RideMatcher: Peer-to-peer Matching of Passengers for Efficient Ridesharing

Nicolae Vladimir Bozdog, Marc X. Makkes, Aart van Halteren, Henri Bal
Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
{n.v.bozdog, m.x.makkes, a.t.van.halteren, h.e.bal}@vu.nl

***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.**

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 65%, the distance traveled by taxi cabs by up to 64%, and the cost of the trips by up to 66%.

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 [1].

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 [2], which provides ridesharing solutions for companies and universities, or BlaBlaCar [3], 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 [4], Lyft [5] or Via [6]. 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 [7]. 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.

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 cloud-based 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.

We identify two key challenges in designing such a system. First, participants need a way to enter the system and discover others with similar routes without relying on a central database. In densely populated cities, this can be done using short-range communication technologies, like Bluetooth or WiFi Direct. These techniques improved considerably in recent years, being now able to operate reliably over long ranges. For example, Bluetooth 5 devices are now able to transfer

data at 50 Mbit/s over distances up to 240 meters [8]. Also, our earlier research [9] proves the usability of Bluetooth Low Energy in remote sensing scenarios. Here, we combine these techniques with a fog-based peer-to-peer gossiping protocol in order to build a reliable mesh network in which users can find ridesharing partners.

The second key challenge is to match the users based on their travel requirements. This is particularly difficult in a distributed setting, where matching operations have to be synchronized in order to avoid inconsistencies. We address this challenge by extending PeerMatcher, our previous protocol for distributed partnership formation [10]. With PeerMatcher it is possible to partition a weighted graph into groups of a fixed size, such that the weights of the groups are maximal. We adapt this idea to the ridesharing problem by assuming that each group consists of two or more participants that share a ride. With this approach, our new system, called RideMatcher, is able to quickly match ridesharing requests based on their routes, without relying on a centralized service. In this work we focus only on matching requests having routes that fully match (from one end to the other). We do not cover the case when rides match only partially, as it was already addressed in earlier research [11].

In summary, our contributions are as follows:

- We propose a peer-to-peer system, called RideMatcher, for matching ridesharing users based on their travel preferences.
- We implement a taxi ride simulator in PeerSim [12] based on a dataset with 34837 real taxi trips from New York.
- We evaluate the effectiveness and performance of RideMatcher under different taxi ridesharing scenarios. The results show that our system can reduce the number of taxi trips by up to 65%, the distance traveled by taxi cabs by up to 64%, and the cost of the trips by up to 66%. Also, RideMatcher is able to find shared rides for 86.4% of the participants and reduces the number of taxi rides with a single passenger by 95%.

The remaining of the paper is structured as follows: Section II presents previous work that we base upon, Section III explains how we model the ridesharing problem, the RideMatcher system is described in Section IV, implementation details about our simulator are provided in Section V, we discuss the results of our evaluations in Section VI, related work is examined in Section VII and Section VIII concludes the paper.

II. Background

We take as starting point our previous research on a peer-to-peer algorithm for solving the *k-clique matching problem* [10]. The goal of this problem is to find all non-intersecting k-cliques in a weighted graph, such that all the cliques have maximal weight. Despite being a NP-hard problem, our peer-to-peer algorithm is able to find very good solutions by the use of heuristics.

We notice that the problem of matching car rides is an instance of the k-clique matching problem, where each node in the graph is represented by a ride, each weighted edge represents the similarity between two rides, and k is the number of passengers each car can carry. Therefore, we study the possibility of adapting our earlier PeerMatcher algorithm [10] to the ride matching problem.

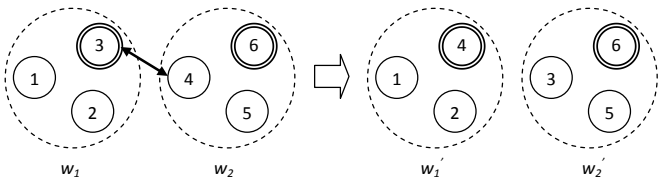

Fig. 1. Swapping in PeerMatcher [10]

We identify three main changes that have to be made: i) we need to find a way for nodes to connect with other nodes upon entering the system, since in PeerMatcher this is done artificially at initialization ii) we have to define a weight function that assesses the similarity between car rides, and iii) since cars can have different capacities, we have to make the algorithm work with cliques of different sizes. Before addressing these changes in detail, let us briefly explain first how PeerMatcher works.

In PeerMatcher nodes organize themselves into cliques of size k, such that the weights of these cliques are maximal. In order to do that, nodes can group with other nodes, move to other cliques, or swap places with nodes from other cliques (Fig. 1). The system reaches the stable state when no operation that improves the total weight of the system can be performed.

PeerMatcher is a decentralized algorithm, which means that all operations happen locally between nodes. At the beginning, each node starts with a random set of neighbor nodes. During the execution, these lists of neighbors are exchanged between nodes using a gossiping algorithm, such that each node learns in time about the other nodes in the system. All operations that lead to the formation of cliques are synchronized locally in order to avoid inconsistencies.

The *k-clique matching problem* has many applications, from matching ridesharing users to matching users in a social network, bundling products in a shop or grouping resources in a datacenter based on demand. Having a distributed approach for solving the problem has the added benefits of increased scalability, better fault tolerance, and fairness, by eliminating the need for a central mediator. In the remaining of the paper we focus on adapting PeerMatcher to the ridesharing problem.

III. System Model

We model a city map as an undirected graph $G = (V, E)$, where V is the set of vertices (intersections) and E is the set of edges (roads) connecting different points. In addition G is mapped on a geographic coordinate system.

We consider a set of $\mathbb{M} = \{m_1, m_2, \ldots, m_n\}$ mobile nodes which are at any given time on graph G and a set of rides $\mathbb{R} = \{r_1, r_2, \ldots, r_p\}$. Each ride $r_i \in \mathbb{R}$ has a start point,

denoted by $S(r_i) = (x_s, y_s)$ and endpoint $F(r_i) = (x_f, y_f)$ where (x, y) are the latitude and longitude coordinates.

Each mobile node m_i can connect, via Bluetooth, with another mobile node m_j if and only if $\text{dist}(m_i, m_j) \leq M$, where M is the maximum connection distance (e.g., 240 m with Bluetooth 5). Also, any two mobile nodes can connect with each other over a routed infrastructure (e.g., fog or cloud), assuming that they previously intersected or they learned about each other from other mobile nodes in traffic[1]. We assume that links in the routed infrastructure can experience transient failures and messages can get lost.

Our system matches rides based on their start- and end-points to reduce the traffic. More formally, we consider the case where two or more rides $r_i, r_j, \ldots \in \mathbb{R}$ have similar starting points and ending points. We say, different rides, i.e., $r_i, r_j, \ldots$, match if both start-points and end-points lie in bounding rectangles having diagonals D_{start} and D_{end}. In addition, we require that all bounding rectangles have a diagonal smaller than a maximum diagonal D_T (distance threshold). Hence, we call a group of rides that fulfill this requirement a *match* group. For example, in Fig. 2 the three rides (r_1 (blue), r_2(green) and r_3(purple)) form a group, as they have similar start and end points, having D_{start} and D_{end} less or equal than D_T. In this figure we represent each ride only by its starting and ending points, as we consider that rides take the optimal (fastest) route to their destination.

To compute D_{start} for the start points and D_{end} for the end points of a set of rides $r_i, r_j, \ldots$, we define the following functions:

$$D_{\text{start}}(r_i, r_j, \ldots) = \text{dist}(\min(S(r_i), S(r_j), \ldots), \max(S(r_i), S(r_j), \ldots))$$
$$D_{\text{end}}(r_i, r_j, \ldots) = \text{dist}(\min(F(r_i), F(r_j), \ldots), \max(F(r_i), F(r_j), \ldots))$$

where $dist((a, b), (c, d))$ function calculates the distance between two geographical points (a, b) and (c, d), on a map using the Haversine formula [13]. Euclidian distance can also be used if distances are short.

To assess the benefit of grouping together a set of rides $\mathcal{R} = \{r_1, r_2, \ldots\}$, we define a weight function as:

$$W_{\mathcal{R}} = \begin{cases} 0 & , D_{start}(\mathcal{R}) > D_T \\ 0 & , D_{end}(\mathcal{R}) > D_T \\ \dfrac{2D_T - D_{start}(\mathcal{R}) - D_{end}(\mathcal{R})}{2D_T} & , otherwise \end{cases}$$

This weight function is guaranteed to take values in the interval $[0, 1]$. A high value of this function indicates a high benefit for a group of rides, while a low one corresponds to a low benefit. A weight with value zero indicates that the rides from the group do not match.

[1]Our system includes a gossiping protocol that mobile nodes use to exchange information about other vehicles they met. This information includes the routes taken, number of passengers and cost of rides.

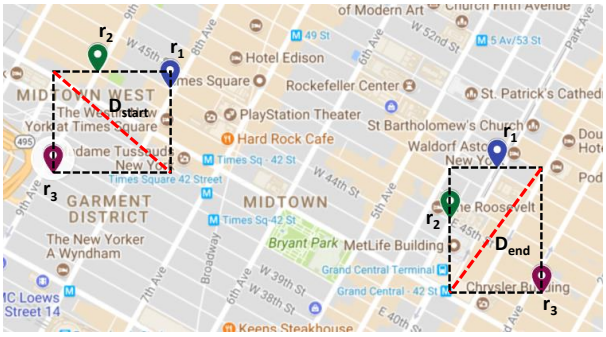

Fig. 2. Ride matching

The main reason for choosing this formula is that it has constant complexity, regardless of the configuration of the city where the RideMatcher system runs. An alternative would be to use the driving distance between the starting and the ending points of the routes involved. This could be useful in a ridesharing system where the driver has to pick up all the passengers. However, in this case complex routing algorithms have to be used, which could increase the complexity of matching rides. In cities with distinctive street patterns, like New York, the Manhattan distance can be used instead. Nevertheless, the weight function in RideMatcher is customizable, so it can be easily replaced with other variants.

IV. RideMatcher System

In this section we describe the components of the RideMatcher 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 [10]. However, as stated in section II, there are several changes that have to be made in order to adapt PeerMatcher to the ridesharing problem:

- **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.
- **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 W_R defined previously.
- **Flexible Groups.** Mobile nodes should be able to organize in groups of different sizes. Therefore, we cannot

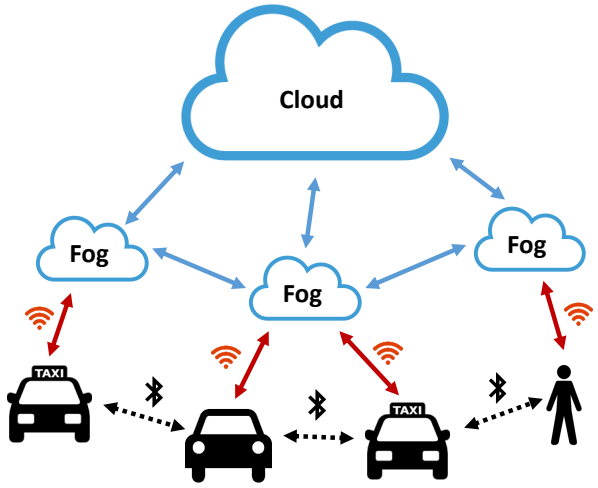

Fig. 3. Distributed communication in RideMatcher

apply the same method as in PeerMatcher, which is only able to find groups of a fixed size k.

In the rest of the section, we discuss the way we address these challenges and we describe the resulting RideMatcher system.

A. Peer Discovery

In RideMatcher, mobile nodes discover each other using short-range communication techniques like Bluetooth or WiFi Direct. These technologies are available in most modern smartphones and have evolved considerably in the recent years. We show later in the paper that using only ad-hoc connections between vehicles, it is possible to obtain a highly connected graph in a city with high traffic like New York.

Just like a person looking for a taxi in the street, a RideMatcher mobile node would scan the environment for available rides advertised by other neighboring nodes. When two nodes find each other, they connect and exchange their routes. If the mobile node is a person travelling to work, she will share her route to work. If the node is a taxi, then it will share its current route. If it is a pedestrian looking for a ride, she will share the desired route (Fig. 3). All these mobile nodes are treated equally by our system. When there is a match between the routes of two mobile nodes, they are notified about the match so that they can share the ride.

Relying only on ad-hoc interactions between mobile nodes in order to discover rides is most often not enough. Even in cities with high traffic the probability of finding a car with a similar ride as yours is low if only cars in the immediate vicinity are considered. To address this issue, in RideMatcher we employ a gossiping protocol whose role is to disseminate information about available rides in the peer-to-peer network. Particularly, RideMatcher uses Cyclon [14] for this purpose. In Cyclon, each peer keeps a small list of other peers that it knows. Whenever two peers connect, they exchange these lists of known peers in order to expand their knowledge about the network. In RideMatcher, this is useful as 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.

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.

B. Matching Car Rides

RideMatcher is a round-based peer-to-peer system, which means that all the actions performed by mobile nodes are executed in rounds. Every round, a node picks up and executes an action, depending on its state:

- If the node does not belong to a ridesharing group, it will look for a node that has a similar ride, or it will try to join an existing group.
- If the node belongs to a group, it will either try to move to a bigger group, as sharing the ride within a bigger group is more cost effective, or it will try to swap places with a node from another group, if the other group has a more suitable ride. We consider that a ridesharing group R_1 is more suitable than a group R_2 if its weight $W(R_1)$ is greater than $W(R_2)$ (see Section III).

The above operations are performed using the grouping protocol described in [10]. This protocol guarantees that the system stays all the time in a consistent state, meaning that any node can only be part of a single ridesharing group at a time. Also, the grouping protocol makes sure that the system converges to a stable state by defining rules to prevent nodes moving back and forth between groups. Basically, these rules state that a node is allowed to join a group or to move between groups only if the cumulative new weight of the groups involved in the operation is higher than the initial cumulative weight. While this greedy approach can cause the system to get stuck in a local optimum in some cases, we did not notice any significant negative impact in our evaluations.

A ridesharing group R can be formed only if the weight W_R of the group is larger than 0 (see Section III). This weight depends mostly on the distance threshold D_T we defined in the previous section. Basically, D_T represents the maximum distance that a person who shares a ride has to walk from her current position to the pickup point of the ride, and also the maximum distance from the drop-off point to the destination of that person. In RideMatcher we do not consider the case when

passengers are picked up on their way. While considering rides that match only partially (e.g., the route of ride r_A is included in the route of ride r_B) would be an interesting addition to our framework, this has been studied in the past [11], therefore it is outside the scope of the paper.

When computing the weight for a group of rides, RideMatcher also takes into account the time of the rides, so that only rides whose end times differ by at most a threshold T_T can be grouped together. We consider the end times as people are usually interested to reach a certain place, like school or job, by a certain time. Tuning this threshold determines the timeliness of the matchings. If a high value is used, then the probability of finding better matching rides is higher at the expense of finding rides occurring at different times. This can be useful in situations like sharing rides to work in the morning, when some people are more interested to find rides that match a desired route, as they can arrange later the time of the ride with the other people from the group. Alternatively, if T_T has a low value, then the probability of finding rides in real-time increases, while the quality of the matching rides decreases.

C. Flexible Groups

One important way in which RideMatcher extends our earlier PeerMatcher system is by supporting the formation of groups of different sizes. While PeerMatcher addresses the *k-clique matching problem*, in which groups of a fixed size *k* have to be found, in RideMatcher we have to eliminate this constraint, as vehicles can have different capacities. However, we still keep a parameter *K* that denotes the maximum capacity for the vehicles in the system. While most consumer vehicles can carry at most 4 or 5 passengers, we also consider in this work vehicles with smaller capacities, up to 2, and with higher capacities, up to 10. We choose this rather high range being motivated by previous research [15] that showed that using high-capacity taxis would greatly help in reducing the number of taxi cabs in New York.

When mobile nodes in RideMatcher advertise their rides, they also include the passenger count in the advertisement. For example, a couple looking for a ride would advertise 2 passengers, or a taxi carrying 3 passengers would advertise a count of 3. Whenever a ridesharing group is formed, besides checking that the rides in the group match, RideMatcher checks whether the total number of passengers is lower or equal to a maximum capacity *K*. We show later that the value of *K* has a significant impact on the efficiency of the system, with higher values leading to better results.

V. 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. PeerSim is lightweight and easily configurable, allowing the simulation of networks of more than 10^7 peers structured in various topologies. To make the simulations more realistic, PeerSim supports dynamic scenarios such as churn or node failures.

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.

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.

```
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
```

Fig. 4. The matching protocol

B. 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 W_R 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.

C. 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.

We use the GraphHopper library [16] to compute the routes for the rides. GraphHopper works by taking as input a map in OpenStreetMap format [17] 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 [18].

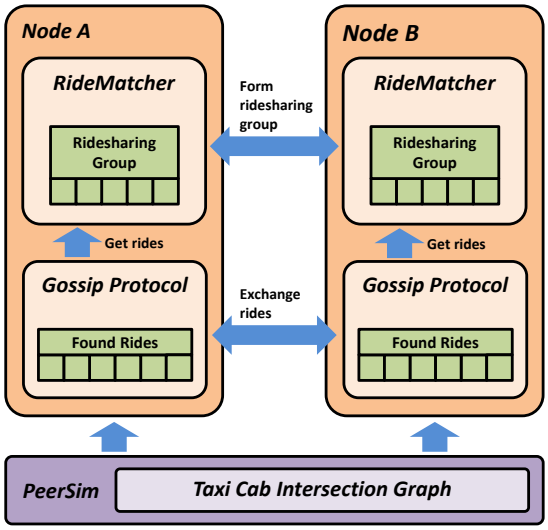

Fig. 5. RideMatcher simulator architecture

Fig. 5 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 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.

VI. Evaluation

We evaluate our RideMatcher system using the simulator presented in the previous section. The simulator is implemented as a standalone Java application. All the evaluations are performed on the DAS5 computing cluster from the Netherlands [19], which is composed of multiple computing nodes equipped with dual 8-core 2.4 GHz CPUs and 64GB of memory. Having this amount of computation power available was useful in our evaluations, as running simulations with large networks of mobile nodes can take up significant resources.

In our evaluations, we focus on the *effectiveness* and *performance* of our system. To study the *effectiveness*, we measure the taxi traffic before and after employing ridesharing,

the decrease in the cost of the rides and the reduction of the distance covered by taxi cabs. These aspects are mostly dependent on the parameters used for matching rides, therefore we analyze how the distance threshold D_T and the time threshold T_T impact the results. We also show that having taxi cabs of higher capacity makes ridesharing more efficient.

To evaluate the *performance*, we measure how fast our system converges to a stable state. The system is considered to be in a stable state if there is no available operation between nodes that would improve one or more ridesharing groups. The convergence speed depends mostly on the distance and time thresholds used for matching, therefore we vary these parameters and see how many rounds it takes for the system to converge. We also assess the scalability of the system by measuring how fast the gossiping protocol is able to disseminate data through the network.

In all experiments we use the taxi rides dataset presented in the previous section, which has data from 34,837 yellow taxi rides taking place between 7 a.m. and 9 a.m. For each experiment, we let the simulator run for a total of 24 hours. Unless otherwise stated, the distance threshold D_T is set to 800 meters and the time threshold T_T is set to the whole duration of each experiment. This means that by default only the similarity of the routes is taken into account when matching rides. Also, we set the default maximum cab capacity to 5, excluding the driver. This is based on the fact that yellow taxi cabs in New York can accommodate either 4 passengers (normal cabs) or 5 passengers (minivans). In our experiments, we do not put any constraint on the number of cabs with a certain capacity. As a result, we assume there are enough cabs of maximum capacity to accommodate all the rides.

We choose the above default values based on a typical carpooling scenario, in which participants are interested to find the best matching rides and are willing to walk at most 10 minutes from their origin to the pickup location and from the drop-off location to their destination. As this is sometimes not the case, we also show that our system performs reasonably well under more strict constrains. For example, we show that our system is still able to reduce the traffic by 29.6% for a distance threshold of 200 meters or to reduce the cost of the rides by 24.3% for a time threshold of 27 seconds.

A. Effectiveness

We start by looking at the impact of ridesharing on taxi cab traffic. To analyze this, we compare the number of taxi rides before and after combining them into ridesharing groups. We are interested to see how the taxi cab traffic is influenced by the parameters of our system, namely the distance threshold D_T, the time threshold T_T and the maximum cab capacity K.

Fig. 6 shows the traffic reduction as a function of the distance threshold D_T used for matching (see Section III). Each point in this figure represents an individual simulation. The dotted line corresponds to the maximum traffic reduction that is achievable (68.8%), which corresponds to the case when all rides have the maximum number of 5 passengers. We compute this by dividing the total number of rides by the maximum cab capacity (which is 5). We can see that RideMatcher is able to achieve a traffic reduction of 65.3%, only 3.5% less than the maximum. We also observe that a distance threshold of only 400 meters leads to a traffic reduction of more than 50%. For a better understanding, we can assume that the average person has a walking distance of 80 meters/minute, therefore a distance of 400 meters can be done in under 5 minutes. In the case of 800 meters, which is the typical walking distance, the traffic reduction is 63.7%.

The other parameter that affects the effectiveness of RideMatcher is the time threshold T_T, which denotes the maximum time difference between the end times of the rides of any two participants in a group. In the next experiment we vary this parameter between one second and two hours and measure the traffic reduction (Fig. 7). As can be seen, the traffic decreases rapidly with the increase of T_T. For a better understanding of the results, we use a logarithmic scale on the *y* axis. Even for a time threshold of just 1 second, our system is able to reduce the traffic by 3.9%. Increasing the threshold to 729 seconds (12 minutes) leads to cutting down the traffic by more than a half.

The maximum number of passengers allowed in New York's taxi cabs is 5. Nevertheless, we were curious to see whether the traffic could be reduced by increasing this limit. To this end, in Fig. 8 we vary the maximum capacity between 2 and 10 and compare the traffic reduction with the maximum possible reduction. Not surprising, the traffic reduction increases when the cab capacity increases, reaching a maximum of 76.9% for a maximum cab capacity of 10 passengers. More interesting is the small difference between the results of RideMatcher and the maximum achievable traffic reduction. This difference varies between 1.9% for a capacity of 2 and 7.1% for 10 passengers. While it is not feasible in the near future to introduce larger cabs in New York, in the long term authorities might consider introducing minibuses in order to reduce traffic congestion.

The taxi dataset also provides the cost and mileage of the rides. Therefore, it is interesting to see how these are affected by the distance threshold, time threshold and maximum cab capacity. As shown in Figures 9, 10 and 11, the variance of cost and mileage is similar to the variance of traffic. These results are obtained by comparing the sums of distances and prices of the rides before and after matching. While before matching we consider all rides, after matching we consider one ride per group, which is the ride with the shortest distance / lowest price in the group, depending on the comparison criteria. The difference between cost and mileage reduction can be explained by the dependence between the price and the mileage of taxi rides, which is not linear. Usually, taxis employ a starting fee that is independent of the distance traveled.

If we look at the absolute values of cost and mileage before and after ridesharing, we find out that the maximum cost reduction from Fig. 11 corresponds to savings amounting to $338,386, which is a significant value, given that the considered rides span over only 2 hours. These savings not only translate into benefit for the participants, but they also

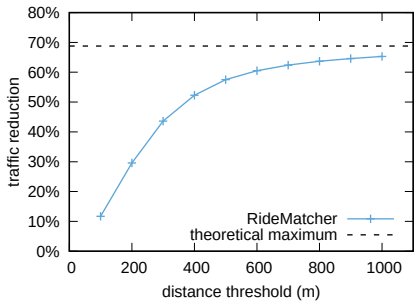

Fig. 6. Traffic reduction for different distance thresholds

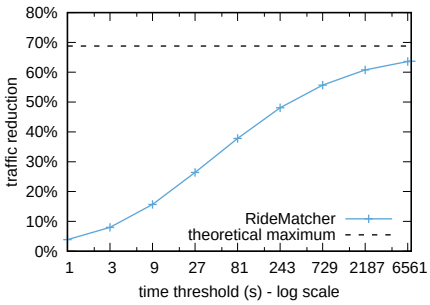

Fig. 7. Traffic reduction for different time thresholds

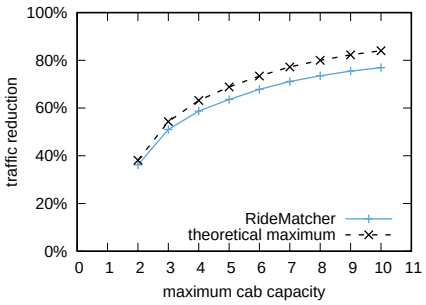

Fig. 8. Traffic reduction for different cab capacities

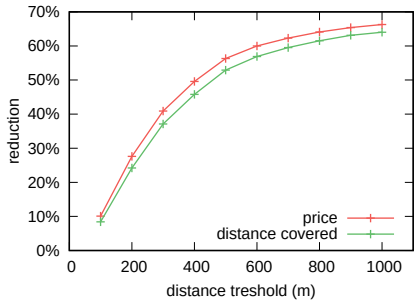

Fig. 9. Cost and mileage reduction for different distance thresholds

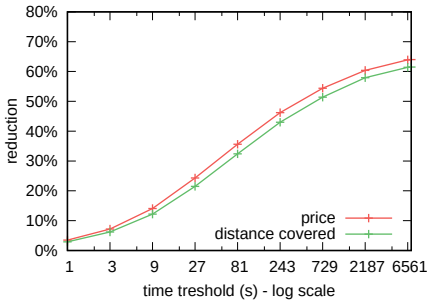

Fig. 10. Cost and mileage reduction for different time thresholds

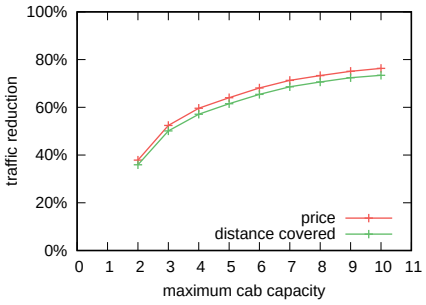

Fig. 11. Cost and mileage reduction for different cab capacities

indicate a lower carbon footprint. The maximum mileage reduction from Fig.11 corresponds to a decrease in the distance traveled by cabs of 53,050 miles. The carbon footprint of a typical taxi cab traveling this distance is 9.22 metric tons of CO2e [20]. It would take a year and 10.9 acres of forest to neutralize this amount of greenhouse gas [21].

Another indicator of the effectiveness of RideMatcher is the cab occupancy levels before and after ridesharing. These levels are depicted in Fig. 12. For this experiment we consider that a cab can carry at most 5 passengers. We can see from the graph that our system is able to reduce the number of cabs with a single passenger by 95% and the number of cabs with two passengers by 76%. With ridesharing, these passengers are re-allocated mostly to cabs with 4 and 5 passengers. The number of fully occupied cabs increases by 77% as a result of ridesharing. There is still a small fraction of rides that are left with one passenger after matching. These are most likely rides with uncommon routes, therefore matches could not be found for them.

We are not only interested in the amount of traffic reduction, but also in the quality of the ridesharing groups. Therefore, in Fig. 13 we study the distribution of group weights at the end of the simulation. As mentioned in Section III, the weight of a ridesharing group indicates the similarity between its members and can take values in the interval [0,1]. Consequently, in this experiment we divide this interval in 10 subintervals and count the number of weights that fall into each. We observe here that the weights are fairly balanced between 0.1 and 0.9, and drop at 1. We notice that most of the weights have values between 0 and 0.2 (28.1%), followed by 22.2% with values between 0.6 and 0.8. These "peak" intervals can be explained by the fact that nodes in our system are selfish, meaning that they try to find better ridesharing groups for themselves, while disregarding others. Because of this strategy, the weights of the groups tend to shift towards the extremes, which is visible in the results. Only 8.6% of the weights have a median value between 0.4 and 0.5.

B. Performance

Since our system is decentralized, its performance is closely related to the speed at which information is disseminated through the peer-to-peer network. We examine this speed by measuring how many rounds a node needs to discover rides in the network. As referred in Section V-A, RideMatcher works in rounds that execute at a certain frequency in real-world scenarios. In the case of RideMatcher, this frequency is largely dependent on the communication latency of the infrastructure. Since our system is designed to use the fog as communication medium, we can assume that the network latency is on the order of tens of milliseconds, as fog resources are placed closer to the mobile network. Therefore, in the remaining paragraphs we estimate pessimistically that each round takes around 100 ms.

Fig. 14 displays the average number of rides discovered by a node over the course of a simulation. The number of discovered rides is represented here as a fraction of the total number of rides (34,837). From the graph we can tell that it takes about 2000 rounds for a node to discover 20% of

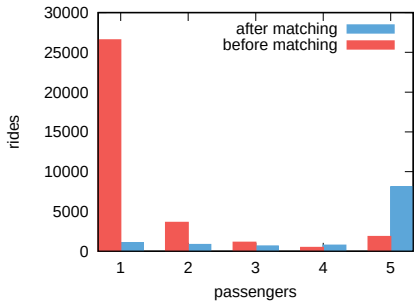

Fig. 12. Cabs occupancy before and after matching

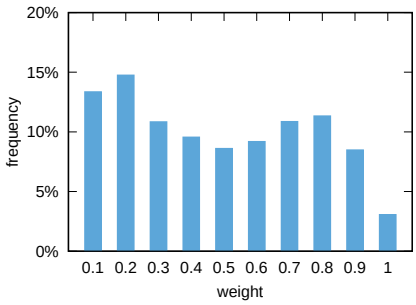

Fig. 13. Weights distribution

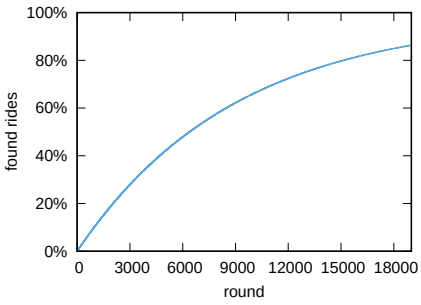

Fig. 14. Gossiping performance

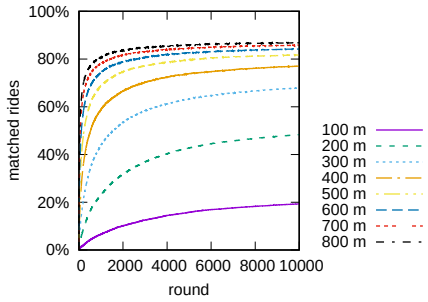

Fig. 15. Matching rate for different distance thresholds

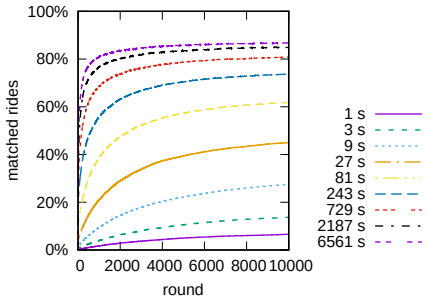

Fig. 16. Matching rate for different time thresholds

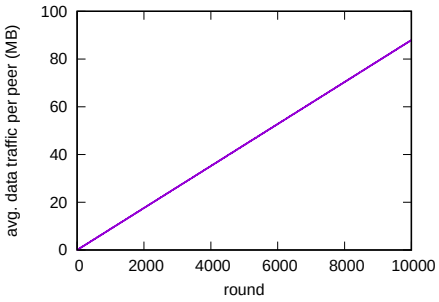

Fig. 17. Network traffic

the rides in the network, 6300 rounds to discover half of the network and 19,000 rounds to discover 86% of the network. If we assume that a round lasts 100 ms, then each 1000 rounds would add 1.6 minutes to the discovery time.

Next, we investigate how long it takes for participants in our system to find matching rides. We do this by looking at the rate at which rides get matched with each other through the simulation. As the probability of finding a match depends largely on the matching parameters, we study the dependency between the matching rate and the two parameters used: distance threshold D_T and time threshold T_T. In Fig. 15, the variance of the matching rate is shown for different values of D_T. Here, the vertical axis represents the percentage of nodes that are part of a ridesharing group. We can see that this percentage grows rapidly for lower values of D_T and converges to a value slight above 80%. If we consider a distance threshold of 800 m, then 862 rounds would be enough to find matches for 80% of the nodes. The same pattern can be noticed if we look at Fig. 16, which depicts the variance of the matching rate for different values of T_T. To better represent the results, the values we pick for T_T are on a logarithmic scale. Like in the previous experiment, the matching rate grows fast with the increase of T_T. If we consider a real-time scenario in which someone looks for a ride that differs by at most 2 minutes from its own ($T_T = 120s$), the chances to find a ride are 52.2% after 5 minutes of waiting. If we increase the time threshold to 5 minutes, then the chances raise to 66.8%. These numbers are based on the assumption that a round lasts 100 ms. If real-time matching is not required, the matching rate grows up to 87.2%.

Finally, we measure the network traffic generated by RideMatcher. Fig. 17 shows the average network traffic for a single node during a simulation. Each point in the graph represents the total network traffic generated by a node up to that moment in time. We observe that the traffic increases linearly throughout the simulation, indicating that each node generates a fixed amount of traffic each round, which is around 9 kB. This is due to the fact that most of the traffic in RideMatcher is generated by the gossiping protocol, which transfers a fixed amount of data in each round.

VII. 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 [22] 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 [23]. The problem of reducing the cost of sharing rides is further elaborated in [24], which proposes a greedy randomized adaptive search procedure (GRAPS) to reduce the cost of the trips. Huang *et. al.* [25] 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 [26] 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 [15], increasing matching stability at the expense of matching optimality [27], or preserving the user's privacy by taking an infrastructure-less approach [28]. A general method for assessing the benefits of vehicle pooling is presented in [29]. 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.

VIII. Conclusions

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 65%, freeing up roads from congestions. In addition, we show that RideMatcher reduces the distance traveled by cabs by 64% and the price of the rides by 66%. This makes the system 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.

Acknowledgements

This work is funded by the Dutch public-private research community COMMIT/. We thank Kees Verstoep for his comments that greatly improved the manuscript. We also thank Spyros Voulgaris for the great effort put into developing PeerSim and PeerMatcher.

References

[1] U.S. Department of transportation, Federal Highway Administration, https://ops.fhwa.dot.gov/freewaymgmt/faq.htm, accessed: 2017-10-20.
[2] Zimride, https://zimride.com/, accessed: 2017-10-20.
[3] BlaBlaCar, https://www.blablacar.com, accessed: 2017-10-20.
[4] Uber, https://www.uber.com, accessed: 2017-10-20.
[5] Lyft, https://www.lyft.com/, accessed: 2017-10-20.
[6] Via, https://ridewithvia.com/, accessed: 2017-10-20.
[7] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*. ACM, 2012, pp. 13–16.
[8] Bluetooth, https://en.wikipedia.org/wiki/Bluetooth, accessed: 2017-10-20.
[9] N. V. Bozdog, M. X. Makkes, A. Uta, R. B. Das, A. van Halteren, and H. Bal, "Sensele: Exploiting spatial locality in decentralized sensing environments," in *Proceedings of the 16th IEEE International Conference on Ubiquitous Computing and Communications*. IEEE, 2017.
[10] N. V. Bozdog, S. Voulgaris, H. Bal, and A. Van Halteren, "Peer matcher: Decentralized partnership formation," in *Self-Adaptive and Self-Organizing Systems (SASO), 2015 IEEE 9th International Conference on*. IEEE, 2015, pp. 31–40.
[11] M. Mallus, G. Colistra, L. Atzori, M. Murroni, and V. Pilloni, "Dynamic carpooling in urban areas: Design and experimentation with a multi-objective route matching algorith," *Sustainability*, vol. 9, no. 2, p. 254, 2017.
[12] A. 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.
[13] C. C. Robusto, "The cosine-haversine formula," *The American Mathematical Monthly*, vol. 64, no. 1, pp. 38–40, 1957.
[14] S. Voulgaris, D. Gavidia, and M. Van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, 2005.
[15] J. Alonso-Mora, S. Samaranayake, A. Wallar, E. Frazzoli, and D. Rus, "On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment," *Proceedings of the National Academy of Sciences*, p. 201611675, 2017.
[16] P. Karich and S. Schroder, "GraphHopper Directions API with Route Optimization (2017)," 2009.
[17] M. Haklay and P. Weber, "Openstreetmap: User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.
[18] Red, yellow, green: The science behind traffic lights, https://www.heraldextra.com/news/local/red-yellow-green-the-science-behind-traffic-lights/article_d488e07e-abbf-545e-b23e-8c22c223e6fd.html, accessed: 2017-10-20.
[19] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff, "A medium-scale distributed system for computer science research: Infrastructure for the long term," *Computer*, vol. 49, no. 5, pp. 54–63, 2016.
[20] Carbon Calculator, https://www.carbonfootprint.com/calculator.aspx, accessed: 2017-10-20.
[21] Greenhouse Gas Equivalencies Calculator, https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator, accessed: 2017-10-20.
[22] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*. IEEE, 2013, pp. 410–421.
[23] ——, "Real-time city-scale taxi ridesharing," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 7, pp. 1782–1795, 2015.
[24] D. O. Santos and E. C. Xavier, "Taxi and ride sharing: A dynamic dial-a-ride problem with money as an incentive," *Expert Systems with Applications*, vol. 42, no. 19, pp. 6728–6737, 2015.
[25] Y. Huang, F. Bastani, R. Jin, and X. S. Wang, "Large scale real-time ridesharing with service guarantee on road networks," *Proceedings of the VLDB Endowment*, vol. 7, no. 14, pp. 2017–2028, 2014.
[26] D. Zhang, T. He, Y. Liu, S. Lin, and J. A. Stankovic, "A carpooling recommendation system for taxicab services," *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 254–266, 2014.
[27] X. Wang, N. Agatz, and A. Erera, "Stable matching for dynamic ride-sharing systems," *Transportation Science*, 2017.
[28] S.-Y. Chiou and Y.-C. Chen, "A mobile, dynamic, and privacy-preserving matching system for car and taxi pools," *Mathematical Problems in Engineering*, vol. 2014, 2014.
[29] P. Santi, G. Resta, M. Szell, S. Sobolevsky, S. H. Strogatz, and C. Ratti, "Quantifying the benefits of vehicle pooling with shareability networks," *Proceedings of the National Academy of Sciences*, vol. 111, no. 37, pp. 13 290–13 294, 2014.