Coursera Course: Cloud Computing

Capstone Project Task 1 Report

Hang Shi
November 3, 2017

1 Overview

This report covers the design and implementation details for Cloud Computing Captone Project Task 1. 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 both questions from Group 3 using Hadoop. 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.

Approaches: Two ways have been used to do the cleanup:

- A python script is developed on top of python CSV package to achieve this.
- Pig script, see `https://pig.apache.org/` for details.

Such data cleanup is a one-time job and below are the fields we extracted:

- AirlineID:chararray, –a8
- Carrier:chararray, –a9
- FlightNum:int, –a11
- Origin:chararray, –a12
- Dest:chararray, –a18
- DepDelay:float, –a26
- DepDel15:float, –a28
- ArrDelay:float, –a37
- ArrDel15:float –a39

The cleaned data are stored on HDFS via below hdfs dfs command as tab delimetered fields with each line indicating one flight.

```
hdfs dfs -copyFromLocal <local files> <hdfs dir>
```

3 System Overview

3.1 Hadoop MapReduce

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. Below link provides some useful information on the setup:
Spinning Up a Free Hadoop Cluster: Step by Step

3.2 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.

4 Group 1 Problems

- Problem 1.1: Rank the top 10 most popular airports by numbers of flights to/from the airport. Solution: We extracted the origin and destination city of the flight for each line of the cleaned files. It is done two steps: counting via map and reduce phases in Hadoop and sort to get top 10 airports. In the map phase, we generate pairs (origin, 1) and (dest, 1). In the reduce phase, if we see a new airport, we output the airport and count; if we see same airport, we add the count to existing

counter. Once the counters are outputted, below commands can be used to get top airports (use 20 to allow duplicated top cases).

```
hadoop fs -cat /output/of/wordcount/part* | sort -n -k2 -r | head
    -n20
```

Alternatively, the top N problem can be solved via this MapReduce design pattern.

- Problem 1.2: Rank the top 10 airlines by on-time arrival performance In the map phase, we generate (airline, 1, delay). In the reduce phase, we use two maps: (airline, count), and (airline, total delay) and accumulate counter and delay into these two maps. Then we calculate arrival performance and sort them. Alternatively and if there are multiple reducers, we will use commands similar as in above problem to do sorting.

- Problem 1.3: Rank the days of the week by on-time arrival performance This is a similar problem as Problem 1.2. We use (weekday, delay) as input data. In the map phase, we generate (weekday, 1, delay). In the reduce phase, we use two maps: (weekday, count), and (weekday, total delay) and accumulate counter and delay into these two maps. Then we calculate arrival performance and sort them.

5 Group 2 Problems

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

  We use Pig Latin script to generate the results into HDFS files and then store them into Cassandra database for queries. Below is the script:

```
in  = LOAD '$PIG_IN_DIR' AS
          ( AirlineID:chararray, --a8
            Carrier:chararray,   --a9
            FlightNum:int,       --a11
            Origin:chararray,    --a12
            Dest:chararray,      --a18
            DepDelay:float,      --a26
            DepDel15:float,      --a28
            ArrDelay:float,      --a37
            ArrDel15:float       --a39
          );

group_by_origin_airline = GROUP in BY (Origin, Carrier, AirlineID);

average_ontime = FOREACH group_by_origin_airline
                 GENERATE FLATTEN(group) AS (Origin, Carrier, AirlineID),
                          AVG(in.DepDelay) AS performance_index;

group_by_origin = GROUP average_ontime BY Origin;
```

```
top_ten_airlines = FOREACH group_by_origin {
   sorted_airlines = ORDER average_ontime BY performance_index ASC;
   top_airlines = LIMIT sorted_airlines 10;
   GENERATE FLATTEN(top_airlines);
}

X = FOREACH top_ten_airlines GENERATE TOTUPLE( \
TOTUPLE( 'origin',$0), TOTUPLE( 'carrier',$1),\
TOTUPLE('airline', $2 )), TOTUPLE($3);

STORE X into '$PIG_OUT_DIR';
```

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

  Below is the PIG script (table loading part is ignored).

```
group_by_origin_dest = GROUP in BY (Origin, Dest);

average_ontime = FOREACH group_by_origin_dest
                 GENERATE FLATTEN(group) AS (Origin, Dest),
                          AVG(in.DepDelay) AS performance_index;

group_by_origin = GROUP average_ontime BY Origin;
top_ten_airports = FOREACH group_by_origin {
   sorted_airports = ORDER average_ontime BY performance_index ASC;
   top_airports = LIMIT sorted_airports 10;
   GENERATE FLATTEN(top_airports);
}

X = FOREACH top_ten_airports GENERATE TOTUPLE( \
TOTUPLE( 'origin',$0), TOTUPLE('dest', $1 )), \
TOTUPLE($2);

STORE X into '$PIG_OUT_DIR';
```

- For each source-destination pair X-Y, rank the top-10 carriers in decreasing order of on-time arrival performance at Y from X.

  Below is the PIG script (table loading part is ignored).

```
group_by_origin_dest_airline = GROUP in BY (Origin, Dest, AirlineID);

average_ontime = FOREACH group_by_origin_dest_airline
                 GENERATE FLATTEN(group) AS (Origin, Dest, AirlineID),
                 AVG(in.ArrDelay) AS performance_index;

group_by_origin_dest = GROUP average_ontime BY (Origin, Dest);
```

```
top_ten_airlines = FOREACH group_by_origin_dest {
   sorted_airlines = ORDER average_ontime BY performance_index ASC;
   top_airlines = LIMIT sorted_airlines 10;
   GENERATE FLATTEN(top_airlines);
}

X = FOREACH top_ten_airlines GENERATE TOTUPLE( \
TOTUPLE('origin',$0), TOTUPLE('dest', $1),TOTUPLE(\
'airline', $2 )), TOTUPLE($3);

STORE X into '$PIG_OUT_DIR';
```

6 Group 3 Problems

- 1. Does the popularity distribution of airports follow a Zipf distribution? If not, what distribution does it follow?

  Zipfs law stats that given some corpus or natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. In our case it means that the airport with a higher rank should have a double number of flights compare with the next airport in rank. Even if from the Flights by Airport figure we can hope for a Zipf distribution, after doing a log-log plot we can see that Airports rank by number of flights is not following this distribution (log log for Zipf should be a straight line and is not). gnuplot tool is used to do plotting and it can be proven that this distribution is not Zipf but more a Lognormal one since the bottom half look very different than the top half.

- 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. It is solved by creating two MapReduce Jobs. One which filter the flights by arrival performance and the second one which computes the options Tom has and pick up the best one based on arrival performance. For the first map reduce the key is based on Origin, Destination and FlightDate. In the second MapReduce we will receive a list of best arrival flights grouped by Origin-Destination-Date and from here using the 12h departure condition we can generate for each day all the options Tom has.

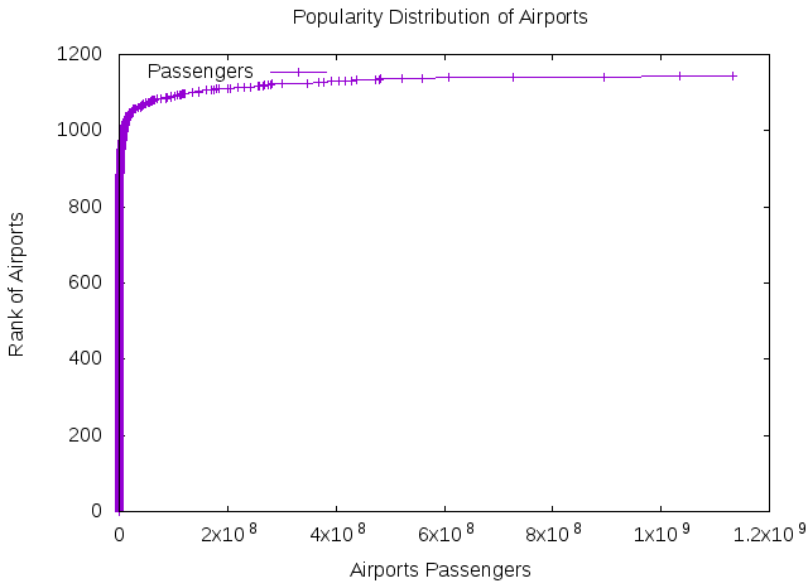

Figure 1: For poblem 3.1 Popularity Distribution of Airports.

7 Optimization Techniques

- 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.
- One optimization is to use the combiner in the map phase in order to minimize the network traffic. This is used in problems in group 1. For example, in question 1, use same reducer class as combiner class.

8 Code and Code Location

- `https://github.com/myhangshi/ccompute`
- `https://vimeo.com/241272936`