Coursera Course: Cloud Computing

Capstone Project Task 2 Report

Hang Shi

November 23, 2017

1 Overview

This report covers the design and implementation details for Cloud Computing Captone Project Task 2. The goals of this task are to perform the following:

- Extract and clean the transportation dataset, and then store the result in HDFS.
- Answer 2 questions from Group 1, 3 questions from Group 2, and Question 3.2 using Spark. Store the results for questions from Group 2 and Question 3.2 in Cassandra.

2 Data Cleanup

The first task is to clean and store the dataset. We first retrieve and extract the dataset from the EBS volume snapshot by following steps in `http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-restoring-volume.html`. One key thing to note is that we need to create EC2 instance in the same region and availabbility zone as the data EBS snapshot (snap-e1608d88 in us-east-1d).

Afterwards, we explore the data set and decide on how we wish to clean it. Then we ignore unrelated rows and extract only the columns from the csv files. The raw data is in `/data/aviation/airline_ontime` directory.

Python and Bash scripts are developed used to do the cleanup:

- A python script is developed on top of python CSV package to extract the fields.

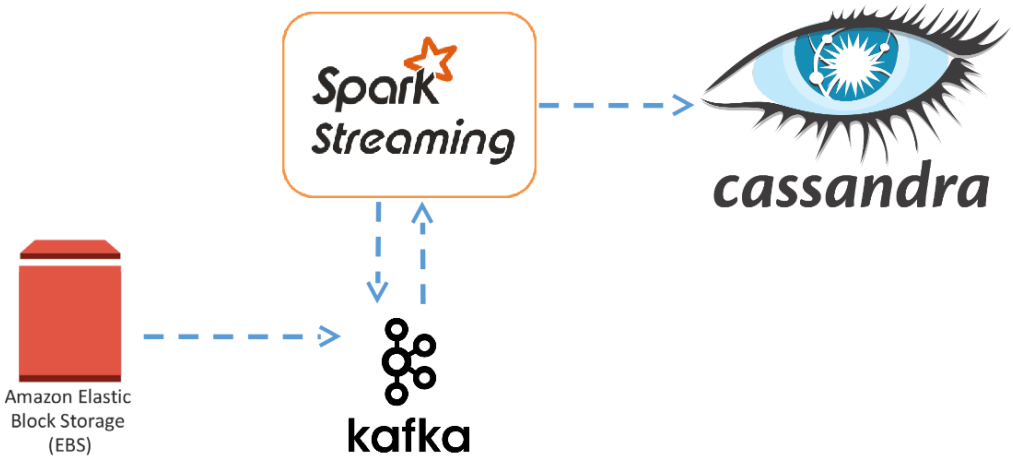

Figure 1: System Architecture for This Project.

- Bash scripts are used to loop through all data sets.

Such data cleanup is a one-time job. The cleaned data are stored on HDFS via below hdfs dfs command as tab delimetered fields with each line indicating one flight. For question 3.2, only 2008 data is used.

```
Cmd: hdfs dfs -copyFromLocal <local files> <hdfs dir>
Fields: Year, Month, Day, Weekday, FullDate, Airline, Carrier
    , Flight, Origin, Dest, CRSDepTime, DepDelay, ArrDelay
```

3 System Overview

See figure 1 for system architecture details for this project.

3.1 Hadoop, Kafka, and Spark Streaming

The system is essentially a 4 node Apache Hadoop cluster with one namenode and three datanodes running on AWS EC2. Hadoop Yarn runs and serves as the resource manager and application manager. Kafka is used to stream the cleaned data stored in HDFS to Spark, which does the RDD/Data Frame processing before storing the results to NoSQL database. See below link for details on how to set up and run Spark on EC2. Running Spark on EC2

3.2 NOSQL Database: Cassandra

Cassandra cluster is deployed on the three Hadoop datanodes to provide data replication and scalability. The cluster run on the same rack and utilizes GossipingPropertyFileSnitch

scheme. One of the nodes is assigned as the seed provider. Initially version 3.6 is used, but due to the support of group by is missing in 3.6, version 3.11 is used.

Cassandra Async Batch operation is used to insert/update data, and query scripts are written via Python Cassandra API as well, Cassandra Python Driver.

4 Group 1 Problems

- Problem 1.1: Rank the top 10 most popular airports by numbers of flights to/from the airport.
- Problem 1.2: Rank the top 10 airlines by on-time arrival performance

Solution: both problems are solved using same technique: stateful updates via Spark API updateStateByKey. See below for details.

```
def updateFunction(newValues, runningCount):
    return sum(newValues, runningCount or 0)
filtered = lines.map(lambda line: line.split("\t"))\
                .flatMap(lambda w: [(w[3], 1), (w[4], 1)] )\
                .reduceByKey(lambda a, b: a+b)\
                .updateStateByKey(updateFunction)\
                .transform(lambda r: r.sortBy(lambda (w, c): -c))
```

5 Group 2 Problems

- Problem 2.1: For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.
- Problem 2.2: For each airport X, rank the top-10 airports in decreasing order of on-time departure performance from X.
- Problem 2.3: For each source-destination pair X-Y, rank the top-10 carriers in decreasing order of on-time arrival performance at Y from X.
- Problem 2.4: For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.

Solution: All four questions are solved using same approach: Spark stateful RDD transformation, Data Frame API, and Cassandra Spark API. See below two code examples from problem 2.2. Other questions are similar.

```
# RDD Processing
def updateFunction(newValues, runningCount):
    values, counter, avg_delay = runningCount or (0., 0, 0.)
    for val in newValues:
        values += val[0]
        counter += val[1]
```

```
    return (values, counter, values/counter)
f1 = lines.map(lambda line:line.split(","))\
          .map(lambda f:Flight(f))\
          .map(lambda f:((f.Origin, f.Dest),(f.DepDelay, 1)))\
          .updateStateByKey(updateFunction)
filtered = f1.map(lambda (x, y):(x[0], y[2], x[1]))
filtered.foreachRDD(lambda rdd:print_rdd(rdd))

# Data Frame Transformation and Data Storing
schema = StructType([
          StructField("origin", StringType(), True),
          StructField("delay", FloatType(), True),
          StructField("dest", StringType(), True)
          ])
df = getSqlContextInstance(rdd.context).createDataFrame(rdd, schema);
df.show()
#insert into cassandra
df.write.format("org.apache.spark.sql.cassandra")\
    .mode('overwrite').options(table="g2e2", keyspace="test")\
    .save()
```

Data Model or Cassandra Table Definitions:

```
CREATE TABLE test.g2e1 (origin text, delay float, airline text,
    carrier text, PRIMARY KEY (origin, airline) );
CREATE TABLE test.g2e2 (origin text, delay float, dest text, PRIMARY
    KEY (origin, dest) );
CREATE TABLE test.g2e3 (origin text, dest text, delay float, airline
    text, PRIMARY KEY ((origin, dest), airline));
CREATE TABLE test.g2e4 (origin text, dest text, delay float, PRIMARY
    KEY (origin, dest));
```

6 Group 3, Problem 3.2

- Problem 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. More concretely, Tom has the following requirements (see Task 1 Queries for specific queries):
  - A. The second leg of the journey (flight Y-Z) must depart two days after the first leg (flight X-Y). For example, if X-Y departs January 5, 2008, Y-Z must depart January 7, 2008.
  - B. Tom wants his flights scheduled to depart airport X before 12:00 PM local time and to depart airport Y after 12:00 PM local time.
  - C. Tom wants to arrive at each destination with as little delay as possible.

Solution: First the dataset is mapped into two dataset f_xy and f_yz based on whether the departure time is earlier than 12:00. During f_yz processing, the departure date is substracted two days. Then the two dataset are joined together with origin of f_xy and

destination of f_yz are the same, and dates are two days apart. The results are stored into Cassandra database using same approach as in group 2 problems. The primary and cluster keys are days combined with airport and delay, like "PHX_JFK_MSP_2008-09-07".

```
def map_xy(f):
    fdate = datetime.date(f.Year, f.Month, f.Day)
    return ((str(fdate), f.Dest), f)
def map_yz(f):
    fdate = datetime.date(f.Year, f.Month, f.Day)
    fdate -= datetime.timedelta(days=2)
    return ((str(fdate), f.Origin), f)
def process_record(record):
    f_xy = record[1][0]
    f_yz = record[1][1]
    route = '_'.join((f_xy.Origin, f_xy.Dest, f_yz.Dest, record
        [0][0]))
    delay =  f_xy.ArrDelay + f_yz.ArrDelay
    details = "%s;%s" % (f_xy, f_yz)
    return (route, delay, f_xy.FlightNum, f_yz.FlightNum, details)
# RDD Processing
f_xy = ff.map(map_xy).filter(lambda (k,v): v.CRSDepTime < '1200')
f_yz = ff.map(map_yz).filter(lambda (k,v): v.CRSDepTime > '1200')

f_xyz = f_xy.join(f_yz).map(process_record)
f_xyz.foreachRDD(lambda rdd: print_rdd(rdd))
```

Data Model or Cassandra Table Definitions:

```
CREATE TABLE test.g3e2 (route text, delay float, details text, f1
    text, f2 text, PRIMARY KEY (route, f1, f2));
```

7 Optimization and Discussion

- Removal of unused data columns: only needed columns have been imported from the original dataset, thus lowering the needs for data transfer and storage capacity.
- Data ordering and partitioning: the import script orders and partitions by date the data from the original dataset. This way the import process into HDFS is faster.
- Data preload into HDFS: data is loaded to HDFS before starting to process it. This way, we can achieve persistent storage in HDFS and fast access from HDFS.
- During Kafka streaming, the Spark executor can't handle full speed of Kafka streaming, two approaches are used:
  - Increase the memory when inovking spark streaming for executor and driver;
  - Sleep a small interval between streaming two files.
- For Problem 3.2, the dataset of join results are huge, it is better to do filtering of data for queries.

- AWS DynamoDB are also used in some of the problem. My personal opinion is that it is not as user friendly as Cassandra.
- For cleaned data, it is easier to use a Python data structure (class) to store that during runtime, see the codes in flight.py for details.
- Python Cassandra Batch operation is used to insert and update data to reduce the connection set up and teardown overhead.
- The results from the design of this project do make sense.For question 3, 2, batch operation may work better; for other problems, streaming processing is faster and provides many useful operations such as join, filter, reduce, stateful update, and map transformation.

8 Code and Video Location

- Code: `https://github.com/myhangshi/ccompute`
- Video: `https://vimeo.com/243786081`