Large Scale Data Engineering Assignment 2

Thomas Koch
Hassan Jalil

March 29, 2016

Abstract

In this paper we describe how we rendered a map of the number of taxi's on different street-segments within New York City. For this purposed we determined the exact path taken by all taxi trips in 2013 using a path-finding algorithm on a cluster using Spark and GraphHopper. The end-result is rendered on the client with WebGL by MapBoxGL.

1 Introduction

Through the Freedom of Information Law (FOIL) a data-set about all the taxi-trips within New York City in the year 2013 was obtained from the New York Taxi and Limousine Commission. The data contains a lot of information about the artery of the metropolis: how it's people are moving. What the data-set did not contain was the exact path taken by the taxi took between the pick-up point and drop-off point, which made it impossible to do analysis on which roads were taken by each taxi to get to it's destination. For this purpose it is necessary to execute a route using a path-finding algorithm for each trip, requiring about 220 million queries to a street-router.
In related work we have seen this being completed for a single day but so far not for the entire data-set as it is very high workload, requiring a lot of resources and/or time.
In this paper we describe the effort to complete this task on the SARA Hadoop cluster using Apache Spark and the routing library GraphHopper that finds shortest paths within OpenStreetMap data.

2 Related Work

The data-set of New York Taxi's for the year 2013 was made publically available. This data-set had the potential to be used in wide array of research. The data-set provided the challenge that it was massive in size and contained information that could be of interest to a lot of people. When we selected this topic we wanted to find out how other people have utilized this data-set and visualized it. We came across a number of examples where this data-set has been used and visualized in form of a map. What we were able to find was that most of the research carried on the data-set dealt with the pickup and drop off points provided in the data and did not try to map the route taken by taxi themselves. We will discuss more about a couple of these projects. Thematic Map of all NYC Taxi Trips in 2013 [1] utilizes the data-set to show the distribution of all taxi drop-off locations in 2013 (170 million) within NYC. Visualization is achieved by dividing the map of New York City into different regions. These regions are clickable and color coded based on the number of pick up and drop off points in them. Clicking on one of the region shows more details like average fare, average number of passengers etc. Clicking on a region also displays to what other region people traveled to or from based on color codes. There is no option to change time of the day or to look at specific streets or road with in the selected region. Hubcab is an interactive visualization that invites you to explore the way in which over 170 million taxi trips connect the city of New York [2]. Hubcab does this by plotting the pickup points with yellow dots and drop off points as blue dots. As mentioned earlier this visualization deals only with pick up point and drop off points and does not calculate the route taken by the taxi. Time is divided into chunks of three hours and the user can switch between them using an interactive clickable dial. The user can drop off a pick up marker at any point on the map. Doing so provides additional details about a radius around that point (user has the option to select from three different radius sizes) including total number of pickups, average distance covered and average duration of the ride for pickups with in that radius. Similarly the user can place the Drop off marker at one of the highlighted areas on the map once the pickup marker has been placed (the drop off marker cannot be dropped at areas that are not highlighted). Doing so does not only provide more information about the drop off location but also provide information about how much money and CO2 emission

[1] `http://nyctaximap.appspot.com`

[2] `http://www.hubcab.org/`

people could have saved by sharing rides instead of taking a taxi. The last project we would like to discuss is NYC Taxis: A Day in the Life by Chris Whong [3]. This visualization displays the data for one random NYC yellow taxi on a single day in 2013. See where it operated, how much money it made, and how busy it was over 24 hours. This is the only project we came across that actually maps the route taken by the taxi driver and not just the pickup and drop off points. This web app visualize the day of a single taxi as it randomly picks a taxi and plots the route followed by the taxi throughout the day, providing information like fare,charges,taxes,tips and the total number of passengers picked up so far along the way. Like our project, this web application also uses Mapbox for its visualization. On the other hand unlike our project this application does not take the complete data-set into account. 30 random cabs per day were queried to collect the data. Moreover the route calculated for the visualization was based on Google's direction API. The best choice route provided by Googles direction API was selected and this was not compared against the distance stored in the data, hence the route might not be exactly the same as used by the taxi driver. As we can see based on the projects we have discussed so far, most of the visualization has been carried out on pick up and drop off points. The only project that calculated the route did so on a much lower scale and only utilized the data of one taxi at a given time. This encouraged us further to work on this project, as our visualization of this data might be the first of its kind

3 Research Questions

The main research question is "What are the utilization of each street-segment taken by taxi's based on the pickup and drop-off coordinate in the 2013 taxi log-files".
To answer this question we need to abstract the city of New York to a graph of its street-network where each street is discomposed into an edge between each intersection.
The other research question is how to do this in a way that this task is complete within a feasible time with regard to the time-line of the project. Since we are dealing with about 220 million queries, that can easily take around 1 millisecond of processing time each, we are dealing with an estimated execution time measured in days when executed without parallelization.

[3] `http://nyctaxi.herokuapp.com/`

4 Project Setup

4.1 Parser

The first step taken was to build a parser that transformed each trip in the comma-separated (CSV) data-set into a plain-old Java objects for further-on processing.

4.2 Routing

The second step was to adapt the GraphHopper library so we could access it one level of indirection below the publicly exposed API. For the routing library we picked GraphHopper, since it is a small light-weight routing library written in Java, which allows us to neatly integrate it into our Java query program. This avoids us a lot of complexity around inter-process communicating and mixing native code with Java. Another benefit is that since it's relatively light-weight we can modify the library for our purposes without too much complexity. Finally we already had some experience on this library prior to this research project.
By default the library returns the geometry of the entire path and route instructions and to answer our research question we were only interested in the unique identifier for each edge on the path. Using this newly exposed API we build a new API for our object that returns the hour-of-day when each edge was visited based on the input of start-coordinate, end-coordinate, start-time and end-time. The time of day was naively determined using the progress based on distance since no historic traffic data was available, making it not possible to determine more accurate timing information.
On a few experiments using a sub-set of the data we determined that the initial results of this API based on GraphHopper were flawed since it returned temporary edges and edges that not just represented a single street segment. The latter where created by the Contraction Hierarchy[1] algorithm, which introduces a different graph structure that includes level of hierarchies and shortcut-edges. After we disabled the Contraction Hierarchy we observed a execution time that was 6 times as large.
Another problem was that the current version of GraphHopper includes edges that it generates on the fly to account for complex intersections and a temporary edge for the initial and final edge to split edges at the exact start/end coordinate. To resolve this issue we resorted to an very old version of GraphHopper that did not include this functionality and did have a

representation where each street-segment mapped to an edge.

4.3 Query using Spark

After completing these two main components we moved to build a query using Spark. We picked Spark since it was in Java, well supported by the cluster we used and it included functionality such as parallelization, countbyvalue. The basic structure of our query consists of these parts.
First we open-up the files from the HDFS cluster, we re-compressed the files as gzip and later on as bzip2 since Spark can read gzip and bzip2 by default whereas for the original zip-format we had build a decompression of our own. To avoid the first line of each CSV file that contains the name of each column we use the filter-function of Spark. Each line of the CSV input is then mapped into a Java object containing the only information we need later on: `pickup-time, drop-off-time, pickup-coordinate, drop-off-coordinate and trip-distance`. If we are unable to parse a line of CSV we return a null-pointer that is filtered on in the next step of the phase where we filter all null-pointers and trips where the pickup-time is later than the drop-off-time which are impossible to have happened in real life due to physical restraints.
Then we are on the core-phase of our query where we use the flatMap function to map each trip into a list of tuple's with the edge-id and the hour-of-day. If we count these tuple's we have sufficient data to create an heatmap. At first we used the `.countByValue()` function however after an `OutOfMemoryException` we found out that this function takes all the (edge,hour-of-day) tuples to one partition to count them there, this resulted in too large dataset to fit in memory. On StackOverflow [4] we found another approach where Spark executes the count-aggregation on each partition, reducing the size of the dataset considerably. To this we first wrap each tuple in another triple to keep the count and then reduce those triples based on the key of the tuple `(edge-id, hour-of-day)` using the `reduceByKey()` function to triplet with the count of each `(edge-id, hour-of-day)` inside the partition.

4.4 Rendering

To make it easier for other applications to actually do something with this data we added an additional render-phase to transform it using Spark into

[4]`http://stackoverflow.com/questions/25318153/spark-rdd-aggregate-vs-rdd-reducebykey`

the open standard called GeoJSON, which is a open standard format to represent geographical features. in the output GeoJSON each edge is a feature inside the main feature-collection. For each edge we include the complete geometry and the number of taxi's for each hour-of-day as properties. This way we can render our result data using numerous application such as QGIS and Mapbox out of the box.
To complete this rendering phase we execute multiple transformations using Spark. First we group all `(edge-id, hour-of-day, count)` by their edge-id, then we transform this list of tuples per edge into a single GeoJSON feature, for each hour-of-day we set the number of taxis and the geometry is extracted from GraphHopper using a extended API. These GeoJSON features are then grouped into a single collection and written as a GeoJSON document to the HDFS cluster. We can then retrieve our result and render it in a GIS application such as QGIS.

4.5 Visualization

The next step was to visualize this data in an easy to use and interactive web application. For this purpose we looked at different available libraries. Two libraries that seemed most suitable for our application were OpenLayers and Mapbox Gl. After comparing both these libraries and going through their Pros and Cons, it was decided that Mapbox GL would serve our purpose the best. The reason Mapbox GL was selected was because creating custom styles and layers is quite easy using their online studio. The user can create a new vector tile set of data by uploading their data. The user has option to import their data in a wide variety of formats including shapefiles, GeoJSON and CSV. Once the tile set has been uploaded and processed by the Mapbox studio, the user can create styles and layers based on this data. Despite ease of use, Mapbox GL also uses WebGl which helps render data in a smooth and responsive way We decided the best way to display the data is to draw lines across the routes the taxi took and color code them to make it easy for users to visualize the data. Moreover we also thought it would be best to separate the data based on time of the day (in hours) so that user can see change in trends of traffic based on different time slots. So we decided to implement the visualization for each one of the 24 hours in a day. We decided to implement a slider on top of the page to let user select which time of the day they wants to view the. To accomplish this we had a few different options in Mapbox. Mapbox allows users to create different styles and on top of those styles they can create different layers. The first option was to create a base view that just contains the map of New Your city along with its roads and streets. Each hour and its heat map can be represented as a group of layers. Each layer within an hour represented a range of data filtered on the basis of number of taxis. Each layer is also give its own color code to help visualize data in a better way. Based on which hour the user has selected on the slider, the layers corresponding to that particular hour would be loaded while the previous layers would be removed from the display. This approach works perfectly fine in theory but in practice we ran into some problem. The issue we faced is that each layer represents large amount of data, removing a layer may be fast but displaying a new one turned out to be quite slow. The reason behind this is that the layer itself just stores the information about what data to show from the entire dataset based on filter. The data set itself is hosted on Mapbox's server. Hence loading a new layers takes time to be displayed. And based on our color code (we use 10 separate color codes), for each hour we have to add 10 new layers on the map. And the more times we remove and load new layers, the slower the application tends to get. Our second approach was to preload all the layers on top of the base view, but setting the visibility of only the first layer as true. Once the user selects a different time slot, the layer corresponding to that time would be displayed while visibility of all other layers would be set to false. Using this technique we were able to drastically reduce the time between switching layers and it was much more responsive now, but it significantly increased the initial loading time of the web page. Initially the page would take up to 8-10 seconds to load before the first view of map was displayed, this was too slow and we decided to scrap this idea as well. Another possible solution available to us was to use styles. Mapbox lets you create styles using its studio based on the vector toilets. Styles can have multiple layers added to them. These styles can either be hosted on Mapbox's server or can be downloaded as JSON files. Mapbox free account allows each user to host 10 styles only. We decided to create one style for each hour of the day. Each style has layers representing different range of data color coded to make it easy to visualize. We decided to download these styles as JSON and, unlike our previous attempt where we just switched between layers, decided to try and switch the entire style of the map based on the hour selected. As it turns out, removing a style and adding a new one is much faster than removing and adding layers. The only problem was that removing and adding new styles causes the entire map to get refreshed and the map goes blank for a split second. We would have really preferred having a way where we could have gracefully shifted from set of data to another without refreshing the entire map, but considering the size of our data we had to settle for changing styles. Once we were able to render the data on a nice responsive map and created a legend that helped user understand the color codes, we decided it would be nice if the user can actually get to know the exact number of taxis went through a certain section of the road at a given time. For this purpose we implemented a function that would get information about the layer under the user's cursor and get the relevant information out of it and display the total number of taxis that went through that point for that given point of the day. This way the user can not only have a general idea about the number of taxis based on color codes, but also get the exact figures.

5 Experiments

5.1 Experimentation on input-format for Spark

We did some experimentation on how Spark handles different compression formats with regard to parallelization. Spark has built-in support for both gzip and bz2 compression, however the default gzip compression does not support splittable files meaning that only one executor can work per file. With bz2 compression Spark uses multiple executors working on different parts of each file, allowing 63 executors to work on our data instead of just 12. Since our input data was very parallelism, each line could be processed individually this was a 5-time increase in process-speed.

5.2 Experimentation with map coloring

When it came to visualization, we had to decide how to color code our map. The first basic idea we had was to have 10 different color codes and divide them evenly. For a given hour we found out the maximum number taxis on any given road and divided it by 10. Hence our color codes were divided into 10 equal chunks. We thought this way we would be able to get the best distribution, but once it was implemented, we saw that majority of the map fell in the very first category of least traffic. In the entire map, only two road sections of the map fell in the highest traffic load categories. On further inspecting the data we realized that the distribution of traffic was not evenly divided. Majority of the roads especially residential areas had quite a low road of taxis on them. Contrary to that very few roads had large amount of traffic. For example 14th street had the greatest number of taxis on it and it was much higher than other roads around. Hence by dividing the data set equally among the 10 categories we were not able to get much representation of data and majority of the roads were green (lowest amount of traffic) with very few roads showing other colors. So we decided we should change the distribution, we decided the categories based on fixed ranges, we had to try out different ranges and fine tune them to get a visualization that seemed to represent the data in the most understandable way. After trying out with different intervals, we decided that a range of 20,000 vehicles seemed to represent our data in the best way possible. So now each category is has a range of 20,000 and has its own color code which the user can see through the legend as well

6 Conclusions

With the help of this project, we can not only see which road had how many taxis on it at any given hour but also compare them against different hours. We can see that trafic in centeral NYC goes down at around 3 am and is at least at 5am. Morover this visualization also helps us find areas that had more taxis going through them. For example in 14th street seems to have maximum amount of taxis on it at any given time. So this is a hot place for taxi picks up and drops off. Another thing we can observe is the heatmap of taxis near the two airports, the JFK airport and La Guardia Airport. It can be seen that the load of traffic to and from JFK is low at night and day time and it gets heavier over the evening. As compared to that La Guardia airport sees large amount of taxi movement during the day time in office hours. This might give us a little insight on what time do these airport have more flights coming in or going away. Simiarrly at 8 pm we see the most amount of taxis in Centeral NYC with massive amount of taxis passing by the 7th Avenue,14th street and Centeral Park which are all tourist locations. This helps us judge which places do people prefer going during the evening. As it clear from the map, famous tourist location has more flow of taxis. Thus using this map we can not only about the concenteartions of taxis across the city but also learn about the where do people hang out most in the evening, What places are more visited during working hours and what time are the airports the busiest

References

[1] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In *Experimental Algorithms*, pages 319–333. Springer, 2008.