Towards Green Transportation: Fast Vehicle Velocity Optimization for Fuel Efficiency

Chenxi Qiu*, Haiying Shen†, Ankur Sarker†, Vivekgautham Soundararaj‡, Mac Devine§, Andy Rindos§ and Egan Ford§
*College of Information Science and Technology, Pennsylvania State University
czq3@psu.edu
†Department of Computer Science, University of Virginia
{hs6ms, as4mz}@virginia.edu
‡Department of Electrical and Computer Engineering, Clemson University
vsounda@clemson.edu
§IBM Research
{wdevine, rindos, egan}@us.ibm.com

***Abstract*—To minimize the fuel consumption for driving, several methods have been proposed to calculate vehicles' optimal velocity profiles on a remote cloud. Considering the traffic dynamism, each vehicle needs to keep updating the velocity profile, which requires low latency for information uploading and profile calculation. However, these proposed methods cannot satisfy this requirement due to (1) high queuing delay for information uploading caused by a large number of vehicles, and (2) the neglect of the traffic light and high computation delay for velocity profile. For (1), considering the driving features of close vehicles on a road, e.g., similar velocity and inter-distances, we propose to group vehicles within a certain range and let the leader vehicle in each group to upload the group information to the cloud, which then derives the velocity of each vehicle in the group. For (2), we propose spatial-temporal DP (ST-DP) that additionally considers the traffic lights. We innovatively find that the DP process makes it well suited to run on Spark (a fast parallel cluster computing framework) and then present how to run ST-DP on Spark. Finally, we demonstrate the superiority of our method using both trace-driven simulation (NS-2.33 simulator and MATLAB) and real-world experiments.**

I. Introduction

Road transportation is considered as one of the major sources of gas emissions, which lead to climate change and ambient air pollution. For instance, with more than 13 million vehicles sold in China in 2014, motor vehicles have emerged as the chief culprit for the throat-choking air pollution in big cities such as Beijing and Shanghai. As a result, there is an increasing need to decrease fuel consumption of vehicles, which is proportional to vehicles' gas emission [1].

Among various strategies to reduce the fuel consumption of vehicles, optimizing vehicle velocity is one of the most efficient methods [2]–[7]. This method outputs the vehicle velocity profile that indicates the velocity at each time point over time from the source to the destination to minimize the number of accelerations and hence fuel consumption.

However, most of these works [2]–[4] require relatively expensive computation devices, such as road side units (RSUs), to help vehicles calculate their velocity profiles, which limits their practical value, especially in rural areas. Some other works [2], [5] make each vehicle communicate with traffic lights to help it approach traffic lights at green whenever possible. However, due to the limited transmission range of vehicles, a vehicle cannot optimize its velocity when it is far away from the traffic lights. Further, these works only consider the next traffic light on the road, which cannot find the optimal velocity profile that leads to the minimum fuel consumption for the entire travelling route.

Actually, calculating the optimal velocity of a vehicle is non-trivial. For each vehicle, its optimal velocity profile depends on the global traffic regulator information (i.e., all the stop signs and traffic lights along the entire source-destination route) and the calculation requires intensive computations, which is beyond the capability of either traffic lights or vehicles' own devices. Recently, a remote computing framework for transportation systems, called *vehicular cloud*, has been proposed to augment these mobile systems' capabilities by migrating computation to more resourceful computers in the cloud [6]–[10]. In the vehicular cloud, each vehicle uploads its information, e.g., velocity and coordinate, to the cloud through base stations (BSs), and the cloud then derives the optimal velocity profile for the vehicle using the dynamic programming (DP) [11]. However, these remote computing methods do not take into account the traffic lights and the DP has high time complexity and delay, so they have very limited practical use in the real-world transportation systems.

Considering that the traffic on a road is highly dynamic and unpredictable, each vehicle needs to periodically update its optimal velocity profile based on the current traffic condition. It requires low latency for information uploading and profile calculation. However, the current remote computing methods cannot satisfy this requirement due to

9781-5090-1445-3/16$31.00 © 2016 IEEE (CloudCom'16)

(1) high queuing delay for information uploading at BSs caused by a large number of vehicles, and (2) the neglect of the traffic light and high computation delay for velocity profile calculation at the cloud. To overcome the above two drawbacks, we design a new remote computing framework, called the *fast Velocity Optimization system (FastVO)*, which is composed of two methods: *congestion-avoidance information uploading* and *trafficlight-considered Spark-based optimal velocity profile calculation*.

Congestion-avoidance information uploading. The objective of this method is to decrease the information uploading delay through decreasing the amount of information to be uploaded to the BSs. Close vehicles on a road have almost the same velocity and inter-distances [12]. By taking advantage of this driving feature of vehicles, FastVO groups vehicles within a certain range and lets the leader vehicle in a vehicle group to upload only its own velocity, coordinate, group vehicles' IDs and the group length (i.e., the distance between the first vehicle and the last vehicle in the group) to the cloud, and let the cloud estimate the velocities and coordinates of all the vehicles in the group based on the driving features. As a result, compared to the previous methods that require each vehicle to upload its information, FastVO reduces the amount of the uploaded information approximately by n times, where n is the number of vehicles in the group. Then, FastVO avoids the congestion in the BSs and reduces the packet transmission delay to the cloud.

TrafficLight-considered Spark-based optimal velocity profile calculation. The objective of this method is to quickly calculate the optimal velocity profile for each vehicle and also consider the traffic lights in DP. Specifically, we propose spatial-temporal DP algorithm (ST-DP) that considers both time and location in DP, which enables to additionally consider the traffic lights. That is, the velocity at the time and location of a red traffic light must be 0. To reduce computation delay, we innovatively find that the DP process makes it well suited to Spark (a fast parallel cluster computing framework) [13]. Spark is a variant of MapReduce [14], a programming model for parallel computing, but is more efficient for iterative computing. Accordingly, we decompose the ST-DP process to a set of independent parts, and carefully design the map and reduce functions and the key for the output of the mapper, so that ST-DP can be conducted as processing different task components in parallel (map function) and then combining the calculated results (reduce function), which are taken as the input in the next iteration if the program has not been finished.

We have conducted both trace-driven simulation (NS-2.33 simulator and MATLAB) and real-world experiments to test the performance of FastVO. Experimental results demonstrate the superior performance of our methods compared with the previous methods.

The remainder of this paper is organized as follows. Section II and Section III present the information uploading and velocity profile calculation in our system, respectively. Section IV evaluates the performance of our proposed methods in comparison with other methods. Section V presents related work. Section VI concludes this paper with remarks on our future work.

II. Congestion-avoidance Information Uploading

The objective of this method is to reduce the amount of information that needs to upload to the cloud in order to reduce the information uploading delay. We notice some driving features for close vehicles on a road including similar velocity and inter-vehicle distance. Based on these features, we form close vehicles to a group and only let the leader vehicle upload its own information, group vehicles' IDs and group length. Then, from this uploaded information, the cloud can recover the information of all other group vehicles based on the driving features.

Each vehicle in a group needs to report to the leader vehicle its vehicle ID and location through either single-hop or multi-hop routing [15]. The leader vehicle maintains an group ID list (which includes the sorted IDs of vehicles based on their position sequence in the lane) and the group length, which needs to upload to the cloud for optimal velocity calculation. Thus, a set of vehicles can form a vehicle group iff they satisfy the following two criterions: all the vehicles are 1) connected with the leader vehicle through single-hop or multi-hop routing, and 2) running in the same lane.

Like [2]–[7], we concentrate on a longitudinal vehicle group, which is composed of n vehicles with position sequence number $\{1,2,...,n\}$ driving in the same lane. Also, we consider a discrete time system, where time $t = 0,1,2,...$ with time slot $\Delta t = 1$ *unit* We use l_t^i to represent the line coordinate of vehicle i at time t, and use v_t^i and a_t^i to represent vehicle i's speed and acceleration at time t, respectively. We use $d_t^{i,j} = |l_t^i - l_t^j|$ to represent the distance between vehicles i and j at time t.

We have two observations for vehicles' movement among close vehicles on a lane [12]: 1) the inter-distances between vehicles are almost the same: $d_t^{1,2} \approx ... \approx d_t^{n-1,n}$, and 2) all the vehicles have almost the same velocity with the leader vehicle: $v_t^1 \approx ... \approx v_t^n$. According to these two observations, we group close vehicles. and let the leader vehicle periodically upload the following information to the cloud: the leader vehicle's coordinate (l_t^1) and velocity (v_t^1), the group length ($l_t^n - l_t^1$), and the group ID list. Then, taking this information as inputs, the cloud can estimate the velocity and coordinates of all other vehicles in the group: $\hat{v}_t^i = v_t^1,\ i = 1,...,n$ and

$$\hat{l}_t^i = \frac{l_t^n - l_t^1}{n-1}(i-1),\ i = 1,...,n, \tag{1}$$

where $\hat{v}_t^i$ and $\hat{l}_t^i$ represent the estimated velocity and coordinate of vehicle i, respectively. Some of the information needed for optimal velocity calculation only needs to be uploaded once, such as vehicle's mass and wheel radius. Each vehicle reports its own such information to the cloud by itself at the beginning.

III. TrafficLight-Considered Spark-based Optimal Velocity Profile Calculation

In this section, we introduce how to quickly calculate the optimal velocity profile to minimize the total fuel consumption of a vehicle using DP, given the vehicle's current velocity (denoted by v_c), its route with source (i.e., current location) and destination, and its driving time constraint (i.e., the maximum time that the driver can accept denoted by T). Different from the previous DP algorithms [6], [7], [11], our ST-DP algorithm additionally considers the effect of traffic lights on a vehicle's movement, which further increases the computation delay. Fortunately, we observe that directed graph building process of DP can be decomposed into a set of independent parts, which can be processed in parallel. Based on this observation, we implement ST-DP using Spark [13], which is a variant of MapReduce but more efficient for iterative computing. In the following, we first introduce the problem statement in Section III-A, then describe the ST-DP algorithm to solve the problem in Section III-B, and finally introduce how we use Spark to implement ST-DP in Section III-C.

A. Problem Statement

We use D to denote the distance between the source and the destination, and use v_t to represent the velocity of the vehicle from time $t-1$ to time t. We use d_t to denote the distance from the source at time t or the location with this $d_t = \sum_{i=1}^{t} v_i$. Suppose $v_{\min}(d_t)$ and $v_{\max}(d_t)$ are the specified maximum speed and minimum speed in the road segment from d_{t-1} to d_t due to speed limit [12]. Then,

$$v_{\min}(d_t) \le v_t \le v_{\max}(d_t). \tag{2}$$

For drivers' comfort and safety issues [11], the acceleration of a vehicle cannot be too large. Hence, the following limitations are imposed on the vehicle acceleration:

$$a_{\min} \le v_{t+1} - v_t \le a_{\max},\ \forall t = 0,1,2,... \tag{3}$$

where $a_{\min}$ and $a_{\max}$ are -1.5 m/s^2 and 2.5 m/s^2 [11]. Notice that when acceleration is negative, it means the velocity is decreased. In addition, since v_0 equals a vehicle's current velocity (denoted by v_c) and its velocity is 0 when it arrives at the destination, we consider the following boundary conditions:

$$v_0 = v_c \text{ and } v_T = 0. \tag{4}$$

Next, we consider the stop signs and red traffic lights. A vehicle must stop in front of each stop sign at location $d(t)$ if its velocity is not 0 at $t-1$. That is,

$$v_t = 0, \text{ if } v_{t-1} \neq 0 \text{ and } d_t \in \mathscr{D}_{\text{stop}} \tag{5}$$

where $\mathscr{D}_{\text{stop}}$ denotes locations of the stop signs in the route. Suppose there are L traffic lights in the entire source-destination route and let $\mathscr{D}_{\text{sig}} = \{d_{\text{sig}}^1, d_{\text{sig}}^2, ..., d_{\text{sig}}^L\}$ denote the set of traffic lights' locations in the route and $\mathscr{T}_{\text{red}}^l$ denote the set of time slots when the l^{th} traffic light is red. Then,

$$v_t = 0, \text{ when } d_t = d_{\text{sig}}^l \text{ and } t \in \mathscr{T}_{\text{red}}^l. \tag{6}$$

The minimum fuel consumption problem: The objective of this problem is to find the optimal velocity profile that minimizes the fuel consumption over a specific source-destination vehicle's route. Let $c_{v_t,v_{t+1}}$ denote the fuel consumption (i.e., edge cost) when the vehicle's velocity is changed from v_t to v_{t+1} and use this velocity in time slot $[t, t+1]$. $c_{v_t,v_{t+1}}$ can be directly calculated by the fuel consumption model introduced in [11] given the vehicle's mass and wheel radius. Then, the minimum fuel consumption problem can be formulated as

$$\min \sum_{t=0}^{T-1} c_{v_t,v_{t+1}} \text{ s.t.} \sum_{t=1}^{T} v_t = D, \text{ and Equ. (2) - (6) are satisfied} \tag{7}$$

It means that the total fuel consumption is minimized with the constraint that the vehicle drives D distance within time T.

B. The Spatial-Temporal DP Algorithm

In the spatial-temporal DP algorithm (ST-DP), we represent the status of a vehicle by a triple point (t, d_t, v_t), which represents that if the vehicle's velocity from time $t-1$ to t is v_t, its travel distance from the source place is d_t. We give this algorithm name because all the points in ST-DP are in the 3-dimensional space containing the spatial, temporal and velocity coordinates. Based on Equ. 4, $(0,0,v_0)$ represents the vehicle status at the source (current location) and $(T,D,0)$ represents the vehicle status when it arrives at the destination. ST-DP builds a graph, which enumerates all possible points from $(0,0,v_0)$ to $(T,D,0)$. Each point (t, d_t, v_t) has an edge directing to ($t+1$, d_t+v_{t+1}, v_{t+1}) with edge cost $c_{v_t,v_{t+1}}$. Also, each point (t, d, v) is associated with an value (called point cost), C_{t,d_t,v_t}, which means the minimum fuel consumption from status $(0,0,v_t)$ to status (t, d, v).

Below, we explain how to build the graph in a centralized manner. The cloud stores the global traffic regulator information. It also pre-calculates and stores $c_{v_t,v_{t+1}}$ for each pair of (v_t, v_{t+1}) ($v_t, v_{t+1} = v_{\min}, ..., v_{\max}$ and satisfy constraint 3), where $v_{\min}$ and $v_{\max}$ are the global maximum speed and minimum speed in all roads in the transportation system. We call it edge cost list.

To build the graph, ST-DP derives all the status points in time sequence, i.e., the points in time $t+1$ are derived only from the points in time t. When $t=0$, the traveling

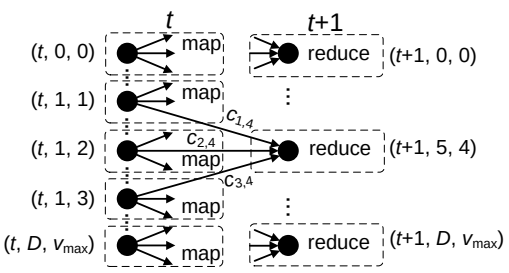


Figure 1. Graph creation in ST-DP.

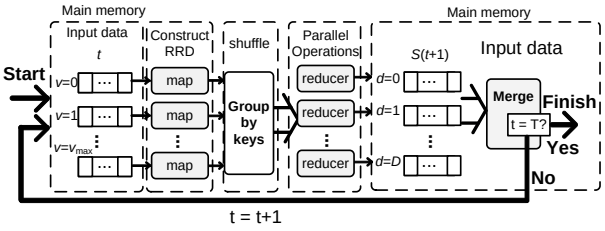


Figure 2. Implementation of ST-DP on Spark.

distance of a vehicle must be 0. Hence, we initiate point cost $C(0,0,v_t)$ by 0. From each point in t, (t, d_t, v_t), we then derive points $(t+1,d_{t+1},v_{t+1})$ considering three cases: (1) the vehicle meets a stop sign (Constraint (5)), (2) the vehicle meets a red traffic light (Constraint (6)), and (3) the vehicle does not meets a stop sign or a red traffic light (Constrains (2) and (3)). In case 1), $d_t \in \mathscr{D}_{\text{stop}}$, based on Equ. (5), if $v_t \neq 0$, then point (t, d_t, v_t) only has an edge directing to ($t+1$, d_t, 0) with edge cost $c_{v_t,v_{t+1}} = 0$. Otherwise, the derivation of points at $t+1$ is the same as case (3). In case (2), $d_t = d^l_{\text{sig}}$ and $t \in \mathscr{T}^l_{\text{red}}$, according to Equ. (6), then point (t, d_t, v_t) only has an edge directing to ($t+1$, d_t, 0) with edge cost $c_{v_t,v_{t+1}} = 0$. For case (3), as shown in Fig. 1, we first derive each point $(t+1,d_t+v_{t+1},v_{t+1})$ when $v_{t+1} = v_{min}(d_{t+1}),...,v_{max}(d_{t+1})$ (Constrain 2) and also satisfies Constrain (3). Then, point (t, d_t, v_t) has edges directing to each ($t+1$, d_{t+1}, v_{t+1}) and $c_{v_t,v_{t+1}}$ equals the pre-calculated value in the edge cost list. For example, in the figure, $(t,1,1)$ points to $(t+1,5,4)$ with edge cost $c_{1,4}$ and other points.

After all points in $t+1$ are derived, we need to calculate the point cost for each $(t+1,d_{t+1},v_{t+1})$. We first identify all the points in t that have directed edges connecting to $(t+1,d_{t+1},v_{t+1})$. We use $S_{t+1,d_{t+1},v_{t+1}}$ to denote the set of (point cost, edge cost)=$(C_{t,d_t,v_t}, c_{v_t,v_{t+1}})$ of these points. Then, $C_{t+1,d_{t+1},v_{t+1}}$ equals the minimum value in all $C_{t,d_t,v_t} + c_{v_t,v_{t+1}}$:

$$C_{t+1,d_{t+1},v_{t+1}} = \min_{(C_{t,d_t,v_t}, c_{v_t,v_{t+1}}) \in S_{t+1,d_{t+1},v_{t+1}}} \{C_{t,d_t,v_t} + c_{v_t,v_{t+1}}\} \tag{8}$$

For example, in Fig. 1, point $(t+1,5,4)$ is directed by points $(t,1,1)$, $(t,1,2)$, $(t,1,3)$ and other points. We use $p(t+1,d_{t+1},v_{t+1})$ to denote the previous point (t,d_t,v_t) that generates the minimum value of all $(C_{t,d_t,v_t} + c_{v_t,v_{t+1}})$:

$$\begin{aligned} & p(t+1,d_{t+1},v_{t+1}) \\ = \ & \arg \min_{(C_{t,d_t,v_t}, c_{v_t,v_{t+1}}) \in S_{t+1,d_{t+1},v_{t+1}}} \left((t,v_t,d_t) | C_{t,d_t,v_t} + c_{v_t,v_{t+1}}\right) \end{aligned} \tag{9}$$

After all points in $t+1$ and their point costs are derived, we then derive all points in $t+2$ and their point costs. This process is repeated until $(T,D,0)$ and its point cost are derived.

In the backtracking algorithm [11], starting from $(T,D,0)$, we iteratively find $p(t+1,d_{t+1},v_{t+1})$ using Equ. (9) until $(0,0,v_0)$ is reached. Consequently, the velocity entries of the points in the discovered path from $(0,0,v_0)$ to $(T,D,0)$ compose the optimal velocity profile.

C. Fast ST-DP Using Spark

The above procedure has high time complexity, which limits its practical use for vehicle velocity optimization. Fortunately, we notice that the features of the ST-DP process make it suitable to run on the Spark, which is a programming model for processing large datasets with a parallel algorithm on a cluster. Spark can outperform Hadoop by 10x [13]. In the following, we first introduce the task execution process on Spark, then introduce how the features of the ST-DP process make it suitable to run on the Spark, and finally present how to execute ST-DP in Spark.

As Fig. 2 shows, the Spark framework has two procedures: `map()` and `reduce()` [13] with two main steps: constructing resilient distributed datasets (RDDs) and parallel operations. In the first step, Spark divides the input into chunks and then passes each chunk to a mapper. The key-value output pairs from each mapper are collected by a master controller and sorted by key. The keys are divided among all reducers, so all key-value pairs with the same key wind up at the same reducer. The reducer combines the data to produce a result, which will be taken as the input in the next iteration if the program has not been finished.

As shown in Fig. 1, in ST-DP, the graph is built at a time sequence, that is, points at $t+1$ are derived from points at t. For each point at t, (t,v_t,d_t), it generates several points $(t+1,d_{t+1},v_{t+1})$. The point cost of a given $(t+1,d_{t+1},v_{t+1})$ depends on the $(C_{t,d_t,v_t}, c_{v_t,v_{t+1}})$ of the points at t that point to $(t+1,d_{t+1},v_{t+1})$, i.e., $S_{t+1,d_{t+1},v_{t+1}}$ and it is calculated based on Equ. (8). Next, taking the output, i.e., points at $t+1$ as the input, the points at $t+2$ and their associated costs are generated. This process repeats until $(T,D,0)$ is reached.

We notice that the first procedure to generate points $(t+1,d_{t+1},v_{t+1})$ from each point (t,v_t,d_t) is the same and this procedure is independent from all points in $t+1$. Also, for different $t+1$, given $S_{t+1,d_{t+1},v_{t+1}}$, the second procedure to calculate $C_{t+1,v_{t+1},d_{t+1}}$ is the same. Then, we can use `map()` for the first procedure and use `reduce()` for the second

procedure, and the output of the second procedure is taken as the input of the first procedure. However, a challenge here is how to find $S_{t+1,d_{t+1},v_{t+1}}$ from the mapper outputs as the input of the second procedure. We notice that all the points at t that point to the same point $(t+1,d_{t+1},v_{t+1})$ have the same (d_{t+1},v_{t+1}). Then, we use (d_{t+1},v_{t+1}) as the key, so that the mapper outputs $(C_{t,d_t,v_t},c_{v_t,v_{t+1}})$ belonging to $S_{t+1,d_{t+1},v_{t+1}}$ from different mappers are sent to the same reducer.

In the following, we introduce how we implement ST-DP by providing `map` and `reduce` functions in Spark. The edge cost list and global traffic regulator information are pre-determined and stored in mappers. In each iteration t, the computation executes as follows. The input of mappers includes all points at t and their associated point costs, i.e., (t,d_t,v_t) and C_{t,d_t,v_t}. Each mapper is assigned a chunk including a few points to handle. The output of each mapper includes $(C_{t,d_t,v_t},c_{v_t,v_{t+1}})$ for each derived point $(t+1,d_{t+1},v_{t+1})$ and its associated key (d_{t+1},v_{t+1}). We introduced the process of this derivation previously in Section III-B. Thus, the set of all the $(C_{t,d_t,v_t},c_{v_t,v_{t+1}})$ with the same key, i.e., $S_{t+1,d_{t+1},v_{t+1}}$, are gathered in the same reducer. Each reducer conducts the computation in Equ. (8) and all reducers output $C_{t+1,d_{t+1},v_{t+1}}$ for all points at $t+1$, $(t+1,v_{t+1},d_{t+1})$. Finally, we check whether $t=T$. If yes, the ST-DP process is finished and we have derived the cost of all the points for ST-DP; otherwise, the output of the reducers becomes the input of the mappers and the iteration continues.

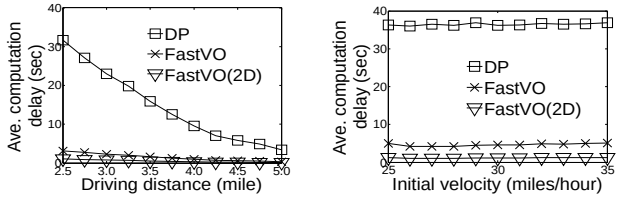
IV. Performance Evaluation

In this section, we compare the performance of FastVO system with two state-of-the-art methods, called *Dynamic programming (DP)* [11] and *Predictive Cruise Control (PCC)* [2]. DP is similar with our approach, except it builds the directed graph in 2-dimensional space (including the spatial and velocity coordinates), which neglects the effect of red traffic lights on vehicles' movement. Also, though DP also uses the cloud to calculate the optimal velocity profile, it does no use parallel computing. In PCC, a vehicle contacts each traffic light to get the traffic light information, and calculates its optimal velocity profile to avoid stopping at the upcoming red traffic light using the subgradient method [12]. We used both trace-driven simulation and real-world experiments to test the performance. We used TCP for all vehicles due to its advantages over UDP in vehicle networks [16], and we set the maximum retransmission time-out by 500ms. We set the mass, the wheel radius, and the drag coefficient of each vehicle (needed in the model) by 1954 kg, 0.363m, and 0.29 [11].

Setting of simulation. In simulation, we used both NS-2.33 and MATLAB based on the real vehicle mobility trace from San Francisco [17], which is a 30-day trace recorded by 536 taxis in San Francisco in May, 2008. In this trace, the destination of taxi is provided and each taxi driver has a tablet that reports timestamp, vehicle ID, GPS coordinate to a central server every 7 seconds. We randomly picked up a route with 5 miles length, and then randomly picked up a vehicle and its 29 following vehicles in this route, where each vehicle can connect to the leader vehicle with the single-hop or multi-hop routing. Hence, the 30 vehicles can form a vehicle group. The simulation takes 10 minutes (from 6:00 pm to 6:10 pm on May 1st, 2008 in the trace) and during the whole process, there are 3 vehicles entering the group and no vehicle leaving the group. The time constraint was set to be 10 minutes. Because drivers cannot strictly follow the suggested velocity when they drive vehicles due to dynamic road traffic and road status, we assume there exists a difference between the suggested velocity and actual velocity for each vehicle at each time point, where the difference follows normal distribution with mean 0 and standard deviation 0.85 mile/hour [18].

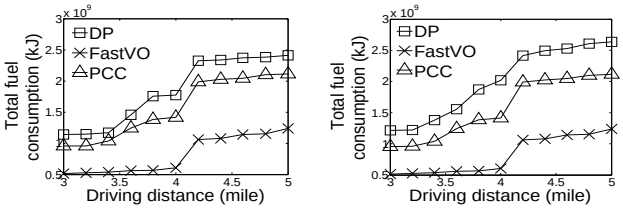
The transmission range of the traffic lights is 80 meters using IEEE 802.11p [2]. In addition, we used the real trace of green-red signal schedule from [2] to simulate the three traffic lights. Unless otherwise indicated, we set the initial velocities of vehicles to 30 miles/hour. The leader vehicle in FastVO and the vehicles in PD report the information to Palmetto every 7 seconds. To simulate the Spark in cloud, we ran 10 mappers and 4 reducers in 7 machines using MATLAB for the computation, where each machine has 8GB memory and two Intel core i5 processor series. We set a queue in BSs and traffic lights with length of 100 packets [19] and 20 packets [20], respectively, and if the queue is full, then the newly entering packets will be dropped.

Setting of real-world experiments. We built an Android application for mobile phones, equipped the phones in three cars, and drove these cars around the campus of Clemson University to test the performance of different algorithms. The route has a length of 2 miles, three traffic lights (located at 0.18 mile, 0.42 mile, and 0.63 mile from the source) and two stop signs (located at 0.56 mile and 0.74 mile from the source). We drove three vehicles with initial velocity 20 miles/hour and we tried to maintain a minimum safety distance of 10 meters. The time constraint was set to be 5 minutes. We used the Palmetto high-performance computing cluster [21] as the cloud. The leader vehicle in FastVO and the vehicles in PD report the information to Palmetto every 10 seconds. We implemented Spark on the Palmetto using 10 servers, each of which has 8GB memory, Intel core i3 processor series, and we ran 10 mappers and 4 reducers on these servers. Currently, Android does not support WiFi ad-hoc network [22]. As an alternative option, we used WiFi Direct API to implement our decentralized network [23], and the communication range of each mobile device is approximately 70m$\sim$80m. By using WiFi Direct API, each mobile device equipped in vehicle can directly communicate with other mobile device without aid of any access point (AP).



(a) Different driving distances (b) Different initial velocities

Figure 3. Optimal velocity profile computation delay in simulation.



(a) Ideal environment (b) Non-ideal environment

Figure 4. Fuel consumption in simulation.

A. Simulation

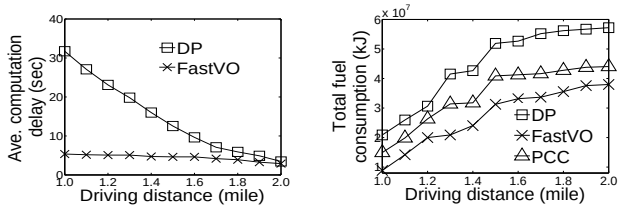
Fig. 3(a)(b) compare the computation delay of DP and FastVO with different driving distances and different initial velocities, respectively. To show the sole advantage of using Spark and considering the red traffic lights, we also tested FastVO without considering the traffic lights, i.e., the directed graph is built in the same way as in DP, denoted by FastVO(2D). In both figures, we find that the computation delay of DP is much higher than that of FastVO and FastVO(2D) in spite of the powerful computation capability of the cloud. FastVO and FastVO(2D) have much lower computation delay than DP because they calculate the optimal velocity profile in parallel using Spark, while DP calculates the profile in a centralized manner. FastVO has slightly higher computation delay than FastVO(2D) because FastVO additionally considers the time coordinate in order to avoid red traffic lights, which increases the time complexity. However, with the consideration of the red traffic lights, FastVO generates much less fuel consumption than DP and FastVO(2D) (DP and FastVO(2D) generate the same optimal velocity profiles) as shown in the experimental results later on. In addition, from Fig. 3(a), we find that as the driving distance increases, the computation delay of DP decreases quickly, while that of FastVO and FastVO(2D) decrease marginally. Driving distance increase leads to the decrease of the remaining distance to the destination for optimal velocity calculation, i.e., fewer points in graph construction, which needs less time in calculation. However, the parallel computing in FastVO and FastVO(2D) greatly improves the time efficiency in calculation, so their computation delay is not high when the driving distance is short.

Finally, we compared the total fuel consumption of DP, FastVO and PCC. Recall that DP does not consider the traffic lights and PCC cannot provide optimal velocity profile when a vehicle is outside of the transmission range of a traffic light. In order to solely test the effectiveness of the calculated velocity profiles on reducing the fuel consumption, we first assume an ideal environment without information uploading delay, information loss, and computation delay, that is, each vehicle can receive its profile in time. We then conduct the testing without the assumption of this ideal environment, that is, with the consideration of these factors. Fig. 4(a) and (b) show the total fuel consumption with and without this assumption. In Fig. 4(a), the fuel consumption follows: DP $>$ PCC $\gg$ FastVO, and the total fuel consumption of FastVO over 5 miles (the length of the whole route) is already about 58% and 50% of that of PCC and DP, respectively. DP has the highest fuel consumption since it does not take into account the effect of traffic light on vehicles' velocities. Then, if a driver has to stop in front of a red traffic light signal, which is not in the profile, it leads to more accelerations and hence more consumed fuel. In PCC, a vehicle lacks the global regulator information, and it can only determine its own velocity profile according to traffic light ahead and hence cannot achieve a global optimal solution. Also, each vehicle in PCC can obtain its optimal velocity only when it is inside of the transmission range of the traffic lights, which just cover three small segments in the route. FastVO considers traffic lights and also achieves global optimal velocity profiles, thus produces the least fuel consumption. In addition, we find that the total fuel consumption increases as the driving distance increases in both figures, because the longer a vehicle drives, the more fuel it consumes.

In Fig. 4(b), the total fuel consumption follows: PCC $>$ DP $\gg$ FastVO, and we can find the that the advantage of FastVO is more significant, i.e., the total fuel consumption of FastVO for 5 miles is about 58% and 47% of that of PCC and DP, respectively. If a vehicle cannot receive its optimal velocity profile in time, it cannot always follow the optimal velocity, leading to more fuel consumption. The fuel consumption of DP is increased compared to Fig. 4(a) due its information uploading delay, information loss rate and computation delay. FastVO has similar performance as Fig. 4(a) because of its low information uploading delay, information loss rate and computation delay caused by its two methods. Though PCC has low communication delay and information drop rate, it still has much higher fuel consumption than FastVO since vehicles do not have optimal velocity profiles to follow when they are out of the transmission ranges of the traffic lights and its solution is sub-optimal.

B. Real-World Experiments

We further compare the computation delay of DP and FastVO over different driving distances in Fig. 5(a). From the figure, we find that the computation delay of DP is higher than that of FastVO. FastVO has lower computation delay



(a) Computation delay (b) Fuel consumption

Figure 5. Real-world experiments.

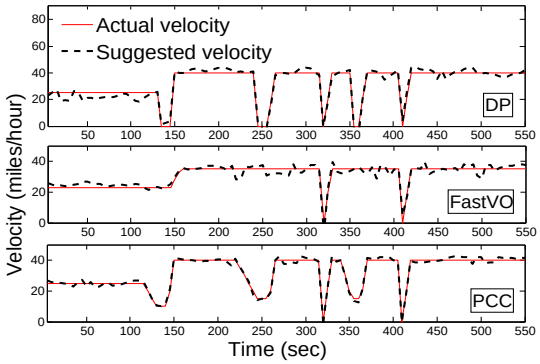


Figure 6. Actual velocity vs. suggested velocity.

than DP because it calculates the optimal velocity profile in parallel, which saves much time for velocity calculation compared with DP. Similar to the simulation results (Fig. 3(a)), we also see that as the driving distance increases, the computation delay of DP decreases quickly, while that of MapReduce and FastVO decrease marginally due to the same reasons.

Finally, we compared the total fuel consumption of DP, FastVO, and PCC. We calculated the fuel consumption of the vehicle using the fuel consumption model [11] based on the vehicle's velocity over time. Fig. 5(b) shows that the fuel consumption follows: PCC $>$ DP $>$ FastVO due to the reasons explained in Fig. 4(a) and (b). The total fuel consumption of FastVO over 2 miles (the length of the whole route) is about 85% and 65% of that of PCC and DP, respectively. The results confirm that FastVO is the most fuel efficient. As mentioned previously, a vehicle cannot strictly follow the suggested velocity and thus needs to periodically update its optimal velocity profile. To verify this, we tested the actual velocities of our vehicles. In the test, our vehicles try to follow the suggested optimal velocity. Fig. 6 compares the suggested velocity and the actual velocity of the leader vehicle in the three systems. The results confirm that vehicles cannot always follow the suggested velocity profiles in practice. Also, as we expected, we find that the FastVO has fewer accelerations compared with DP and PCC and hence generates less fuel consumption.

V. Related work

Vehicle networks. During recent few years, many works have been proposed to improve the performance of vehicle networks. Son *et al.* [24] investigated the benefits of bit rate selection in a vehicular network. They also proposed a simple bit rate selection algorithm to ensure the packet delivery rate in vehicular networks. Shevade *et al.* [25] presented a high-bandwidth vehicular content system that provides high-bandwidth content access to vehicular passengers by utilizing opportunistic connections to Wi-Fi access points along the road. Considering that a multi-hop vehicular ad-hoc network (VANET) requires high packet delivery ratio for safety application, Xiang *et al.* [26] proposed a packet-value-based distributed data dissemination protocol to guarantee the packet delivery. Chen *et al.* [27] presented a lane level cooperative collision avoidance (LCCA) system to avoid chain vehicle collisions using onboard sensors. LLCA disseminates emergency message if there is a sudden change of velocity to avoid further collision. Different from these works, our work focuses on how to leverage the features of vehicles driving in a lane to reduce the packets needed to upload to report each vehicle's status.

Speed optimization. Recently, many methods have been proposed for speed profile optimization. Asadi and Vahidi [2] proposed a control algorithm that adapts the velocity profile to guarantee that a vehicle approaches a traffic light at green whenever possible. The authors used a short-range radar and traffic light information to predictively schedule a suboptimal velocity profile and implemented the algorithm in an existing cruise control system. Ozatay *et al.* [4] presented a non-linear velocity calculation problem with several user defined constrains (e.g., driver comfort constraint) and they derived the velocity using DP with the objective of fuel consumption minimization. In [2], [6], [7], algorithms based on traffic and topographic information of the road for fuel consumption reduction have been proposed. However, these methods cannot either achieve the optimal solution or neglect the effect of traffic lights on vehicles' movement. Different from these works, our method can achieve the optimal solution and considers the red traffic lights. Hence, our method is more practical compared to the previous works.

VI. Conclusion

For high fuel efficiency, it is important for vehicles to receive their optimal velocity profiles in time to reduce fuel consumption. Previous remote computing frameworks require each vehicle to periodically upload its information to the cloud, which calculates its optimal velocity profile. However, the large amount of data uploaded from vehicles may generate high congestion at BSs, which leads to high queueing delay. Also, using the dynamic programming (DP) may lead to a high computation delay. Further, these previous methods neglect the effect of red traffic lights on vehicle velocity profiles. To handle these problems, in this paper, we propose a new remote computing framework, called the fast Velocity Optimization system (FastVO), which is com-

posed of two methods: congestion-avoidance information uploading and trafficlight-considered Spark-based optimal velocity profile calculation. We conducted extensive trace-driven simulation and real-world experiments to compare FastVO with previous methods using various metrics including information uploading delay, information loss rate, computation delay, and fuel consumption. Experimental results show that FastVO outperforms the previous remote computing framework method. In our future work, we will further take into account human and economic factors for velocity calculation.

ACKNOWLEDGEMENTS

This research was supported in part by U.S. NSF grants NSF-1404981, IIS-1354123, CNS-1254006, IBM Faculty Award 5501145 and Microsoft Research Faculty Fellowship 8300751.

REFERENCES

[1] E. Hellstrom and M. Ivarson, "Average annual emissions and fuel consumption for gasoline-fueled passenger cars and light trucks," *USEPA*, 2008.

[2] B. Asadi and A. Vahidi, "Predictive cruise control: Utilizing upcoming traffic signal information for improving fuel economy and reducing trip time," *Trans. CST*, 2011.

[3] C. Raubitschek, N. Schutze, E. Kozlov, and B. Baker, "Predictive driving strategies under urban conditions for reducing fuel consumption based on vehicle environment information," in *Proc. of FISTS*, 2011.

[4] E. Ozatay, U. Ozguner, S. Onori, and G. Rizzoni, "Analytical solution to the minimum fuel consumption optimization problem with the existence of a traffic light," in *Proc. of DSCC*, 2012.

[5] E. Hellstrom, J. A. M. Ivarson, and L. Nielsen, "Look-ahead control for heavy trucks to minimize trip time and fuel consumption," *Control Eng. Pract*, 2009.

[6] H. Khayyam, S. Nahavandi, and S. Davis, "Adaptive cruise control lookahead system for energy management of vehicles," *Exp. Syst. Appl.*, 2012.

[7] S. Park, K. A. H. Rakha, and K. Moran, "Predictive eco-cruise control: Algorithm and potential benefits," in *Proc. of FISTS*, 2011.

[8] J. Liu, H. Shen, and L. Chen, "Corp: Cooperative opportunistic resource provisioning for short-lived jobs in cloud systems," in *Proc. of IEEE CLUSTER*, 2016.

[9] J. Liu and H. Shen, "A low-cost multi-failure resilient replication scheme for high data availability in cloud storage," in *Proc. of HiPC*, 2016.

[10] A. Sarker, C. Qiu, H. Shen, A. Gily, J. Taibery, M. Chowdhuryz, J. Martinx, M. Devine, and A. Rindos, "An efficient wireless power transfer system to balance the state of charge of electric vehicles," in *Proc. of IEEE ICPP*, 2016.

[11] E. Ozatay, S. Onori, J. Wollaeger, U. Ozguner, G. Rizzoni, D. Filev, J. Michelini, and S. D. Cairano, "Cloud-based velocity profile optimization for everyday driving: A dynamic-programming-based solution," *Trans. ITS*, 2014.

[12] K. Yi and Y. D. Kwon, "Vehicle-to-vehicle distance and speed control using an electronic-vacuum booster," *JSAE review*, 2001.

[13] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: cluster computing with working sets," in *Proc. of HotCloud*, 2010.

[14] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *Proc. OSDI*, 2004.

[15] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *Proc. of Sigcomm*, 1994.

[16] S. Hu, H. Liu, L. Su, H. Wang, T. F. Abdelzaher, P. Hui, W. Zheng, Z. Xie, and J. A. Stankovic, "Towards automatic phone-to-phone communication for vehicular networking applications," in *Proc. of Infocom*, 2014.

[17] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser, "A parsimonious model of mobile partitioned networks with clustering," in *Proc. of COMSNETS*, 2009.

[18] S. M. Abuelenin and A. Y. Abul-Magd, "Empirical study of traffic velocity distribution and its effect on vanets connectivity," in *Proc. of ICCVE*, 2006.

[19] A. Eryilmaz and R. Srikant, "Fair resource allocation in wireless networks using queue-length-based scheduling and congestion control," in *Proc. of Infocom*, 2005.

[20] J. Auge and J. Roberts, "Buffer sizing for elastic traffic," in *Proc. of NGI*, 2006.

[21] "Palmetto." http://citi.clemson.edu/palmetto/. [Accessed: July 2015].

[22] C. K. Toh, *Ad hoc mobile wireless networks: protocols and systems*. Prentice Hall PTR, 2001.

[23] "WiFi-direct." http://www.wi-fi.org/discover-wi-fi/wi-fi-direct. [Accessed: July 2015].

[24] G. Son and P. A. Ward, "Bit-rate selection in multi-vehicular networks," in *Proc. of AINA*, 2013.

[25] U. Shevade, Y.-C. Chen, L. Qiu, Y. Zhang, V. Chandar, M. K. Han, H. H. Song, and Y. Seung, "Enabling high-bandwidth vehicular content distribution," in *Proc. of CoNext*, 2011.

[26] Q. Xiang, X. Chen, L. Kong, L. Rao, and X. Liu, "Data preference matters: A new perspective of safety data dissemination in vehicular ad hoc networks," in *Proc. of Infocom*, 2015.

[27] L.-W. Chen and P.-C. Chou, "A lane-level cooperative collision avoidance system based on vehicular sensor networks," in *Proc. of Mobicom*, 2013.