cells

Question 2.2: For each airport X, rank the top-10 airports in decreasing order of on-time departure performance from X.

		Origin Airport Code				
		SRQ	**CMH**	**JFK**	**SEA**	**BOS**
Top-10 Destination Airport Code	1	BWI	MSP	TUS	ONT	SJC
	2	MEM	BNA	ISP	SJC	LGA
	3	MSP	SDF	PDX	PSP	MDW
	4	MCO	DTW	UCA	SNA	SLC
	5	RDU	CLE	BQN	OGG	BDL
	6	BNA	DFW	PSE	LGB	SJU
	7	TPA	MEM	STT	GEG	BNA
	8	IAH	ATL	SNA	CVG	CVG
	9	CLT	MCO	BDL	SLC	RDU
	10	DCA	IND	SRQ	SIT	MKE

Question 2.4: For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.

Origin	Destination	Mean Arrival Delay (Minutes - rounded)
LGA	BOS	1.4
BOS	LGA	3.5
OKC	DFW	4.2
MSP	ATL	6.7

Group 3:

Question 3.2: Tom wants to travel from airport X to airport Z with stop at airport Y where flight Y-Z must depart two days after flight X-Y, flights departing airport X are before 12:00 PM local time and airport Y after 12:00 PM local time, and Tom wants to arrive at each destination with as little delay as possible (assuming the actual delay of each flight is known).

	Airport			1st Segment (X->Y)		2nd Segment (Y->Z)	
DATE	X	Y	Z	Carrier	Flight #	Carrier	Flight #
2008-04-03	BOS	ATL	LAX	DL	661	DL	885
2008-07-09	PHX	JFK	MSP	B6	178	NW	609
2008-01-24	DFW	STL	ORD	AA	314	AA	640
2008-05-16	LAX	MIA	LAX	AA	280	AA	456

System and Application Optimizations

The system level optimization mainly involved using Spark Streaming which optimizes I/O operations through the use of memory to store the data compared to the disk. On the application level, optimization was mainly achieved (1) via elimination of data processing that is needed to answer the questions, for example by processing only a subset of columns in airline_ontime data set, and (2) via optimization of queries to obtain the results. The latter was achieved through the use of the SQLContext within StreamingContext which creates lazily instantiated singleton instance RDDs are converted into data frames within temporary tables and queried using optimized SQL. Note that each SQL query in this case uses the temp table to reduce the I/O cost for the query. Spark in EMR uses EMRFS to directly stream data from S3 without the need to copy it into HDFS first. Further, the results sets stored in DynamoDB employing local indexes and range keys.

Discussion

Overall the results make sense and are useful in several ways, for example in scheduling and route optimization as shown in the results for question 3.2. Performance ranking of carriers and airports using basic statistical measures as shown in other answers also are examples of how the results can be useful. Although the results can be achieved using many systems, the key advantage of cloud computing shows through the use of big data technology when the data sets are huge. In Task 1, results were obtained using Hadoop Pig, which provides a rapper for MapReduce, a distributed processing on the Hadoop cluster but a batch processing paradigm. Apache Spark Streaming, also a distributed processing framework in the Hadoop ecosystem, is proving to be an enticing engine by increasing job performance and development velocity for certain workloads compared to MapReduce. The main reason is Spark uses a directed acyclic graph (DAG) execution engine creating a more efficient query plan for data transformations employing in-memory resilient distributed datasets (RDDs). The latter keeps intermediates, inputs, and outputs in memory instead of on disk resulting in substantial performance advantage for certain workloads when compared to Hadoop MapReduce.

Video Link: https://youtu.be/hejBXWxYu7k