Ride-Matcher Architecture

Sana Shaikh
Student , Computer Engineering
Trinity Academy of Engineering
Pune , India
sanashaikh0701@gmail.com

Alfiya Shahbad
Asst. Professor , Computer Engineering
Trinity Academy of Engineering
Pune, India
alfiyashahbad.tae@kjei.edu.in

***Abstract*—The daily home-office commute of millions of people in crowded cities puts a strain on air quality, traveling time and noise pollution. This is especially problematic in western cities, where cars and taxis have low occupancy with daily commuters. To reduce these issues, authorities often encourage commuters to share their rides, also known as carpooling or ridesharing. To increase the ridesharing usage it is essential that commuters are efficiently matched.**

***a) :* In this paper we present RideMatcher, a novel peer-to-peer system for matching car rides based on their routes and travel times. Unlike other ridesharing systems, RideMatcher is completely decentralized, which makes it possible to deploy it on distributed infrastructures, using fog and edge computing. Despite being decentralized, our system is able to efficiently match ridesharing users in near real-time. Our evaluations performed on a dataset with 34,837 real taxi trips from New York show that RideMatcher is able to reduce the number of taxi trips by up to 65distance traveled by taxi cabs by up to 64trips by up to 66**

***Index Terms*—Keywords: Ride Sharing System,Ride Mtcher.**

I. Introduction

Ridesharing is an important way to reduce traffic congestion and travel costs for commuters around the world. Ridesharing is also environmentally friendly, as sharing journeys reduces carbon emissions. To encourage ridesharing, many countries employed high-occupancy vehicle lanes, which are traffic lanes restricted to vehicles carrying more than one passenger .

a) : Recently, several online ridesharing platforms that match commuters with drivers have been put in place. There are two major categories of such platforms: offline and real-time. Offline platforms work by keeping databases of commuters and putting into contact those who have similar routes to work. Examples include Zimride, which provides ridesharing solutions for companies and universities, or BlaBlaCar, which focuses on sharing long-distance trips between cities. Real-time ridesharing has seen a significant increase in popularity recently, with the emergence of mobile applications like Uber , Lyft or Via. These applications make use of complex routing algorithms to determine if trip requests from many users can be served by the same driver. Additional features like GPS tracking, automated payments and the validation of drivers through feedback make these applications extremely attractive to customers. However, they rely on occasional drivers, which makes them less suitable for commuters, who prefer a reliable and long-term transportation method.
One thing that all the above systems have in common is that they rely on traditional cloud infrastructures to provide their services. This has several advantages, like good reliability and usability, which stem from the fact that cloud computing is a mature technology. However, the emergence of Internet of Things and the recent developments in the area of mobile computing created a demand for highly responsive cloud services. To fulfill this demand, a new paradigm, called fog computing, was introduced . Fog computing can be seen as an extension of cloud computing, in which computing nodes are less centralized and placed closer to the sources of data, which leads to better scalability, shorter response times and increased fault tolerance.We notice that all these advantages of fog computing are also requirements in a ridesharing platform, as it has to process large amounts of mobile data in a timely manner.

b) : In this paper we study the possibility of designing a decentralized ridesharing system that can be deployed on fog infrastructures. In this system, participants equipped with mobile devices (e.g., smartphones) connect in a peer-to-peer fashion in order to share their rides. As opposed to a traditional cloudbased service, participants do not use a central database to find available rides. Instead, they use short range communication technologies, like Bluetooth or WiFi Direct, to discover other participants that provide rides matching their needs. Just like a person who goes in the street to find a taxi, the system running on a mobile device scans the surroundings and attempts to find rides that match a desired route. To further improve the chances of finding good rides, the system employs a gossiping technique that uses a mesh network on top of the fog infrastructure. This mesh network is used by the participating mobile nodes to advertise and find available rides. When two or more mobile nodes determine that they can share a ride, they organize themselves into a ridesharing group. This happens automatically and autonomously, without any mediation.

c) :

II. Ride Matcher System

In this section we describe the components of the Ride-Matcher system and the interactions between them. The main feature of our system is decentralization. In RideMatcher, all mobile nodes (e.g., personal cars, taxis, pedestrians) perform only local operations, without relying on a central coordinator. The goal of each mobile node is to group with other mobile nodes that have similar routes at a certain time, such that sharing a car ride with these nodes cuts down the cost of the trip. In pursuing this goal, each mobile node is selfish, being interested only in obtaining a higher benefit for itself, without caring about other nodes in the system. However, in order to ensure that the system converges to a stable state, we define a set of rules in order to prevent situations when nodes keep grouping and ungrouping indefinitely. Our system is based on our earlier PeerMatcher system . However, as stated in section II, there are several changes that have to be made in order to adapt PeerMatcher to the ridesharing problem:

1) Peer Discovery. Mobile nodes need a mechanism to discover other nodes in the system. This task is not trivial, as there is no central service that can provide them with information about other participants. 2) Matching Car Rides. When mobile nodes find each other in the network, they need a way to assess the similarity of their rides. For this purpose, we use the weight function WR defined previously. 3)Flexible Groups. Mobile nodes should be able to organize in groups of different sizes. Therefore, we cannot it allows mobile nodes to learn about rides of other nodes they did not discover directly. Therefore, the chances of finding a matching ride increase considerably.

a) : In order for the gossiping protocol to work properly, any two nodes in the peer-to-peer network should be able to connect with each other. To realize this, we assume that all the mobile nodes are permanently connected to an edge, fog or cloud infrastructure. The only requirement for this infrastructure is to provide a reliable communication medium for the mobile nodes, as our system does not rely on any centralized service or database. This has the advantage that if a part of the infrastructure goes down, the RideMatcher system will continue to work for the nodes that do not rely on that part. Scaling the infrastructure up is also easier, as only the networking part has to be scaled. In RideMatcher, the computation is performed by the mobile nodes, without relying on any centralized computing facility. This is possible thanks to the fact that RideMatcher is very lightweight, in terms of both computation and communication.

III. MATCHING TAXI RIDES

To evaluate our system, we implemented RideMatcher in the PeerSim simulation environment [12] and tested it on a set with 34,837 taxi rides from New York. A. PeerSim PeerSim is a peer-to-peer simulator that assists the implementation and testing of peer-to-peer protocols on a large scale.

1) Algorithm:

```
1: function NEXTCYCLE()
2: refreshKnownNodes()
3: if node is not in ridesharing group then
4: group ← findSuitableGroup()
5: if group is not null then
6: joinGroup(group)
7: end if
8: else if node is in ridesharing group then
9: group ← findBetterGroup()
10: if group is not null then
11: if group is not full then
12: joinGroup(group)
13: else
14: swapGroup(group)
15: end if
16: end if
17: end if
18: end function
```

PeerSim is lightweight and easily configurable, allowing the simulation of networks of more than 107 peers structured in various topologies. To make the simulations more realistic, PeerSim supports dynamic scenarios such as churn or node failures.

a) : One particular feature of PeerSim is that it supports stacking multiple protocols on each peer. This is especially useful in the case of RideMatcher, where each node has to run the gossiping protocol and the matching protocol simultaneously. In PeerSim, this can be done easily by creating two classes that extend the Protocol abstract class, which incorporates common methods that are useful for any peer- to-peer protocol, like sending messages to other peers or executing periodic tasks. PeerSim also facilitates the exchange of data between multiple protocols that are stacked. In our case, the gossiping protocol periodically informs the matching protocol about new mobile nodes that were discovered.

b) : In PeerSim, protocols work in rounds (or cycles). Each protocol has to define a method called nextCycle, where it declares a set of actions that have to be taken in each round. In PeerSim, these rounds are executed one after another, without interruptions. However, in a real-world application, rounds are executed at a predefined frequency. Choosing the right frequency depends on the usage scenario. While a high frequency would give better performance, a low frequency would determine less resource usage. Fig. 4 shows the actions taken by the matching protocol in each round. We have already discussed them in the previous section.

c) : Taxi Rides Dataset We evaluate our RideMatcher system using an extensive dataset of taxi rides from New York. The dataset contains 34,837 yellow taxi rides that took place on 10 June 2016 between 7 a.m. and 9 a.m., which is the typical rush hour in New York. We also considered rides from other days randomly picked from March, April and May 2016. Since the results were similar, in this work we focus only on the dataset from 10 June. The following details are provided for each ride in the dataset: start time, end time, origin coordinates (latitude and longitude), destination coordinates, cost, distance traveled and passenger count. For this dataset, we focus on the typical carpooling scenario, where people share rides to work in the morning. Therefore, we omit in our experiments those rides that have an airport as origin or destination (these are excluded in the 34,837 rides). Given that the dataset contains only rides by yellow cabs, most of the rides were done in Manhattan. We also considered using data from green taxi cabs, which operate mostly outside Manhattan, but the data was insufficient, as there are usually only 3,000-4,000 green cab rides during rush hours. By running RideMatcher on this dataset, we aim to match as many taxi rides as possible in order to reduce the traffic caused by taxi cabs and also to reduce the cost of the rides. The matching is done using the weight function WR discussed in the previous sections. We match cab rides based only on their actual routes, without taking into account empty vehicle repositioning (when a cab is on the way to pick up a passenger). In our simulations, we also take into account the number of passengers carried by each cab, as discussed in Section IV-C. For most of the simulations, we consider that the maximum capacity of each cab is 5 passengers, as required by law in New York.

IV. Implementation

To be able to use the taxi rides dataset, we have to simulate the information exchange between taxi cabs in traffic. This helps us test whether our peer-to-peer system would be effective in real-world situations. This is not directly attainable using only the rides dataset, as it only contains data about the origins and destinations of the rides, without providing any information about their routes. Therefore, our approach is to first generate routes for each ride, and then compute the intersections between taxi cabs in traffic.With this information, we can build a graph where each node represents a ride operated by a taxi cab and each edge represents an intersection between two rides in traffic. RideMatcher uses this graph to initialize the gossiping protocol, which uses the graph to further disseminate information about taxi rides in the network. Finally, the nodes from the graph use the disseminated data about rides to form ridesharing groups.

a) : We use the GraphHopper library to compute the routes for the rides. GraphHopper works by taking as input a map in OpenStreetMap format and building a graph of roads, in which each edge represents a road segment and each node represents an intersection. We use this graph to compute the routes for all the rides in the dataset. Each route is represented as a list of road segments and a list of intersections. In RideMatcher we use this list of intersections to determine whether any two rides intersect. Since the duration of each ride is available in the original dataset, we divide this duration evenly among the segments of the ride’s route and compute the timing for each intersection point of the route. Finally.

we compare the intersection points and timings of every pair of routes and determine which rides intersect. We consider that two rides intersect if they have at least one common intersection point and the timings of that intersection point differ by less than a predefined threshold. In our simulations we use a threshold smaller than 2 minutes, which is the duration of a typical traffic light cycle.

b) : Fig. 1 shows the components of our simulator. Initially, the taxi cab intersection graph is computed as described above. This graph is then used to initialize the gossiping protocol on all nodes. During the execution, the gossiping protocol and the matching protocol work in parallel, with the matching protocol periodically querying the gossiping protocol about new rides that were discovered. When new rides are

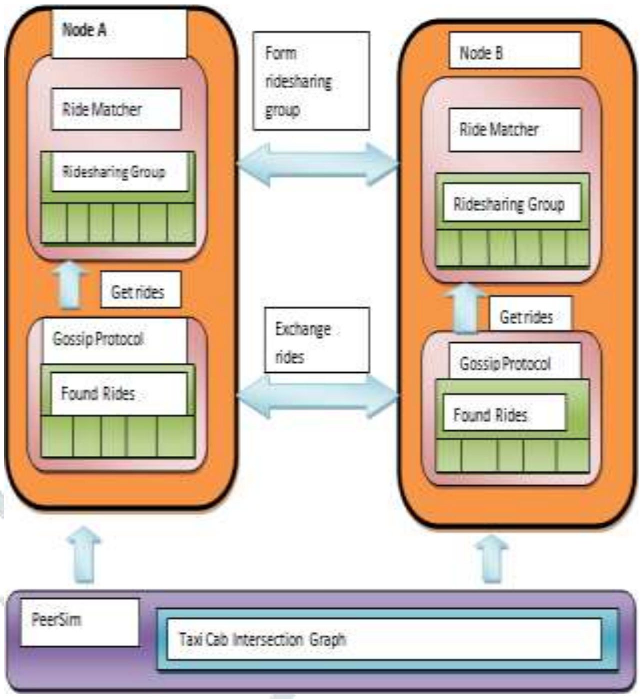

Fig. 1. RideMatcher Simulator Architecture.

available, the matching protocol attempts to form ridesharing groups or to swap groups using the rules described in the previous section. Whenever a ridesharing group is formed or changed, this information is disseminated through the network by the gossiping layer.

V. RELATED WORK

In this section we review the contributions in the field of ridesharing, with a focus on large-scale dynamic ridesharing systems and goal-oriented systems.

A. *Large-scale Dynamic Ridesharing Systems*

Like RideMatcher, these systems address the ridesharing problem at city-scale. T-Share proposes a taxi searching and scheduling algorithm that uses a spatio-temporal index to serve dynamic taxi cab queries in real-time. The focus of the system is to reduce the total distance traveled by cabs and also to increase the query processing throughput. The system is later extended to take into account the monetary implications of ridesharing . The problem of reducing the cost of sharing rides is further elaborated in , which proposes a greedy randomized adaptive search procedure (GRAPS) to reduce the cost of the trips. Huang et. al. focus on the user’s benefit, by proposing a system that lets the user define waiting time and service time constraints.

Like the systems above, RideMatcher is able to operate on a large scale while taking into account metrics like cost and traveled distance. Additionally, our system has the advantage of operating in a decentralized way, which further increases scalability and response time.

B. Goal-oriented Ridesharing Systems

The complexity of the ridesharing problem makes it difficult to design a system that has all the benefits. Therefore, systems usually focus on optimizing just a few ridesharing metrics. For example, CallCab aims at increasing the availability and affordability of taxicab services by using historical data to learn the usual routes that cabs take and using this data to increase the opportunities for ridesharing. Other systems focus on using bigger cabs to reduce taxi traffic , increasing matching stability at the expense of matching optimality , or preserving the user's privacy by taking an infrastructure-less approach . A general method for assessing the benefits of vehicle pooling is presented in . This method uses shareability networks to model the collective benefits of ridesharing, with a focus on reducing the travel time of the shared rides. While we designed RideMatcher as a general ridesharing system, it can be easily customized to serve specific goals, like the above systems. This can be done by adapting the function used for matching rides to the desired goal.

VI. CONCLUSION

Sharing vehicles for daily commutes results in less traffic. If ride sharing is applied in dense populated areas, a direct noticeable impact on travel time, noise pollution and air quality can be noticed. This paper introduces RideMatcher, a distributed ridesharing system designed to exploit fog computing. Using a dataset comprising 34,837 taxi rides from New York, we show that we can reduce the number of taxi rides by up to 65show that RideMatcher reduces the distance traveled by cabs by 64system directly applicable on large scale for commuters around the world while saving trip costs. While the promising results presented here are limited to simulations, we plan to implement RideMatcher as a complete solution for mobile devices assisted by fog.

.

VII. REFERENCES

[1] 1) N. V. Bozdog, S. Voulgaris, H. Bal, and A. Van Halteren, "Peer matcher: Decen-tralized partnership formation," in Self-Adaptive and Self-Organizing Systems(SASO), 2015 IEEE 9th International Conference on. IEEE, 2015, pp. 31–40

[2] 2) M. Mallus, G. Colistra, L. Atzori, M. Murroni, and V. Pilloni, "Dynamic car-pooling in urban areas: Design and experimentation with a multiobjective routematching algorith," Sustainability, vol. 9, no. 2, p. 254,2017.

[3] 3)Bozdog, N.V., Makkes, M.X., Van Halteren, A. and Bal, H., 2018, May. Ride-Matcher: peer-to-peer matching of passengers for efficient ridesharing. In 20184 NAME OF SEMINAR18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Comput-ing (CCGRID) (pp. 263-272). IEEE.

[4] Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in Peer-to-Peer Computing, 2009. P2P'09. IEEE Ninth International Conference on. IEEE, 2009, pp. 99–100.