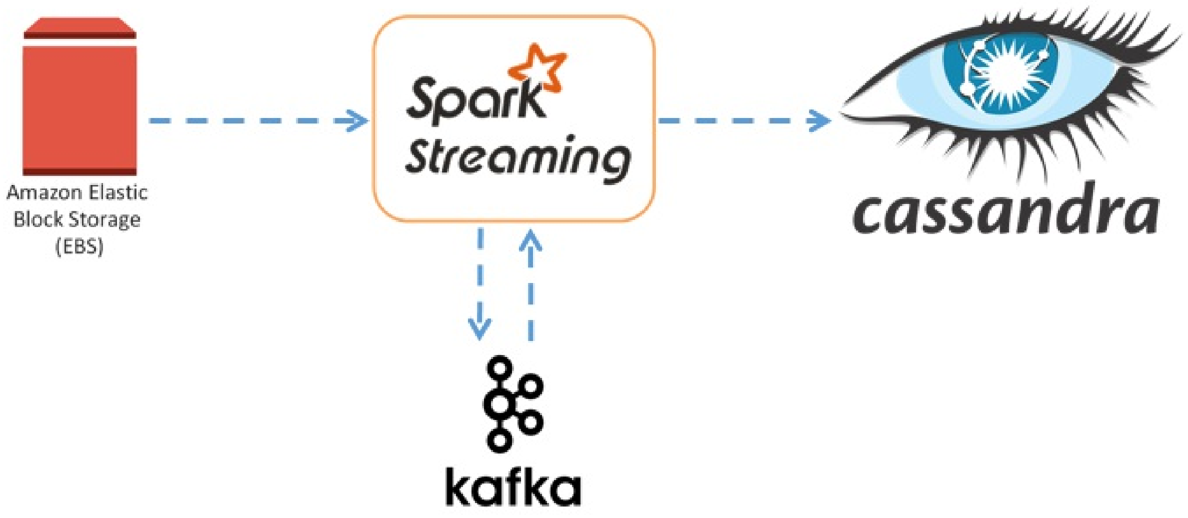
System Integration

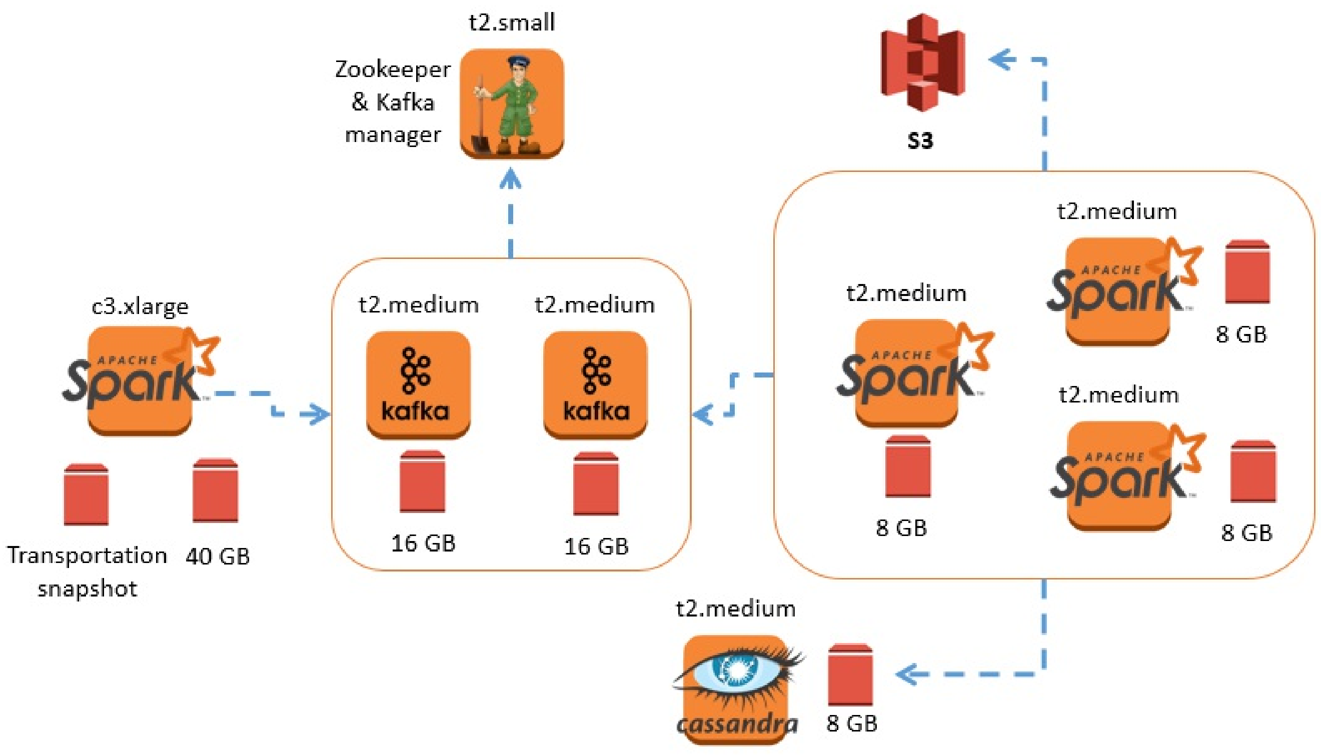
Components and Data Flow

Input is being read from EBS volume using Spark Streaming. The results are trimmed and directed to Kafka cluster. Separate Spark Steaming jobs are reusing this stream from Kafka, continuously refining the data flow before saving it to Cassandra.



Deployment View

Transportation dataset is mounted as EBS volume under a c3.xlarge instance. This also has a pretty big 40 GB local EBS volume for the extracted CSV files. These are ingested by local Spark Streaming job running in 4 threads parallel. Kafka cluster has 2 t2.medium nodes and coordinated by Zookeeper and kafka-manager. These are installed on a separate t2.small node. Spark Streaming cluster contains 1 director and 2 worker nodes. They're using an S3 bucket for saving checkpoints. For simplicity Cassandra is only installed to one t2.medium.



Cassandra Migration

Migrating to Apache Cassandra is done by using Spark Cassandra Connector. This allows shifting loaded DataFrames from Spark to Apache Cassandra.

References

- Spark Cassandra Connector
- kafka-manager

Solution Approach

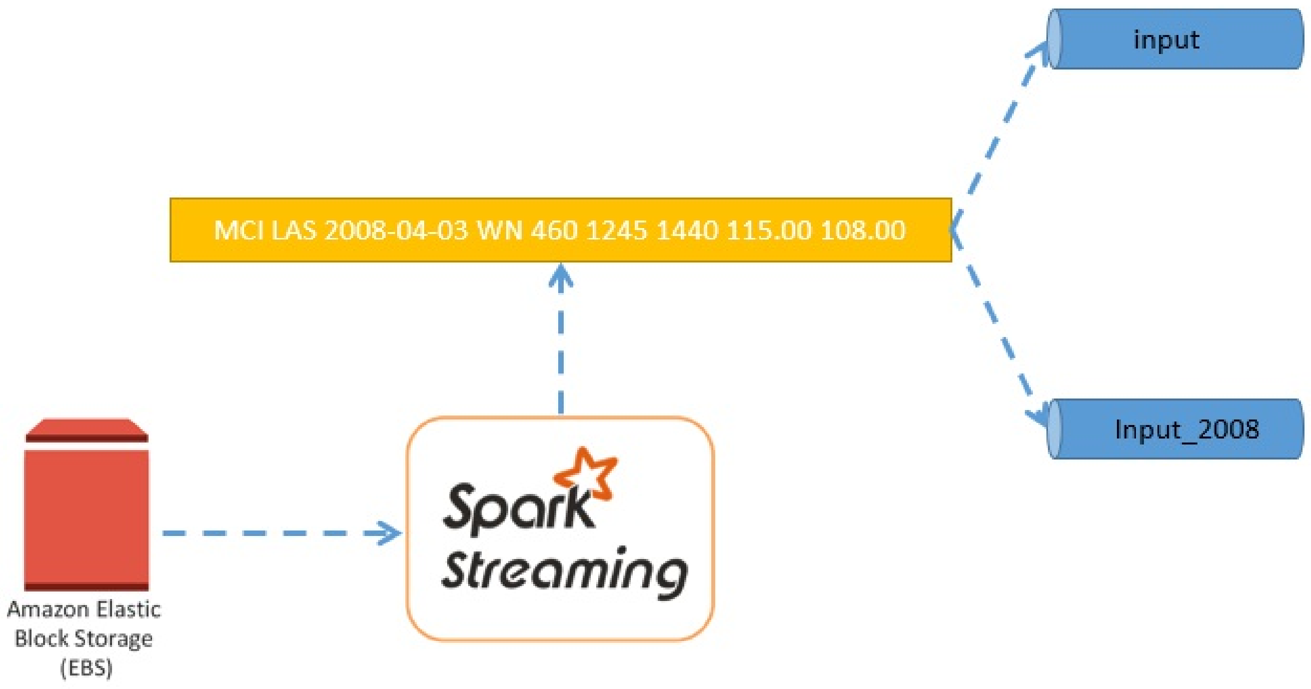
Feeding data into Kafka

First we start a Spark job locally, that watches a directory for incoming files. That directory is populated with CSV files from the `airline_ontime` folder of transportation dataset. The CSV extraction is done by a `bash` script.

The Spark Streaming job will cut off all unnecessary columns from the on-time performance CSV files. The structure for one message is the following:

AIRPORT_FROM | AIRPORT_TO | DEPARTURE_DATE| CARRIER_ID | FLIGHT_NUM | SCHEDULED_DEPARTURE_TIME | DEPARTURE_TIME | DEPARTURE_DELAY | ARRIVAL_DELAY |

We're populating two input queues. One is just feeded with data from 2008.


input
MCI LAS 2008-04-03 WN 460 1245 1440 115.00 108.00
Input_2008
Spark Streaming
Amazon Elastic Block Storage (EBS)

Starting ingestion job locally on 4 threads.

```
~/spark-2.1.0-bin-hadoop2.7/bin/spark-submit --master local[4] --conf spark.streaming.backpressure.enabled=true --conf
spark.streaming.receiver.maxRate=4000 ./ingest_files_to_kafka.py input
```

Populating the input folder

```
./move-ontime-perf-to-localfs.sh data/aviation input
```

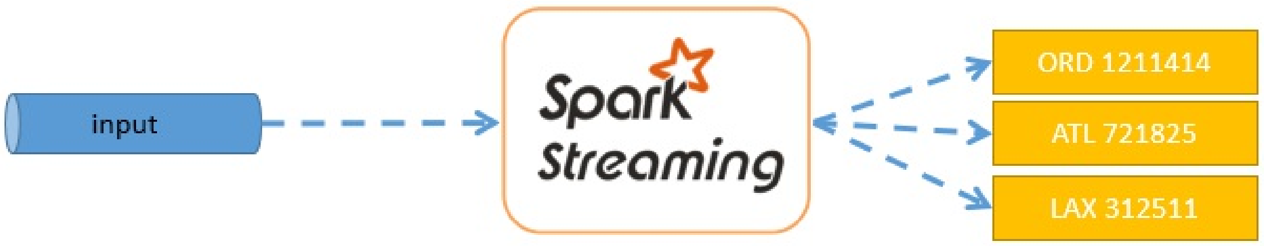
Sample data in Kafka

```
MCI LAS 2008-04-03 WN 460 1245 1440 115.00 108.00
MCI LAS 2008-04-03 WN 1758 0900 0854 -6.00 -6.00
MCI LAS 2008-04-03 WN 2888 0705 0703 -2.00 -7.00
MCI LAX 2008-04-03 WN 238 1440 1553 73.00 57.00
MCI LAX 2008-04-03 WN 450 1135 1226 51.00 41.00
```

References

- Migration script on GitHub
- Spark Streaming job on GitHub

Question 1.1



Airport from-to information is collected by using flatMap from input stream

```
rows.flatMap(lambda row: [row[0], row[1]])
```

We use the `updateStateByKey` function with Spark checkpoints to count all the occurrences for all airports. The updateFunction is a simple counter function.

```
airports.map(lambda airport: (airport, 1)).updateStateByKey(updateFunction)
```

To reduce the traffic, we cut off the amount of records to just the top 10 most popular in each partition.

```
sorted.transform(lambda rdd: rdd.mapPartitions(cutOffTopTen))
```

Then we sort RDDs by popularity.

References

- Spark Streaming job on GitHub

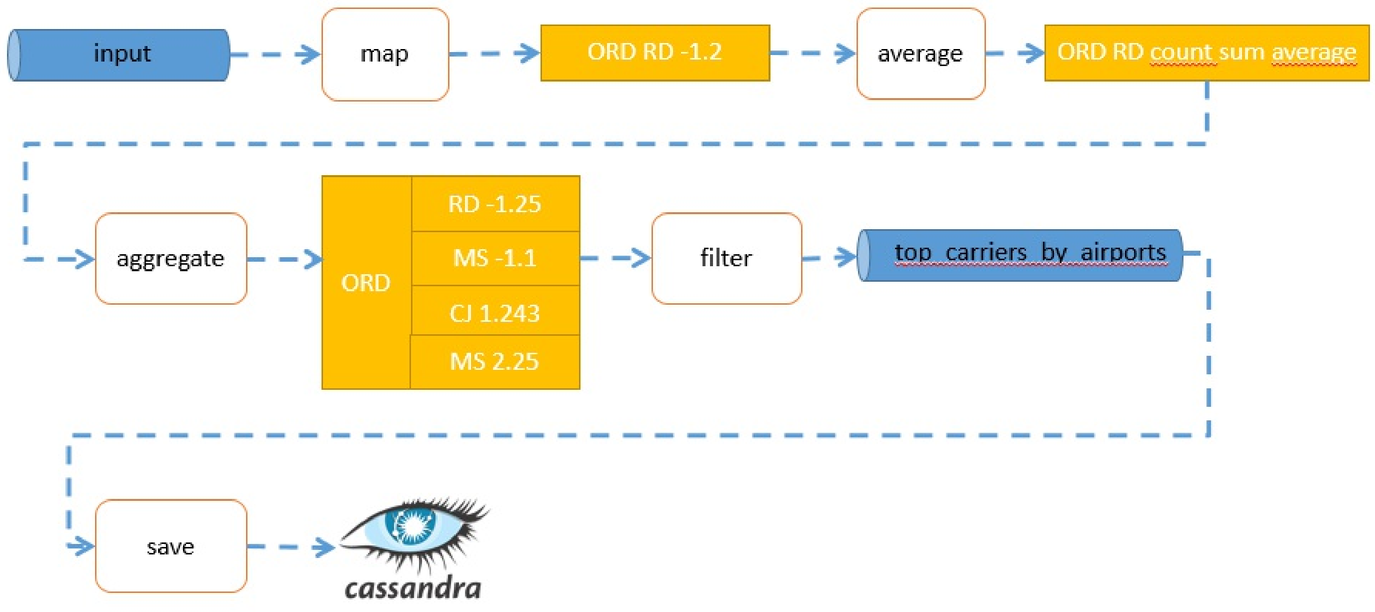
Question 1.2

This is analogue to Question 1.1

References

- Spark Streaming job on GitHub

Question 2.1



First we use the `updateStateByKey` function with Spark checkpoints to count average departure delays for all airport-carrier pairs. The `updateFunction` calculates three values for each: sum, count and sum/count.

```
airports_and_carriers.updateStateByKey(updateFunction)
```

```
def updateFunction(newValues, runningAvg):
    if runningAvg is None:
        runningAvg = (0.0, 0, 0.0)
    # calculate sum, count and average.
    prod = sum(newValues, runningAvg[0])
    count = runningAvg[1] + len(newValues)
    avg = prod / float(count)
    return (prod, count, avg)
```

Then we use the `aggregateByKey` to have an ordered list of top ten performing carrier for each airport. This is tricky at first, but keeps calculations and data traffic at minimum. Aggregate contains top ten carriers and departure delays. Sample aggregated value for an airport: `[('TZ',-0.0001), ('AQ',0.025), ('MS',0.3)]`

```
airports = airports.transform(lambda rdd: rdd.aggregateByKey([],append,combine))
```

When this is done, all continuously refined top ten performing carriers are delivered to a separate topic called `top_carriers_by_airports` This topic is then consumed by another Spark Streaming job, which saves and updates values to Cassandra.

References

- Spark Streaming job on GitHub
- Cassandra migration job on GitHub
- Cassandra table definitions

Question 2.2

This is analogue to Question 2.1

References

- Spark Streaming job on GitHub
- Cassandra migration job on GitHub
- Cassandra table definitions