A Dynamic *Carpooling* Application

Shardendu Gautam, Arushi Shah,
Srinidhi Raghavan, Vidya Venkiteswaran
(sg3391, avs2155, ss5145, vv2269)

COMSE 6998: Big Data and Cloud Computing
Department of Computer Science, Columbia University

Air pollution is the one of the most important environmental issues. One simple yet effective way to control this is Carpooling. Carpooling allows people to share their vehicle for economic and environmental benefits. It reduces the emission and fuel costs, decreases the need for vehicular maintenance, minimises traffic jams and gives monetary benefits to the driver. The Dynamic Car application provides an option for any driver to pick up passengers on his way such that the maximum travel time of every passenger is optimised. The application as opposed to many current technologies provides the optimal automated solution to both passengers and drivers.

Introduction

As opposed to travelling alone, Carpooling offers several benefits. These benefits can be environmental , financial, social. It also has other pros like reducing the traffic on the roads. In this application, a user requests for a carpool in advance. Based on his request and the maximal optimal solution, the application provides the user with a perfect match. While finding these matches, a couple of assumptions are being made. The users must request for a journey at-least an hour in prior of the actual journey time. The total number of passengers picked by the driver in the entire journey is at-most equal to the capacity of the vehicle. The maximum acceptable delay in time is 15 minutes and the deviation in distance is 2 miles. The entire application is biased towards the driver and hence gives the best solution with respect to him.

Problem Statement

In this problem, a user is of two types: Driver and Rider. Every user sends a request for a carpool. The request is represented as follows:

$$(q_T, q_O, q_D, q_C)$$

where q_T is the type of the user, q_O is the origin of the ride, q_D is the destination and q_C is the capacity of the vehicle.

If the user is a driver, then q_c denotes the maximum capacity of the vehicle; otherwise, it is set to 0. This is because the application just serves one passenger and does entertain a group.

Each request is further accompanied by additional details. These details include the drivers start and estimated reach time, riders start-time, etc. All these inputs must be fed at-least an hour before the actual start time of the journey. As soon as a request for journey arrives, it is fed into the Database. Every hourly, a sub-list of drivers with their journey time with-in the next two hours are drawn from the database and enqueued into a SQS Queue. These drivers are then processed one by one asynchronously as per their start time.

A driver offers a free ride to a rider. The number of riders escorted by a driver is constrained by the maximum capacity of the drivers vehicle. The application is completely automated and suggests the drivers a list of possible riders just an hour before his journey time. Similarly, it suggests the corresponding driver to the rider. The objective function here, is to minimize the delay in time and deviation in distance of the driver and simultaneously maximize the total travel time of a passenger constrained by the capacity of the vehicle. Mathematically, this can be expressed as follows:

$$min \delta_d + \delta_t$$

$$max\ T$$

$$max\ P$$

such that

$$\delta_d \leq 2miles$$

$$\delta_t \leq 15minutes$$

$$T \leq C.t_d$$

$$|P| \leq C$$

$$\delta_d, \delta_t, T, P \geq 0$$

where δ_d, δ_t is the total deviation in distance and the delay in time respectively due to incorporating the set of passengers. The maximum acceptable delay in time is 15 minutes and that of distance is 2 miles. Both the quantities can be dynamically set by the driver. T is the total travel time of all the passengers put together. P is the number of passengers. Note that the maximum number of passengers throughout the journey is C. C is the capacity of the vehicle and t_d is the travel time of the driver. In other words, the application does not maximize the number of passenger pick-ups rather it tries to limit it to the capacity and maximize the total journey time of a rider.

Algorithm

The optimization problem mentioned above is a discrete combinatorial optimization problem with a non-polynomial run-time. Hence, we must simplify the constraints and the objective function to execute efficiently in polynomial time. To do this, we compute a feasible region and a feasible set of passengers and optimize the maximum travel time of the passengers. Besides, the route suggested to the driver is always the shortest path between the source and the destination. The other constraints are always applied on this shortest path. The basic flow of the algorithm is given as follows:

1. Pull a driver from the queue

2. Get source and destination of the driver. Here,

   sd: Source of the Driver,
   dd: Destination of the driver, and
   c: Capacity of Driver's Vehicle.

3. Compute the shortest path between sd and dd using Google Maps API

   $$D = Haversine(sd, dd)$$

4. Break the path into small steps using the Maps API

5. Find the maximum and minimum latitude and longitude points from these steps and this is used to compute the boundaries of the feasible region: s_{low}, s_{high}, d_{low}, d_{high}

6. There are three separate cases for finding these regions based on the curvature of the paths. The paths can either be a vertical line or a horizontally aligned path

   CASE 1: Vertical Path

   $$s_{low} = (min\ P_{lat}, s_{long})$$
   $$dl_{ow} = (max\ P_{lat}, s_{long})$$
   $$s_{high} = (\min P_{lat}, d_{long})$$
   $$d_{high} = (\max P_{lat}, d_{long})$$

   CASE 2: Horizontal Path

   $$s_{low} = (s_{lat}, \min P_{long})$$
   $$s_{high} = (slat, \max P_{long})$$

$$d_{low} = (d_{lat}, \min P_{long})$$
$$d_{high} = (d_{lat}, \max P_{long})$$

where

$s_{low}, s_{high}, d_{low}, d_{high}$:Boundaries of the feasible region

d_{lat}, d_{long} : Coordinates of driver's destination

s_{lat}, s_{long} : Coordinates of driver's source

P_{lat}, P_{long} : Coordinates of the in-between way-points

7. Remove the passengers within the rectangular region (s_{low}, s_{high}, d_{low}, d_{high}) within the start and end time of the drivers journey

8. Retain only those passengers whose travel distance is at-least αD. For this case, $\alpha = 0.1$

9. For each of these remaining passengers, find the nearest step-point such that the difference between the estimated arrival time of the driver and the start time of the passenger is as minimal as possible (tentatively 5 minutes). Note that the distance between a passenger and his nearest step-point should at-least be 0.5.

10. For the remaining list of passengers, choose C passengers with the maximum distance

11. Return these C passengers to the driver

After processing, the passenger is notified with the details of the driver and the driver gets the list of passengers with their source and destination. Note that neither the driver nor the driver gets the flexibility to reject. If in the worst case, a rider rejects a driver then, the rider wont any ride for the chosen path. And if rejected by a passenger, the driver never gets a side option.

AWS Components

The AWS Components used for this set-up are: Dynamo-DB, SNS, SQS and Elastic Beanstalk.

- DynamoDB:It is a fully managed cloud database which supports both document and key-value store models. Its flexibility and easy integration with MongoDB, makes us choose it for storing the data

- SNS:The various notifications to the driver and user are sent using the SNS. Its scalability makes it possible to send multiple messages to multiple recipients simultaneously

- SQS: The Amazon SQS system is used for storing the drivers and riders in real-time. It provides a cost-effective mechanism to couple and decouple elements in cloud. Dynamic Car uses two different queues- one for the driver and the other one for the rider. The queues are created at every fixed intervals in-order to recommend the best maximal solution.

- Elastic Beanstalk: This component is used to manage and deploy application in a large-scale.

Implementation

The Github link of the project is as given: https://github.com/vidyavnv/DynamiCar/tree/Dynamo

Code Structure

- Technology Stack

  - FrontEnd: HTML, CSS, JS
  - BackEnd: Python (Flask)

- *application.py:* It consists of views for the UI. Each url is directed to its view in application.py. It helps in storing user and trip details received from UI to DynamoDB. As the user signs up, a SNS notification is sent to the customer.

- *utils.py*: Common functions which are used throughout the application. Specifically, it contains getPath which calls the Google Map API to look for waypoints between source and destination. processPath is used to structure the waypoints by adding the time from the source to a waypoint

- *algorithm/*:

  - *algorithm.py*: This is the main algorithm which returns a list of riders matched with a driver

  - *sqs_producer.py*: Every hour, data is pulled from DynamoDB to SQS queue (driver queue and rider queue) associated with drivers and riders scheduled for 1 hour from the algorithm run till estimated arrival of driver at his/her destination. Example: If driver D is scheduled to depart from source at time 9PM, the algorithm will run at 8PM and pick all the drivers which were scheduled for departure from their source between 9 to 10PM. All the drivers who are scheduled between time 9PM and till the maximum arrival time are fetched from the database and inserted into SQS

  - *sqs_consumer.py*: From driver queue, data corresponding to one driver is pulled and sent to the algorithm along with all the riders in the rider queue.

Conclusion

Intelligent Carpool Systems are a very important set of optimization problems. The main issue with such ICS systems is that they have to implement discrete combinatorial problems which are NPC. In order to get rid of these disadvantages, the algorithm processes the requests to find the optimal solution at every fixed intervals. However, this algorithm can be improved by making it real-time and including features like rating, preferences, etc. It can also include financial benefits and optimize the maximal solution.