2016 IEEE First International Conference on Internet-of-Things Design and Implementation

VShare: A Wireless Social Network Aided Vehicle Sharing System Using Hierarchical Cloud Architecture

Yuhua Lin and Haiying Shen
Department of Electrical and Computer Engineering
Clemson University
Clemson, South Carolina 29634
{*yuhual, shenh*}*@clemson.edu*

***Abstract*—Carpool commuting enables multiple individual travelers with similar schedules and itineraries to share a common vehicle during a trip, and travelers can split travel costs in gas, parking and tolls with each other. It emerges as an effective way to solve traffic congestion, parking space tension and air pollution resulting from vehicle emissions. One of the challenges that restrict widespread adoption of carpool commuting lies in matching carpoolers. Existing carpooler matching methods include building carpool lanes in main airports and bus stops, using centralized servers to identify carpoolers based on historical travel data or real time travel requests. However, these methods cannot be applied to large scale adoption or incur long matching latency. To overcome drawbacks of existing methods, we propose *VShare*, a dynamic carpool system that leverages the wireless social network characteristic and hierarchical cloud server architecture. *VShare* incorporates two design components: matching through the wireless social network and a hierarchical cloud server structure. Upon receiving a user travel request, *VShare* first identifies possible carpoolers from neighbors in nearby locations, which reduces latency of sending travel requests to remote servers. If no carpool is found within nearby locations, a hierarchical cloud server architecture is used to match the travel requests. We have implemented the design of *VShare* and conducted trace-driven experiments. Experimental results show the effectiveness of *VShare* in substantially reducing matching latency while providing high success rate in matching carpollers.**

***Keywords*-Wireless social networks; Car sharing; Carpool commuting; Cloud servers**

I. Introduction

With rapid development of automobile industry, traffic congestion and air pollution resulting from automobile exhaust now become two of the greatest challenges in increasingly crowded urban areas all over the world. Carpool commuting (also known as carsharing and ridesharing), enables multiple passengers to share a single vehicle, which saves travelers' costs in fuel, parking and tolls and becomes an effective way to mitigate traffic congestion and pressure in parking spaces. Recently, companies like Uber [1] and Lyft [2] offer cheap peer-to-peer taxi services, in which a driver shares rides with other passengers mainly for the purpose of earning taxi fare. Carpooling is getting popularity and it represents 10% of all commute trips in the United States in 2009 [3]. However, instead of sharing vehicles with nearby travelers who are heading to the same destination, the majority (over 60%) of carpoolers commute with family members [4]. Various policies can be enforced by the government to encourage travelers to participate in carpooling, such as reducing the toll fares and parking fees for carpoolers and deploying high occupancy vehicle (HOV) lanes. For example, a vehicle carrying at least one passenger is able to gain access to HOV lanes or reduced tolls, which can motivate travelers with similar itineraries to share a common vehicle. Besides prompting carpooling by endowing travels with particular benefits, we also need mediums where travelers are easy to identify potential carpoolers who have similar travel schedules.

One of the most straightforward ways to match carpoolers is building carpool lanes in airports and bus stops [5]–[7], where passengers meet each other without specific prior arrangements and share taxis based on mutual agreement. For example, a number of carpool lanes are built at designated locations in the Washington DC area and East Bay of San Francisco [5], [6], where travelers wait in queues and make up informal carpools spontaneously on a first-come-first-service basis. Applications are also developed to facilitate carpool matching at designated locations. Bandwagon [7] is an practical application of rideshare car service, which helps travelers in long taxi lines at LaGuardia Airport to share taxis. However, matching travelers in carpool lanes at designated locations is defective as travelers identify carpoolers only when they are waiting in taxi lines, and the locations need to have a large volume of travelers in order to provide acceptable matching rate. Travelers are more likely to find carpoolers when they can make arrangements beforehand, e.g., they can schedule carpooling before coming to the airport. Also, this strategy only works in small-scale user population in a number of designated locations.

In large urban areas where thousands of vehicles running hundreds of thousands of trips per day, building scattered carpool lanes is not sufficient to greatly boost the possibility of carpooling. The optimization of carpool assignments with different departure and destination locations of travelers is challenging due to the large scale of the participants.

978-1-4673-9948-7/16 $31.00 © 2016 IEEE
DOI 10.1109/IoTDI.2015.30

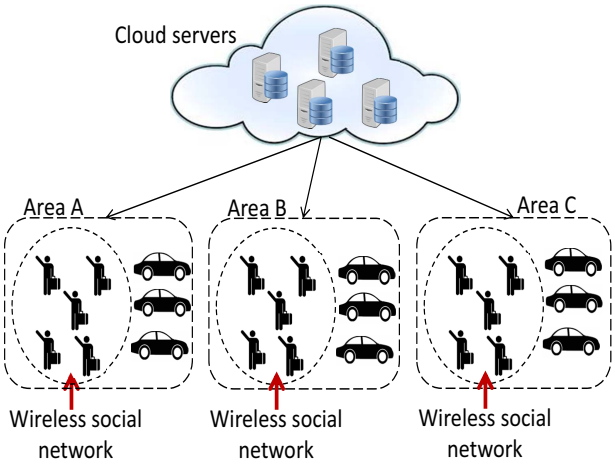


Figure 1: The architecture of the VShare carpooling system.

The increasing ubiquity of mobile devices provide potential solutions to this problem, in which prior user mobility knowledge is utilized to make potential carpools [8]–[10]. In this theme, portable devices such as smartphones and tablets are used to collect individual trips automatically without any explicit user involvement. Travel routes and mobility models are generated for each user, that is, we can infer the itineraries and travel schedules based on past observations. Potential carpoolers can be identified by using these travel routes and mobility models. However, this carpooler matching theme works in a static manner and can not be adaptive to real time scenario where users upload their travel requests on demand. Also, it cannot be used for a person's travel (e.g., business trip, unplanned shopping) that is not in his/her routine mobility.

To match carpoolers with real-time travel requests, some works propose using a centralized server to gather all real-time travel requests sent from users, and then calculate carpooling schedules that reduce users' travel costs and at the same time minimize total travel time [11]–[15]. In these dynamic carpooling systems, both riders and drivers provide preferred travel information including desired departure time window, location and maximum travel distance. A number of predefined objectives in determining carpool matches are also specified by the systems, such as minimizing total travel time and trip costs. The carpool matching problems is typically formed into a linear programming problem, where objectives are optimized subject to a set of constraints. The problem is then solved and carpool assignments are returned to the travelers.

Using a centralized server to match carpoolers requires long computation latency. Also, it takes long transmission latency since user travel requests are sent from portable devices to the remote server. To reduce the matching latency, we propose *VShare*, a dynamic carpool system that leverages the wireless social network characteristic and hierarchical cloud server architecture. Figure 1 shows the architecture of the *VShare* carpooling system, which is formed by utilizing the wireless social network and the cloud servers. When a user sends out a travel request, *VShare* aims to identify carpoolers through the wireless social network in the first step. It broadcasts this request to a number of the requester's neighbors in his/her neighborhood. A neighbor heading to the same destination will respond to the request with a matching score towards the request that measures the similarity of their itineraries and travel times. After receiving a number of responses from the requester's neighbors, a carpool of multiple passengers is formed and all carpoolers are notified of the travel schedule. If no carpoolers are identified within a neighborhood, *VShare* then sends this request to cloud servers. As the cloud has proved to be an effective platform to host a variety of applications [16]–[18], it is used to store and match travel requests. In *VShare*, a hierarchical cloud server architecture is used to store all travel requests with the same departure location and destination in the same server. The server responsible for the target request will then identify potential carpoolers by calculating the matching scores between existing requests and target request. Finally, this server then returns the schedule of carpooling to all passengers involved when matches are found. In *VShare*, the carpool matching latency is substantially shortened by first looking for carpoolers within nearby neighbors. Also, the hierarchical cloud server architecture is effective in matching travel requests with short latency. *VShare* can be used for different vehicles including taxis, personal vehicles, rented vehicles.

We summarize the contribution of this paper as follows:

- We have proposed *VShare*, a wireless social network aided vehicle sharing system using hierarchical cloud architecture. In *VShare*, carpoolers are matched by broadcasting messages to neighbors and using a hierarchical cloud server architecture.
- We have designed algorithms to calculate a matching score between two travel requests that measures the similarity of their itineraries and travel times, we then identify potential carpoolers by refering to the mathcing scores.
- We have conducted trace-driven experiments by using a PlanetLab node to simulate cloud servers. We have presented the experimental results in a number of performance metrics, which show the effectiveness of *VShare* in matching carpoolers with short latency.

The remainder of the paper is organized as follows. Section II presents an overview on the related work. Section III-A describes the detailed design of *VShare* with description of each step. Section IV presents the performance evaluation. Section V concludes this paper with remarks on our future work.

II. Related work

Carpooling aims to match travelers with similar itineraries and traveling schedules, and it provides both economical and environmental benefits by increasing the number of passenger on taxis [19]–[25]. Existing works on matching carpoolers can be broadly grouped into three categories.

The first type of carpooling systems are matching at designated locations, where drivers and a larger number of travelers line up in queues and establish carpoolings to save taxi and toll fares [5]–[7], [26]. For example, the Washington DC area and East Bay of San Francisco have developed a number of carpool lanes at designated locations [5], [6]. Applications such as Bandwagon have been developed to enable carpool commuting at major airports [7], campuses [27] and large companies [28]. The main disadvantage of spontaneous matching is that it does not allow to make carpool arrangements beforehand. Also, this strategy only works in a number of designated locations where high density of travelers are waiting in lines, which limits widespread adoption of this carpooling systems.

The second type of carpooling systems are static matching, where itineraries, routes and travel schedules are collected based on historical records [29]–[31]. These systems take advantage of the fact that a user's travel routes and locations can be predicted based on some mobility context such as time and positions [32]. MobiCrowd [8] leverages smartphones to collect trip information for users without any explicit effort from them. This scheme generates daily trips and mobility models for each user, and then makes carpooling schedules using these mobility models. Naoum *et al.* [33] proposed a stochastic mixed integer programming model to optimize the carpool assignment of employees in large organizations such as companies and government offices. In this model, employees' home locations are denoted by vertices on the graph, which are then clustered based on proximity from each vertex to a specific company. All vertices in the same cluster share a common vehicle. Some works applied optimization models to identify the most suitable locations to build carsharing stations [34], [35], with the objective of improving system performance like the average number of rides per day.

The third type of carpooling systems are dynamic matching, where carpooling schedules are made based on real-time user requests. Ma *et al.* [11] used a spatio-temporal index to retrieve candidate taxis that matches a user's travel schedule; they then selected a taxi with minimum additional incurred travel distance if this user shares the taxi with existing passengers. Huang *et al.* [36] proposed kinetic tree algorithms which can efficiently schedule dynamic travel requests and adjust travel routes on demand. Zhang *et al.* [22] proposed a carpool service system named coRide, which aims to reduce total travel mileage for less gas consumption in a large-scale taxicab network. coRide uses

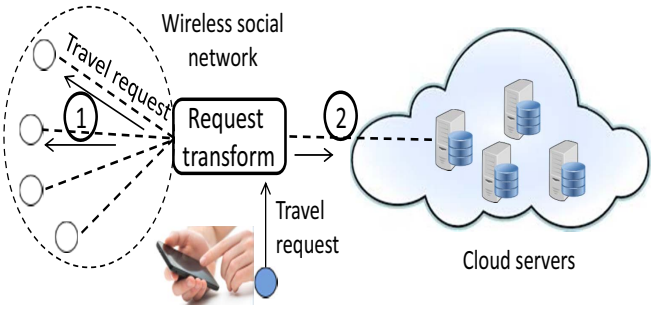


Figure 2: An overview of matching carpoolers in the VShare carpooling system.

the linear programming to solve the route selection problem under different practical constraints. As users call for taxis on demand and this dynamic ride-sharing problem is a NP-hard optimization problem [37]. Some works [38]–[41] propose heuristic approaches to solve the car sharing and route planning problem with reduced computational complexity. BlueNet [17], [42] is a cloud-based carpool matching module, which uses a genetic algorithm to accurately find the optimum carpool schedule and matching results by simulating natural evolution. Simulation methodologies [43]–[46] are also used to assist the decision makers in selecting optimal carpool scheduling strategies, with the objectives of maximizing the participants' satisfaction level and minimizing the number of vehicles used. This strategy requires long computation latency as all user travel requests are processed by centralized servers which may be far away from the travelers.

To reduce the matching latency, in *VShare*, when a user sends out a travel request, this request will be broadcasted to a number of neighbors in nearby locations. A neighbor heading to the same destination will respond to the request and a carpool is formed. If no carpool is identified within neighborhood, this request is then sent to a cloud. A hierarchical cloud server architecture is used to match the new request with existing requests, it then returns the schedule of carpooling when matches are found. The carpool matching latency is substantially shortened by first looking for carpoolers within nearby neighbors. Also, the hierarchical cloud server architecture is effective to match travel requests with short latency.

III. System design of VShare

Matching carpoolers is challenging as users' travel requests and positions are highly dynamic and difficult to predict. The goal of a ride sharing system is to increase the carpooler matching success rate and at the same time minimize the matching latency. *VShare* achieves this goal by leveraging the wireless social network and hierarchical cloud server architecture to help the matching of carpoolers.

We will describe the detailed design of each component in this section. Important notations used in this paper are listed in Table I. Note that we use "predefined" to indicate that the parameter is predefined by the vehicle sharing service provider.

Table I: Table of major notations.

p_i	passenger i
r_i	the travel request for passenger p_i
P	the list of passengers
R	the list of candidate travel requests
t_i	the travel time for passenger p_i
w_i	the maximum wait time for passenger p_i
m_{ij}	the matching score between r_i and r_j
C	a carpool
$t(C)$	travel time for carpool C
S	vehicle capacity, i.e., number of passenger seats
ϕ	predefined max # of neighbors to forward a request
$\tilde{T}$	predefined latency threshold of matching with neighbors

A. Overview of VShare

Our proposed *VShare* carpooling system incorporates two-step operations: matching via the wireless social network and matching using a hierarchical cloud server architecture. Figure 2 shows an overview of the *VShare* carpooling system. When a user sends out a travel request from its mobile app, this request will be transformed into a request numerical string.

To reduce the matching latency, in the first step, *VShare* tries to match carpoolers within nearby locations instead of forwarding the request to the remote cloud. The travel request is broadcasted to a number of neighbors in nearby locations. Each neighbor user's mobile app receiving the request will then check its travel schedule. If its user is heading to the same destination at a close time, it will respond to the request. Based on the responses, a carpool is formed. The detailed description of the first step is presented in Section III-D.

If no carpoolers are returned, in the second step, this request numerical string will be sent to the cloud. Through a hierarchical cloud server architecture, this request will be forwarded to the host server where travel requests with the same departure location and destination location are stored. This server will match the departure time and maximum wait time between the new request and existing ones, and then returns the matching carpooler results to all users in the carpool. The detailed description of the second step is presented in Section III-E.

Before we present the matching process, we first introduce how to transform a request to a string for easy matching operations in Section III-B. Then, we introduce how to find carpoolers for a given travel request in Section III-C2.

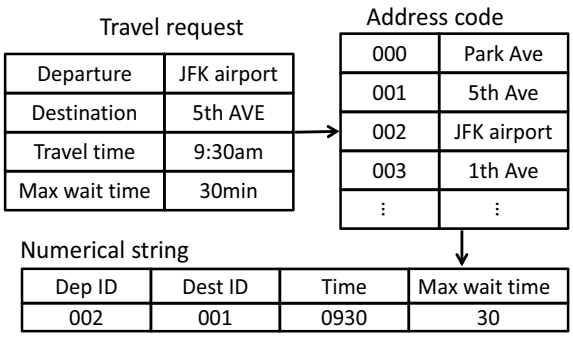


Figure 3: The process of the travel request transformation.

B. Transformation of Travel Requests

In the *VShare* carpooling system, it is critical to match potential carpoolers who have similar itineraries and travel schedules. For this purpose, *VShare* needs to compare the similarity of the travel requests. For the ease of the comparison, in this section, we introduce how to transform a user travel request to a numerical string. Compared to text representation, numerical strings are more effective in fast storage and information retrieval and quick comparison. Each user travel request is first represented by a number of attribute values, which indicate the travel schedule. The travel request can be entered by a user through the *VShare* mobile phone app, where each request is represented by four attributes, i.e., departure location, destination, travel time and maximum wait time. For example, a user is traveling from the John F. Kennedy (JFK) airport to the 5th AVE at 9:30am and his/her maximum wait time is 30 minutes. This travel request is then presented with four attribute values "JFK airport; 5th AVE; 9:30am; 30min". Note that these attributes are determined by users' travel preferences. Actually, the attributes used in the system can be flexible based on the requirements from carpool service providers.

The carpool service provider defines a table of address codes that use a number of digits to represent each address. An address code is generated in a similar way as using a zipcode to represent an area. An example of such an address code table is shown on top right of Figure 3. Each mobile phone installed with the *VShare* app stores the address code table. We then transform each user request into a numerical string. As the attributes of departure location and destination are important in travel requests, we set them as primary keys and transform them into IDs. We use *Dep ID* and *Dest ID* as abbreviations for the departure ID and destination ID. The mapping process is conducted by applying the address code. For instance, location "JFK airport" is denoted by code "002" (i.e., a number of digits) and "5th AVE" is denoted by "001". Thus, the *Dep ID* and *Dest ID* in our example are "002" and "001", respectively. We also represent the travel time and maximum wait time as numerical values.

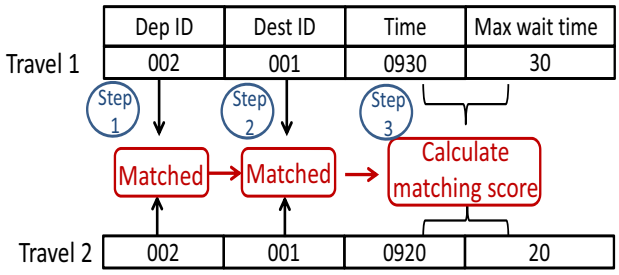


Figure 4: Process of matching two travel requests.

For example, "9:30am" is represented by "930" and "30min" is represented by "30". We then combine the digits of all attribute values together and get an n-digit numerical string.

Figure 3 shows the process of transforming a travel request into a numerical string. The top left row lists all travel schedule attributes in the system. When a user sends out a travel request with attribute values "JFK airport; 5th AVE; 9:30am; 30min". After we infer the address codes for *Dep ID* and *Dest ID*, the travel request is transformed to a numerical string "00200193030". This numerical string is used to match users who have similar travel schedules and are likely to share the same vehicle.

C. Matching of Potential Carpoolers

In this section, we first discuss how to match two travel requests by comparing the corresponding numerical strings of these travel requests, which can help identify two potential carpoolers. We then extend the method to form carpools with more than two passengers.

1) Matching of Two Travel Requests: In *VShare*, we define a matching score to measure the likelihood that passenger p_i is able to share vehicle with passenger p_j. The matching score is measured by the similarity between these two passenger's travel numerical strings r_i and r_j. When *VShare* identifies potential carpoolers for a given request via using the wireless social network or the cloud, it needs to calculate the matching score between two passengers. Figure 4 shows the process of matching two travel requests, which includes 3 step operations. We first compare the values of *Dep ID* and *Dest ID* sequentially in the first two steps. If two travel requests have different *Dep ID* or *Dest ID*, the process stops and these two requests are not matched. Otherwise, we continue to examine the values of *Travel time* and *Maximum wait time* (denoted by t and w, respectively).

Given two travel requests r_i and r_j that are initiated by passengers p_i and p_j, r_j has an earlier travel time than r_i (i.e., $t_i > t_j$). We use $(t_i - t_j)$ to calculate the travel time gap between r_i and r_j in minutes. When $(t_i - t_j) > w_j$, i.e., p_j's waiting time is not long enough to share vehicle with p_i, p_j will not respond to the p_i. When $(t_i - t_j) < w_j$, i.e., the travel time gap is less than the maximum wait time of p_j and p_j is able to share vehicle with p_i, we calculate the matching score m_{ij} between them by:

$$m_{ij} = 1 - (t_i - t_j)/w_j, \quad (1)$$

m_{ij} represents the degree of satisfaction in waiting for possible carpoolers. A large value of m_{ij} means that one traveler does not need to wait long for another traveler to share a common vehicle.

Algorithm 1 Matching of two travel requests.

1: **Input:** travel numerical strings of r_i and r_j;
2: **Output:** matching result;
3: **if** *Dep ID*(r_i)≠*Dep ID*(r_i) **then** //Departure locations are unmatched
4: **return false**
5: **end if**
6: **if** *Dest ID*(r_i)≠*Dest ID*(r_i) **then** //Destinations are unmatched
7: **return false**
8: **end if**
9: **if** $(t_i - t_j) > w_j$ **then** //Travel time gap exceeds maximum wait time
10: **return false**
11: **end if**
12: calculate m_{ij} by $m_{ij} = 1 - (t_i - t_j)/w_j$
13: **return** m_{ij}

Algorithm 1 shows the pseudocode of matching two travel requests. The algorithm first matches these requests by their departure locations (Lines 3-5). It then matches requests by traveling destinations (Lines 6-8). It continues to check if the travel time gap between two requests is less than the maximum wait time (Lines 9-11). It finally calculates the matching score based on travel time and maximum wait time and returns the matching result (Lines 12-13). Two potential carpoolers are identified when $m_{ij} > 0$, and this matching result will be return to both p_i and p_j.

2) Matching of Multiple Travel Requests: Given travel request r_i, and a list of candidate travel requests $R = (r_1, r_2, ...r_u)$ initiated by a list of passengers $P = (p_1, p_2, ..p_u)$. All requests in R are order by their matching scores towards r_i in descending order. When p_i carpools with p_1, these two passengers can travel earlier compared to carpoolers of p_i and p_2. Our strategy to form a carpool (denoted by C) is adding one candidate from R at a time and check if all passengers in the carpool are satisfied with the carpool travel schedule, i.e., the wait time of each passenger is within its maximum wait time. If the carpool's travel time is beyond a passenger's maximum wait time, we then remove the newly added passenger and return the current carpool schedule.

When adding a travel request to the carpool, we select requests from R in order. Assume that carpool C is built when we add r_j to the carpool. We use $t(C)$ to denote carpool C's travel time, which is the latest travel time of all passengers in C, i.e., $t(C) = \max(t_1, t_2, ..t_j)$. For each travel request r_k in C, we sequentially calculate $t(C) - t_k$ as wait time of p_k in minutes. C is invalid if $t(C) - t_k > w_k$, i.e., p_k does not agree with the travel time of carpool C.

In this case, we remove r_j from C and return current carpool schedule to all passenger involved. Otherwise, we continue this process until the number of passengers in current carpool reaches the capacity S of the vehicle. S is determined as the number of passenger seats on a vehicle.

Algorithm 2 Matching of multiple travel requests.

```
1: Input: travel request r_i and candidate list R = (r_1, r_2, ...r_u);
2: Output: carpool C;
3: for r_j ∈ R do
4:     if size of C equals S then  //Vehicle capacity is reached
5:         return C
6:     end if
7:     add r_j to carpool C   //Increase the size of the carpool
8:     calculate C's travel time t(C) = max(t_1, t_2, ..t_j)
9:     for r_k ∈ C do
10:        if t(C) − t_k is greater than w_k then
11:            remove r_j from carpool C
12:            return C
13:        end if
14:    end for
15: end for
```

Algorithm 2 shows the pseudocode of matching multiple travel requests. When we input a target travel request and a list of candidate requests R, Algorithm 2 aims to identify multiple carpoolers from R to share a vehicle with the requester. The algorithm first examines if the number of passengers in current carpool reaches the capacity of a vehicle (Lines 4-6). When $|C_j| = S$, the algorithm returns the current carpool schedule. Otherwise, it adds r_j to the carpool and calculates the travel time of this carpool (Lines 7-8). For each travel request in the new carpool, the algorithm continues to check if the carpool's travel time satisfies all passengers' maximum wait time (Lines 9-14). If one of the passenger does not agree with the carpool's travel time, the algorithm removes r_j from the carpool and returns the result.

D. Matching Via the Wireless Social Network

A passenger p_i's wireless social network consists of users in p_i's neighborhood in the same area. We consider a passenger's wireless social network first in forming a carpool for the passenger because these users tend to have the same departure location and similar travel time. The local carpooler matching rather than the global matching can expedite the speed for finding carpoolers for the passenger.

When a user sends out a travel request, he/she is required to enter travel information, this information is then broadcasted to a number of neighbors in the area. *VShare* defines a time-to-live (TTL) value, which is the maximal number of hops that a travel request can be forwarded to. In our paper, we set TTL to 2 hops for all travel requests. *VShare* defines a constant ϕ, which is the largest number of neighbors to forward a user travel request in each hop. After a user p_i submits a travel request, the *VShare* app will transform the request into a numerical string r_i and forward r_i to ϕ random

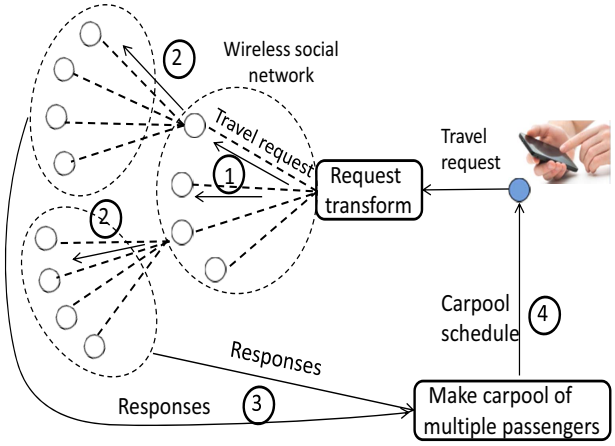


Figure 5: Demonstration of matching via the wireless social network.

neighbors. A neighbor p_j receiving this request will apply Algorithm 1 to check if its travel request is matched with p_i's. If the matching score m_{ij} is greater than 0, p_j replies to p_i with the value of m_{ij}. Also, p_j will forward the travel request to ϕ neighbors in its own neighbor list and reduce the travel request's TTL by 1. The request is forwarded along wireless social links until TTL=0.

From this process, we can see that a problem needs to be resolved is how to determine carpoolers for p_i when it receives a number of replies from its neighbors. p_i's *VShare* app first puts all received responses from p_i's neighbors in a list $R = (r_1, r_2, ...r_u)$ so that elements in R are ordered in decreasing order of their matching scores with r_i. The app then uses Algorithm 2 to make a carpool of multiple passengers. When a number of passengers are identified as potential carpoolers for p_i, the app then starts an instant conversation with these passengers to confirm the plan of sharing vehicles. Figure 5 demonstrates the steps of matching via the wireless social network with TTL=2. *VShare* aims to identify carpoolers through the wireless social network in step 1 and step 2. After receiving all responses from neighbors in step 3, *VShare* makes a carpool of multiple passengers among all received responses and notifies all passengers in step 4.

If the travel requester p_i has not received a response after a certain latency threshold $\tilde{T}$ that is specified corresponding to TTL (e.g., 5 minutes), this travel request is forwarded to the cloud, where the request is matched with existing requests by the hierarchical cloud.

In the wireless social network, neighbors may not be acquainted with each other in real life, so they may have concern about their safety and privacy when sharing a vehicle with strangers. This issue will prevent users from actively participating in the vehicle sharing activities. In order to provide trust among ridesharing travelers, we may

integrate *VShare* with some existing social networks (e.g., Facebook and LinkedIn), so that participants can acquire more personal information of their travel partners. However, this issue is beyond the scope of this paper and it is our assumption that users are honest and reliable when using the *VShare* system.

E. Matching Via Hierarchical Cloud Architecture

If no carpool is identified within a requester's wireless social network, this request is then sent to cloud servers, where carpoolers are matched. In a cloud data center, there are a larger number of servers and each server stores and processes a large amount of user requests. A fundamental challenge in the cloud datercenter is to efficiently identify potential carpoolers in the complex environment characterized by distributed requesters and large scale data volume. In such an environment, thousands or even millions of user travel requests are scattered across distributed servers inside cloud datacenters. Thus, matching carpoolers requires the communication between different servers when they need to calculate matching scores between travel requests. In order to match travel requests efficiently, we need to reduce the number of communications between different servers. To achieve this goal, we organize the cloud servers in a hierarchical structure.

Figure 6 shows the three-level hierarchical cloud server architecture. Specifically, the cloud servers are formed into a three-level hierarchy from the top to the bottom: a centralized server (CServer), departure managers (DepM) and destination managers (DesM). The first level is a centralized server and it is responsible for distributing travel requests to different departure managers in the second level based on the departure IDs of the requests. Each departure manager is responsible for travel requests with a specific departure ID and distributes the requests to different destination managers in the bottom level based on the destination IDs of the requests. Each destination manager receives, handles and stores user travel requests with the same departure ID and destination ID. For this purpose, the centralized server maintains an index of departure IDs and their responsible departure managers, and each departure manager maintains an index of destination IDs and their responsible destination managers.

We then describe the function of each level by using Figure 6 as an example. All requests received by the cloud are assigned to the centralized server. Upon receiving a travel request, the centralized server passes this request to the departure manager that is responsible for the request's departure ID. The departure manager then forwards the request to the destination manager that is responsible for the request's destination ID. From this process, we can see that the travel requests with the same departure ID and destination ID will go to the same destination manager. To find potential carpoolers for a newly received request, each

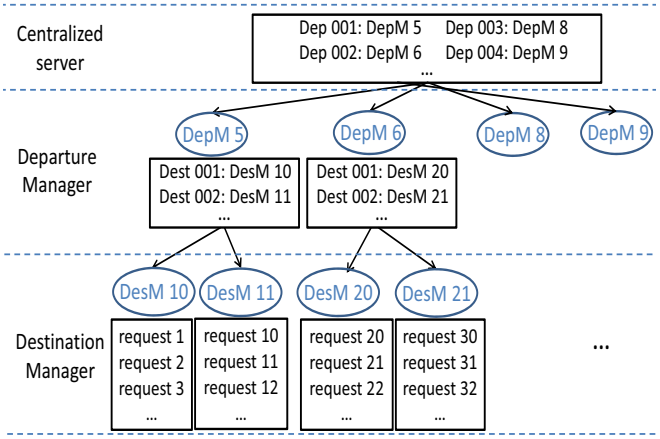


Figure 6: An example of three-level hierarchical cloud server structure.

destination manager only needs to match the travel time and maximum wait time of the request with the requests stored in itself. If no potential carpoolers are found, it stores the new request in order to match it with subsequently received requests. After a destination manager stores a request for a time period, it notifies the requester that carpoolers cannot be found. This time period can be specified by the users or the system based on the departure time of users. Thus, *VShare* does not need to involve multiple servers when matching potential carpoolers, which reduces the matching latency. The servers in the second and third levels are independently scalable, i.e., the carpool service provider can adjust the number of servers dynamically according to the data size of user requests.

For example, after the centralized server receives a travel request r_i represented by numerical string "00200193030", it dispatches r_i to DepM 6 which is responsible for requests with departure ID "002". DepM 6 then forwards r_i to DesM 20, which is responsible for travel requests with both departure ID "002" destination ID "001". DesM 20 has all requests with the same departure ID "002" and destination ID "001". Inside each destination manager, all travel requests are ordered sequentially by their travel times. The destination manager will then selects a number of u existing travel requests that have the closest travel time to that of r_i, and puts them in a list $R = (r_1, r_2, ...r_u)$. All travel requests in R are ordered in decreasing order of their closeness of travel time with r_i. Next, the destination manager matches carpoolers for travel request r_i using Algorithm 2 described in Section III-C2.

After a number of carpoolers are identified for r_i, if none of the carpoolers drive a car, the destination manager randomly assigns an available vehicle (like taxi) currently in the departure location to these carpoolers and sends the travel schedule to them. The *VShare* app will start an instant conversation for these carpoolers to confirm the plan of sharing vehicles.

IV. Performance Evaluation

We conducted trace-driven experiments using the Cab mobility trace dataset [47].

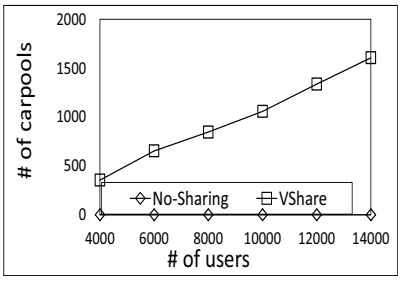
A. Experimental Settings

The Cab mobility trace contains mobility traces of taxi cabs in San Francisco, USA. It contains GPS coordinates of approximately 500 taxis collected over 30 days in the San Francisco Bay Area. We used the DBSCAN clustering algorithm [48] to identify 338 locations in this trace. We generated user requests from the Cab mobility dataset using the following process: 1) at a specific time t_i, if a cab turns from vacant to occupied in location A, we assume that this cab picks up one passenger in location A at time t_i; 2) at next time point t_j, this cab turns from occupied to vacant in location B, we assume that this passenger gets off in location B at time t_j; 3) when a cab is occupied, we assume that there is only one passenger in the cab. Thus, a user travel request was generated as "a passenger wants to travel from A to B at time t_i". We then assigned this travel request to a random node and set its maximum wait time randomly in [5,30] minutes. Before a user gets on a cab in location A, we assumed that this user has stayed in a location for 10 minutes, and users stay in the same location form a wireless social network. We used the number of cabs in the Cab mobility trace dataset as the default number of cabs, which is 536; and we used an approximate value of the average number of travel requests per day in this dataset as the default number of nodes, which is 14000. In order to vary the number of users, we also created additional users and randomly assign requests to these uses. We assumed that each user is a single traveler, i.e., originally, it does not travel with any other users. Each cab's capacity is 4, i.e., it accommodates at most 4 passengers at one time.

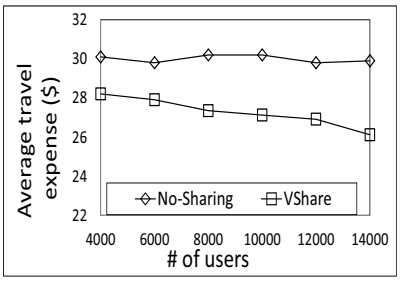
In our proposed *VShare* system, when a user sends out a travel request including its departure time, location and destination, this request is broadcasted to 10 neighbors in this location. We set TTL=2, i.e., each neighbor will forward this request to its neighbors before the forwarding procedure halts. A neighbor heading to the same destination at close departure time will respond to the request and a carpool is formed. If no responses are returned after 5 minutes (i.e., $\tilde{T}$ =5 minutes), this request is then sent to the central processor in the cloud, where the request is matched with all other requests and possible carpoolers are identified. The cloud then selects a vehicle in this area and allocates to the carpoolers. We compared *VShare* with *Cloud* and *No-Sharing*. In the *Cloud* system, all user travel requests are gathered and processed by the cloud. It uses a hierarchical cloud server architecture introduced in Section III-E to store travel requests and match potential carpoolers. We also implemented the *Cloud* system without using a hierarchical cloud server architecture (denoted by *Cloud-D*). In *Cloud-D*, a user request is stored in a random cloud server. To match potential carpoolers for a target travel request, each server first selects 4 travel requests from its storage that have the same departure ID, the same destination ID and closest travel time to the target request. It then sends them to a centralized server. The centralized server collects all similar travel requests and matches potential carpoolers for the target request. In the *No-Sharing* system, there is no carpool commuting, and each user occupies a single cab. Each user will wait in a queueing line of the location, when there are available cabs, these users will be picked up on a first-come-first-serve (FCFS) manner. If a user cannot get on a cab within its maximum wait time, it will leave the queue and we regard it a failure in catching a cab.

We used a PlanetLab [49] node to simulate cloud servers, which is with IP 128.112.139.43 in Princeton University. The experiment simulates a 7 day scenario. Each user takes one or two cabs per day with equal probability, we recorded the experimental results for each day and report the average values during the 7 day period. In our experiment, each user will get on an available cab if he/she cannot find any carpooler, otherwise, all carpoolers share one available cab. If there is only one passenger on the cab, he/she pays all travel fare; otherwise, the travel fare is evenly split between all passengers in the carpool. We are interested in the following metrics:

- **The number of carpools.** The number of cabs that carry more than one travelers at the same time. This metric indicates the effectiveness of the proposed system in matching users to carpools.
- **Average travel expense.** Users can find travelers who are willing to share rides with in carpool commuting. The travel expense is reduced as travelers split the travel fare and users are motivated to use the VShare system to find carpoolers.
- **Average matching latency.** It is the latency in identifying a user that can share a ride with another user, which is the time span from when a user submits a travel request until the time he/she receives the carpooler and cab information. This metric is used to show that using the wireless social network based matching can quickly identify a user that can share rides.
- **Success rate of catching a cab.** It is the probability that a user can catch a cab within its maximum wait time, which is calculated as the ratio of travel requests that can travel within its maximum wait time over all travel requests. By sharing cabs, more cabs will be available and users are easier to catch a cab.
- **The number of cabs needed.** Assume we have unlimited number of cabs and we need to transport all users within their maximum wait times, we record the minimum number of cabs needed to use. When users can share cabs with each other, a smaller number of cabs are needed to transport a specific number of users,

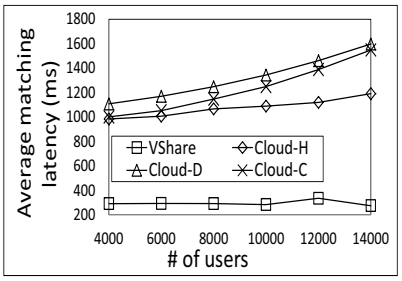


(a) Number of carpools.

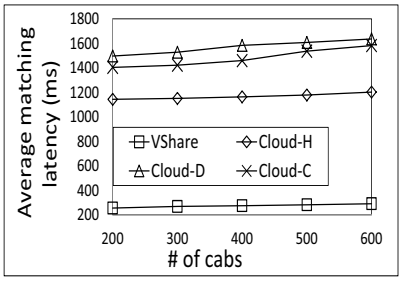


(b) Average travel expense.

Figure 7: Performance with different number of users.



(a) Performance with different number of users.



(b) Performance with different number of cabs.

Figure 8: Average matching latency.

which mitigates the traffic load in urban areas.

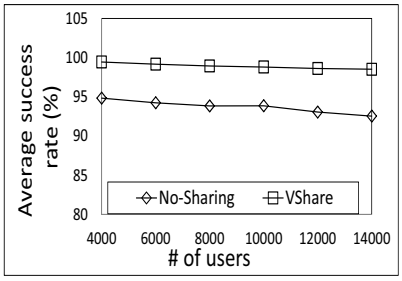
B. Experimental Results

We present the experimental results with respect to all metrics introduced above.

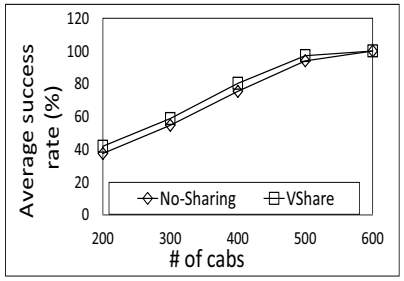
1) The Number of Carpools: If a user shares a cab with at least one passenger, we regarded it as a carpool, i.e., a cab carries more than one user. Figure 7(a) shows the number of carpools with different number of users in the system. We see that the number of carpools in *VShare* increases steadily when the user population gets larger. This is due to the reason that as user density becomes higher in a location, a user is easier to find carpoolers. In *No-Sharing*, each user occupies a single cab, so the number of carpools in this system remains 0. Figure 7(a) shows that *VShare* is effective in identifying potential carpoolers for users.

2) Average Travel Expense: One of the potential benefits of sharing vehicles for users is reducing the travel expense. In this experiment, we assume that the travel fare for a single trip costs f dollar regardless of the travel distance, and f was randomly selected in [20,40]. We also define a travel fare increment factor α, which is the ratio of increase in travel fare when the number of passenger is increased by one. We set the travel fare increment factor α defined to 0.1. Therefore, if there are n carpoolers sharing the same vehicle, the total travel fare that the driver receives from all passengers is $(1+\alpha)^{n-1}f$, and the travel fare is evenly split between all passengers in the cab. Figure 7(b) shows the average travel expense of all users with different number of users in the system. We see that compared to *No-Sharing*, the proposed *VShare* system is effective in saving users' travel fares as users can find carpoolers who can share payment to the taxi service. We also see that the average travel expense drops as the number of user increases in the system. This is due to the reason that users are more likely to potential carpoolers when user density is high. Figure 7(a) and Figure 7(b) show that *VShare* can effectively match travel requests and identify carpoolers, thus save travel expense for users.

3) Average Matching Latency: Next, we plot the average matching latency with different number of users and various number of cabs in Figure 8(a) and Figure 8(b), respectively. These metrics are important in that users are generally reluctant to wait long in the queue and they prefer to find carpoolers quickly. We see that the proposed *VShare* system generates substantially lower matching latency than the *Cloud* system. As *VShare* first matches a user's travel request with nearby users using the wireless social network, and a user is likely to identify carpoolers within a short latency. On the other hand, *Cloud* processes all user requests by using remote cloud servers, which incurs a relatively long transmission delay from the user mobile device to the cloud. Also, the matching results need to travel long distance back to the users. *Cloud-D* generates longer matching latency than *Cloud* due to the reason that the centralize server needs to collect similar travel requests from all servers before it matches potential carpoolers for the target request. On

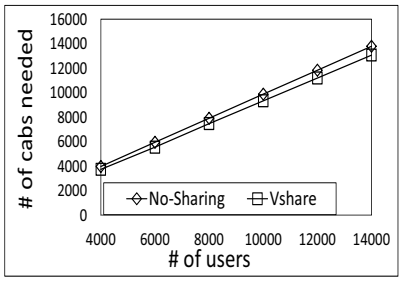


(a) Performance with different number of users.



(b) Performance with different number of cabs.

Figure 9: Success rate of catching a cab.



(a) Number of cabs needed.

Figure 10: Performance at different # of users.

the other hand, *Cloud* uses a hierarchical server structure to quickly locate the destination manager that stores all travel request with the same departure ID and destination ID as the target request, and this destination manager then matches potential carpoolers for the target request with short latency. By comparing the matching latency of *Cloud-D* and *Cloud*, we can see that the hierarchical cloud architecture is advantageous in storing and matching travel request efficiently. Figure 8(a) and Figure 8(b) show that *VShare* is able to provide satisfactory carpool matching service with relatively short latency.

4) Success Rate of Catching a Cab: Figure 9(a) shows the success rate of catching a cab with different number of users. We see that *VShare* produce higher success rate than *No-Sharing*. In the *No-Sharing* strategy, each cab is occupied by only one passenger, given a limited number of cabs and a large number of travelers during peak hours, some travelers may not ba able to catch a cab as all cabs are generally busy. While in *VShare*, multiple users heading to the same destination can identify each other and share one cab, thus, more passengers are transported with the same amount of cabs. As a result, users in *VShare* are easier to catch a cab when more empty vehicles are available to use. The success rate drops slightly as the number of users increase due to the reason that the capacities of cabs are fully utilized with larger number of users, so some users are not able to get an available cab.

Figure 9(b) shows the success rate of catching a cab with various number of cabs. We see that the success rate rises substantially when the number of cabs increase due to the reason that more passengers can be accommodated with larger number of cabs in service. We also see that *VShare* produce higher success rate than *No-Sharing* due to the same reason explained in Figure 9(a). That is to say, *VShare* aims to let multiple users heading to the same destination share rides with each other. From Figure 9(a) and Figure 9(b), we infer that *VShare* can effectively mitigate the tension in catching taxis during rush hours by enabling travelers to share rides with each other.

5) The Number of Cabs Needed: Figure 10(a) shows the number of cabs needed to transport different number of users. We see that more cabs are needed when the user population increases from 4,000 to 14,000 due to the same reason explained in Figure 9(a) and Figure 9(b). Also, given the same amount of users, *VShare* uses smaller number of cabs than *No-Sharing*. This is due to the reason that each user in *No-Sharing* takes one cab, while users in *VShare* are able to identify carpoolers nearby and share cabs with each other, the number of cabs needed to transport all users are thus reduced. When the number of cabs on the road are decreased in urban areas, some problems emerging in the process of urbanization can be mitigated, e.g., traffic congestion and air pollution. Figure 10(a) shows that *VShare* is an environment-friendly carsharing system that can be applied to improve transportation system.

V. Conclusions

Dynamic carpool commuting systems are getting popular and provide promising benefits to both the society and environment protection. To promote carpool commuting by effectively matching carpoolers, one solution is to build carpool lanes in main airports and bus stops where travelers are crowded, but this method works in designated locations and cannot be widely adopted. Another solution is using centralized servers to identify carpoolers based on historical travel data or on demand travel requests, but this method incurs long matching latency. To overcome the drawbacks of these methods, we propose *VShare*, a dynamic carpool system that leverages the wireless social network characteristic and hierarchical cloud server architecture. Upon receiving a user travel request, *VShare* first utilizes the wireless social network to match possible carpoolers from nearby neighbors. If no carpools are matched through the wireless social network, *VShare* then matches potential carpoolers through a hierarchical cloud server architecture. *VShare* is able to identify carpoolers with short latency due to its usage of the wireless social network and the hierarchical architecture in the cloud for carpooler matching. We evaluated the performance of *VShare* using a cab mobility dataset. Experimental results show the effectiveness of *VShare* in matching carpoolers, reducing user travel expense, minimizing carpool matching latency, increasing users' success rate of catching a cab and minimizing the number of cabs needed to transport a specific number of users. In our future work, we will improve the design of *VShare* so that traveler with different destinations can also be identified as carpoolers. Also, we aim to develop a real application of *VShare* that can deploy on mobile phones.

Acknowledgements

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, and Microsoft Research Faculty Fellowship 8300751. We would like to thank Dr. Andrej Ivanco for his valuable comments.

References

[1] Uber, "https://www.uber.com/."

[2] Lyft, "https://www.lyft.com/."

[3] T. N. Y. Times, "Car-Pooling Declines as Driving Becomes Cheaper, 2011," *http://www.nytimes.com/interactive/2011/01/29/us /20110129-nat-CARPOOL.html, [Accessed in Oct 2015].*

[4] S. DeLoach and T. Tiemann, "Not driving alone: Commuting in the twenty-first century," *Elon University Department of Economics*, 2010.

[5] F. Spielberg and P. Shapiro, "Mating habits of slugs: Dynamic carpool formation in the i-95/i-395 corridor of northern virginia," *Transportation Research Record: Journal of the Transportation Research Board*, no. 1711, pp. 31–38, 2000.

[6] C. C. P. News, "Casual car pool locations," *http://www.ridenow.org/carpool/#locations, [Accessed in Oct 2015].*

[7] B. taxishare, *http://bandwagon.io, [Accessed in Oct 2015].*

[8] N. Liu, Y. Feng, F. Wang, B. Liu, and J. Tang, "Mobility crowdsourcing: Toward zero-effort carpooling on individual smartphone," *International Journal of Distributed Sensor Networks*, 2013.

[9] R. Baldacci, V. Maniezzo, and A. Mingozzi, "An exact method for the car pooling problem based on lagrangean column generation," *Operations Research*, vol. 52, no. 3, pp. 422–439, 2004.

[10] R. Calvo, F. de Luigi, P. Haastrup, and V. Maniezzo, "A distributed geographic information system for the daily car pooling problem," *Computers & Operations Research*, vol. 31, no. 13, pp. 2263–2278, 2004.

[11] S. Ma, Y. Zheng, and O. Wolfson, "T-share: A large-scale dynamic taxi ridesharing service," in *Proc. of ICDE*, 2013.

[12] N. Agatz, A. Erera, M. Savelsbergh, and X. Wang, "Dynamic ride-sharing: A simulation study in metro atlanta," *Transportation Research Part B: Methodological*, vol. 45, no. 9, pp. 1450–1464, 2011.

[13] M. Rigby, A. Krüger, and S. Winter, "An opportunistic client user interface to support centralized ride share planning," in *Proc. of ACM SIGSPATIAL*, 2013.

[14] H. Yotsutsuji, K. Sasaki, and M. Yamamoto, "Availability of volunteer-based dynamic ridesharing with bipartite group in a low-density small community in japan," *Journal of the Eastern Asia Society for Transportation Studies*, vol. 10, no. 0, pp. 1009–1024, 2013.

[15] S. D. Martino, R. Galiero, C. Giorio, F. Ferrucci, and F. Sarro, "A matching-algorithm based on the cloud and positioning systems to improve carpooling." in *Proc. of DMS*, 2011.

[16] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[17] S. Huang, M. Jiau, and C. Lin, "A genetic-algorithm-based approach to solve carpool service problems in cloud computing," *TITS*, vol. 16, no. 1, pp. 352–364, 2015.

[18] Y. Lin and H. Shen, "Autotune: game-based adaptive bitrate streaming in p2p-assisted cloud-based vod systems," in *Proc. of P2P*, 2015.

[19] C. Morency, "The ambivalence of ridesharing," *Transportation*, vol. 34, no. 2, pp. 239–253, 2007.

[20] T. Teubner and C. Flath, "The economics of multi-hop ride sharing," *Business & Information Systems Engineering*, vol. 57, no. 5, pp. 311–324, 2015.

[21] N. Bicocchi and M. Mamei, "Investigating ride sharing opportunities through mobility data analysis," *Pervasive and Mobile Computing*, vol. 14, pp. 83–94, 2014.

[22] D. Zhang, Y. Li, F. Zhang, M. Lu, Y. Liu, and T. He, "coride: carpool service with a win-win fare model for large-scale taxicab networks," in *Proc. of SenSys*, 2013.

[23] J. Hrnčíř, M. Rovatsos, and M. Jakob, "Ridesharing on timetabled transport services: A multiagent planning approach," *Journal of Intelligent Transportation Systems*, vol. 19, no. 1, pp. 89–105, 2015.

[24] N. Chan and S. Shaheen, "Ridesharing in north america: Past, present, and future," *Transport Reviews*, vol. 32, no. 1, pp. 93–112, 2012.

[25] Z. Siddiqi and R. Buliung, "Dynamic ridesharing and information and communications technology: past, present and future prospects," *Transportation Planning and Technology*, vol. 36, no. 6, pp. 479–498, 2013.

[26] K. Kelly, "Casual carpooling-enhanced," *Journal of Public Transportation*, vol. 10, no. 4, p. 6, 2007.

[27] A. Amey, "A proposed methodology for estimating rideshare viability within an organization, applied to the mit community," in *TRB Annual Meeting Procediings*, 2011, pp. 1–16.

[28] M. Furuhata, M. Dessouky, F. Ordóñez, M. Brunet, X. Wang, and S. Koenig, "Ridesharing: The state-of-the-art and future directions," *Transportation Research Part B: Methodological*, vol. 57, pp. 28–46, 2013.

[29] W. He, D. Li, T. Zhang, L. An, M. Guo, and G. Chen, "Mining regular routes from gps data for ridesharing recommendations," in *Proc. of ACM SIGKDD Workshop on Urban Computing*, 2012.

[30] W. He, K. Hwang, and D. Li, "Intelligent carpool routing for urban ridesharing by mining gps trajectories," *IEEE Transactions on ITS*, vol. 15, no. 5, pp. 2286–2296, 2014.

[31] L. Knapen, A. Yasar, S. Cho, D. Keren, A. Dbai, T. Bellemans, D. Janssens, G. Wets, A. Schuster, I. Sharfman, I. Sharfman, and K. Bhaduri, "Exploiting graph-theoretic tools for matching in carpooling applications," *Journal of Ambient Intelligence and Humanized Computing*, vol. 5, no. 3, pp. 393–407, 2014.

[32] A. Amey, J. Attanucci, and R. Mishalani, "Real-time ridesharing: opportunities and challenges in using mobile phone technology to improve rideshare services," *Transportation Research Record: Journal of the Transportation Research Board*, no. 2217, pp. 103–110, 2011.

[33] J. Naoum, R. Cogill, B. Ghaddar, S. Sajja, R. Shorten, N. Taheri, P. Tommasi, R. Verago, and F. Wirth, "Stochastic optimization approach for the car placement problem in ridesharing systems," *Transportation Research Part B: Methodological*, vol. 80, pp. 173–184, 2015.

[34] V. Kumar and M. Bierlaire, "Optimizing locations for a vehicle sharing system," in *Swiss Transport Research Conference*, 2012.

[35] C. de Almeida, H. Gonçalo, and A. Antunes, "Optimization approach to depot location and trip selection in one-way carsharing systems," *Transportation Research Part E: Logistics and Transportation Review*, vol. 48, no. 1, pp. 233–247, 2012.

[36] Y. Huang, F. Bastani, R. Jin, and X. 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.

[37] S. Ma and O. Wolfson, "Analysis and evaluation of the slugging form of ridesharing," in *Proc. of ACM SIGSPATIAL*, 2013.

[38] D. Santos and E. Xavier, "Dynamic taxi and ridesharing: A framework and heuristics for the optimization problem," in *Proc. of IJCAI*, 2013.

[39] W. Herbawi and M. M. Weber, "Evolutionary multiobjective route planning in dynamic multi-hop ridesharing," in *Evolutionary Computation in Combinatorial Optimization*, 2011, pp. 84–95.

[40] B. Cici, A. Markopoulou, E. Frias-Martinez, and N. Laoutaris, "Assessing the potential of ride-sharing using mobile and social data: a tale of four cities," in *Proc. of UbiComp*, 2014.

[41] J. Jung, R. Jayakrishnan, and J. Park, "Dynamic shared-taxi dispatch algorithm with hybrid-simulated annealing," *Computer-Aided Civil and Infrastructure Engineering*, 2015.

[42] M. Jiau, S. Huang, and C. Lin, "Optimizing the carpool service problem with genetic algorithm in service-based computing," in *Proc. of SCC*, 2013.

[43] A. E. Fassi, A. Awasthi, and M. Viviani, "Evaluation of carsharing network's growth strategies through discrete event simulation," *Expert Systems with Applications*, vol. 39, no. 8, pp. 6692–6705, 2012.

[44] M. Nourinejad and M. Roorda, "A dynamic carsharing decision support system," *Transportation research part E: logistics and transportation review*, vol. 66, pp. 36–50, 2014.

[45] J. Yousaf, J. Li, L. Chen, J. Tang, and X. Dai, "Generalized multipath planning model for ride-sharing systems," *Frontiers of Computer Science*, vol. 8, no. 1, pp. 100–118, 2014.

[46] W. Herbawi and M. Weber, "A genetic and insertion heuristic algorithm for solving the dynamic ridematching problem with time windows," in *Proc. of GECCO*, 2012.

[47] M. Piorkowski, N. Sarafijanovoc-Djukic, and M. Grossglauser, "A Parsimonious Model of Mobile Partitioned Networks with Clustering," in *Proc. of COMSNETS*, 2009.

[48] M. Ester, H. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Proc. of Kdd*, 1996.

[49] PlanetLab, "http://www.planet-lab.org/."