CS 598 Cloud Computing Capstone Task 2 Report

1. Overview of the approaches to solve the questions

In Task 2, Java, Zookeeper, Kafa, Spark, S3, Cassandra are setup and used as illustrated in below diagram.

- To installer ZooKeeper cluster, the steps in link are followed. The Zookeeper a centralized service to handle distributed synchronization, which will be used by Kafka to handle multiple brokers to ensure higher availability and failover handling. In case of failure of one Zookeeper, a Zookeeper cluster has been setup to avoid a single point of failure.
- To install Kafka cluster, the steps in link are followed. Kafka is used to handle real-time data streams. The producer in Kafka will publish message with specified topics, and consumer in Kafka would receive the message using the the same topic.
- The AWS S3 bucket has been created, which is used for Spark checkpoint.
- To install Spark, the steps in link are followed. Notice that based on compatibility of the Spark Cassandra Connector, the version of Spark has been installed as 2.1 rather than the latest version.
- To install Cassandra, the steps in link are followed.
- To setup Spark Cassandra Connection, the steps in link are followed.

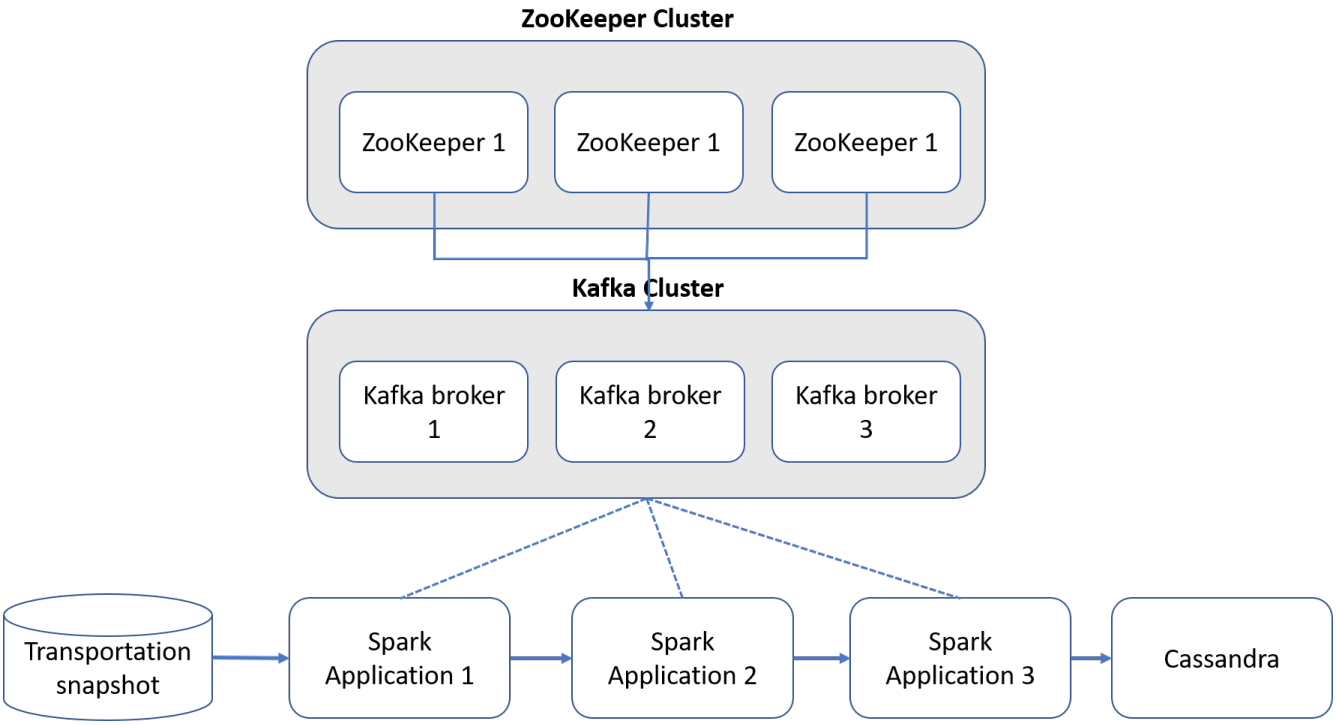

Figure 1. System structure of Task 2

The EBS snapshot ID for the transportation dataset is given (snap-e1608d88). To use the data, a new volume was created with size 100 GB (a default 8 GB was chosen, however it has not enough space to store the extracted dataset), availability zone us-east-1b, and the given snapshot id. Then, a new instance is launched and attached with the volume created.

To mount the attached volume, follow link and do:

```
sudo mkdir /data
sudo mount /dev/xvdf /data
```

To produce the streaming data, the files from *airline_ontime* folder are extracted and put into a local input folder, using bash scripts:

```
for FILE in `ls $DATA_FOLDER/airline_ontime/*/*.zip`; do
```

```
    for CSV_NAME in `unzip -l $FILE  | grep csv | tr -s ' ' | cut -d ' ' -f4`; do
        unzip -p $FILE $CSV_NAME > $TARGET_FOLDER/$CSV_NAME
        echo "$CSV_NAME has been unzipped into $TARGET_FOLDER"
    done
done
```

While unzipping the data into the local folder, the 1st Spark application is monitoring the folder using the following code:

```
lines = ssc.textFileStream(sys.argv[1])
```

With the streaming data monitored, the 1st Spark application would rearrange the data and use Kafka Producer to send the data to Kafka Cluster with topic “input”.

The 2nd Spark application will always use Kafka Consumer to receive the data with topic “input”, rearrange the data for the need of each question, and either save to output files (as in Part 1 questions), or use Kafka Producer to send the data to the cluster for further processing.

The 3rd Spark application is mainly used for talking to Cassandra. After receiving the data sent from the 2nd Spark application, it saves the data into Cassandra tables. Then, in cqlsh of Cassandra, queries can be conducted to the streaming data results.

2. Steps and results of each question

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

The 1st Spark application is used for producing streaming data through Kafka Producer:

```
spark-submit --master local[4] --conf spark.streaming.backpressure.enabled=true --conf
spark.streaming.receiver.maxRate=4000 project2/python/send_to_kafka.py input
```

The 2nd Spark application setup the S3 bucket as checkpoint:

```
ssc.checkpoint("s3a://cloudcapstones3/checkpoints/question11/"
```

Also, it uses updateStateByKey API to update the count of each airpoint, and sort the counts by partition, and by all the data:

```
sorted =airportsCounted.transform(lambda rdd: rdd.mapPartitions(TopTenData)).transform(lambda rdd:
rdd.sortByKey(False))
```

The RDD data will then be save to local file.

To start the 2nd Spark application:

```
$SPARK_HOME/bin/spark-submit --master local[4] --conf park.streaming.kafka.maxRatePerPartition=250
--packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.1.0,org.apache.hadoop:hadoop-aws:2.7.3
Project2/python/streaming_top_airports.py 172.31.5.71:9092,172.31.60.234:9092,172.31.53.23:9092
topten_airports.log
```

The result is:
[(12446097, u'ORD'), (11537401, u'ATL'), (10795494, u'DFW'), (7721141, u'LAX'), (6582467, u'PHX'), (6270420, u'DEN'), (5635421, u'DTW'), (5478257, u'IAH'), (5197649, u'MSP'), (5168898, u'SFO')]

Question 1.2 Rank the top 10 airlines by on-time arrival performance

Similar to Question 1.1, the 1st and 2nd Spark applications are used for producing streaming data and process the data respectively.
The result is:
[(-1.01, u'HA'), (1.16, u'AQ'), (1.45, u'PS'), (4.75, u'ML'), (5.35, u'PA'), (5.47, u'F9'), (5.56, u'NW'), (5.56, u'WN'), (5.74, u'OO'), (5.87, u'9E')

Question 2.1 For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.
Compare to Part 1 questions, queries need to be conducted in Cassandra for the streaming data. Therefore the 3rd Spark application is used for saving the results into Cassandra tables.

The 2nd Spark application counts the average departure delays for each airport-carrier pair, then sort to have an ordered list of top ten carrier for each airport. The resulted data are sent through Kafka producer to the 3rd Spark application.

A table is created in Cassandra cqlsh:

```
apache-cassandra-3.11.2/bin/cassandra
apache-cassandra-3.11.2/bin/cqlsh $(hostname -i)

create keyspace result WITH replication = {'class': 'SimpleStrategy', 'replication_factor': 1 };
create table result.airport_carrier_departuredelay (
  airport text,
  carrier text,
  departure_delay decimal,
  PRIMARY KEY(airport, departure_delay, carrier)
);
```

The 3rd Spark application receive the data and update the data in Cassandra tables:

```
$SPARK_HOME/bin/spark-submit --master local[4] --conf park.streaming.kafka.maxRatePerPartition=250
--packages org.apache.spark:spark-streaming-kafka-0-8_2.11:2.1.0,org.apache.hadoop:hadoop-
aws:2.7.3,datastax:spark-cassandra-connector:2.4.0-s_2.11
Project2/python/streaming_top_carriers_by_airports_to_cassandra.py
172.31.5.71:9092,172.31.60.234:9092,172.31.53.23:9092 topten_airports.log
```

To query the result in Cassandra cqlsh:

```
select * from result.airport_carrier_departuredelay where airport = 'CMI' order by departure_delay limit 10;
```

The given query result is:

airport	dep_delay	carrier
JFK	5.06	RU
JFK	5.96	UA
JFK	8.20	CO
JFK	8.74	DH
JFK	10.08	AA
JFK	11.12	B6
JFK	11.52	PA
JFK	11.63	NW
JFK	11.98	DL
JFK	12.41	AL

airport	dep_delay	carrier
SEA	2.70	00
SEA	4.72	PS
SEA	5.12	YV
SEA	6.01	AL
SEA	6.34	TZ
SEA	6.43	US
SEA	6.49	NW
SEA	6.53	DL
SEA	6.85	HA
SEA	6.94	AA

airport	dep_delay	carrier
BOS	2.12	RU
BOS	3.06	TZ
BOS	4.44	PA
BOS	5.73	ML
BOS	7.20	EV
BOS	7.24	NW
BOS	7.44	DL
BOS	8.62	AL
BOS	8.68	US
BOS	8.73	AA

airport	dep_delay	carrier
CMH	3.49	DH
CMH	3.51	AA
CMH	4.04	NW
CMH	4.36	ML

airport	dep_delay	carrier
SRQ	-0.38	TZ
SRQ	-0.08	RU
SRQ	3.40	YV
SRQ	3.64	AA

CMH | 4.71 | DL CMH | 5.20 | PI CMH | 5.93 | EA CMH | 5.99 | US CMH | 6.02 | AL CMH | 6.10 | RU	SRQ | 3.95 | UA SRQ | 3.96| US SRQ | 4.30 | TW SRQ | 4.85 | NW SRQ | 4.86 | DL SRQ | 5.03 | XE	

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

Similar to Question 2.1, the 2nd Spark is used for processing the data, and the 3rd Spark application is used for saving the results into Cassandra.

The given query result is:

airport | dep_delay | airport_to ---+---+--- JFK | -10.5| SWF JFK | 0.0 | MYR JFK | 0.0 | ABQ JFK | 0.0 | ISP JFK | 0.0 | ANC JFK | 1.91 | UCA JFK | 3.21 | BGR JFK | 3.60 | BQN JFK | 4.40 | CHS JFK | 4.50 | STT	airport | dep_delay | airport_to ---+---+--- SEA | 0.0 | EUG SEA | 1.0 | PIH SEA | 2.65 | PSC SEA | 3.87 | CVG SEA | 4.26 | MEM SEA | 5.17 | CLE SEA | 5.19 | BLI SEA | 5.37 | YKM SEA | 5.40 | SNA SEA | 5.48 | LIH	airport | dep_delay | airport_to ---+---+--- BOS | -5.0 | SWF BOS | -3.0 | ONT BOS | 1.0 | GGG BOS | 1.20 | AUS BOS | 3.05 | LGA BOS | 3.24 | MSY BOS | 5.13 | LGB BOS | 5.78 | OAK BOS | 5.89 | MDW BOS | 5.98 | BDL
airport | dep_delay | airport_to ---+---+--- CMH | -5.0 | SYR CMH | -5.0 | AUS CMH | -5.0 | OMA CMH | 1.0 | MSN CMH | 1.0 | CLE CMH | 1.35 | SDF CMH | 3.70 | CAK CMH | 3.93 | SLC CMH | 4.15 | MEM CMH | 4.15 | IAD	airport | dep_delay | airport_to ---+---+--- SRQ | 0.0 | EYW SRQ | 1.32 | TPA SRQ | 1.44 | IAH SRQ | 1.70 | MEM SRQ | 2.0 | FLL SRQ | 2.06 | BNA SRQ | 2.36 | MCO SRQ | 2.53 | RDU SRQ | 2.83| MDW SRQ | 3.35 | CLT	

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

Similar to Question 2.1 and 2.2, 3 Spark applications are used, together with Cassandra for sql quries.

The given query result is:

airport | airport_to | arr_delay ---+---+--- LGA | BOS | 1.48	airport | airport_to | arr_delay ---+---+--- BOS | LGA | 3.78	airport | airport_to | arr_delay ---+---+--- OKC | DFW | 5.07
airport | airport_to | arr_delay ---+---+--- MSP | ATL | 6.73		

Question 3.2 Tom wants to travel from airport X to airport Z. However, Tom also wants to stop at airport Y for some sightseeing on the way...

This question could be spitted into 2 sub-questions:

1. Find the flight from X to Y on given date in the morning, with minimum arrival delay;

*2. Find the flight from Y to Z on 2 days after given date in the afternoon, with minimum arrival d*elay.

A table is created in Cassandra cqlsh:

```
create table aviation.optimum_flights (
```

```
 airport_from text,
 airport_to text,
 given_date date,
 am_or_pm text,
 carrier text,
 flight_num text,
 departure_time text,
 arr_delay decimal,
 PRIMARY KEY(airport_from, airport_to, given_date, am_or_pm)
);
```

PySpark is then executed together with Spark-Cassandra_Connector, and write the data into Cassandra table.
To query the result in Cassandra cqlsh:

```
select * from aviation.optimum_flights where airport_from = ‘CMI' and airport_to = 'ORD' and given_date = '2008-04-03' and am_or_pm = 'AM';
select * from aviation.optimum_flights where airport_from = ‘ORD' and airport_to = 'LAX' and given_date = '2008-04-05' and am_or_pm = 'PM';
```

The given query result is:

	Carrier	Flight number	Departure time	Flight date	Arrival delay
BOA → ATL → LAX, 04/03/2008	FL	270	06:00	2008-03-04	7.0
	FL	40	18:52	2008-05-04	-2.0
PHX → JFK → MSP, 09/07/2008	B6	178	11:30	2008-07-09	-25.0
	NW	609	17:50	2008-09-09	-7.0
DFW → STL → ORD, 01/24/2008	AA	1336	07:05	2008-24-01	-14.0
	AA	2245	16:55	2008-26-01	-5.0
LAX → MIA → LAX, 05/16/2008	AA	280	08:20	2008-16-05	10.0
	AA	456	19:30	2008-18-05	-19.0