2022-05-20

Graph-based Heuristic Solution for Placing Distributed Video Processing Applications on Moving Vehicle Clusters

Kanika Sharma
South East Technological University, kanika.sharma@waltoninstitute.ie

Bernard Butler
South East Technological University, bernard.butler@setu.ie

Brendan Jennings
Technological University Dublin, brendan.jennings@tudublin.ie

Recommended Citation

K. Sharma, B. Butler and B. Jennings, "Graph-based Heuristic Solution for Placing Distributed Video Processing Applications on Moving Vehicle Clusters," in IEEE Transactions on Network and Service Management, doi: 10.1109/TNSM.2022.3173913.

Funder: Science Foundation Ireland

Graph-based Heuristic Solution for Placing Distributed Video Processing Applications on Moving Vehicle Clusters

Kanika Sharma *(Member, IEEE),* Bernard Butler *(Member, IEEE),* and Brendan Jennings *(Member, IEEE)*

***Abstract*—Vehicular fog computing (VFC) is envisioned as an extension of cloud and mobile edge computing to utilize the rich sensing and processing resources available in vehicles. We focus on slow-moving cars that spend a significant time in urban traffic congestion as a potential pool of onboard sensors, video cameras, and processing capacity. For leveraging the dynamic network and processing resources, we utilize a stochastic mobility model to select nodes with similar mobility patterns. We then design two distributed applications that are scaled in real-time and placed as multiple instances on selected vehicular fog nodes. We handle the unstable vehicular environment by a), Using real vehicle density data to build a realistic mobility model that helps in selecting nodes for service deployment b), Using community-detection algorithms for selecting a robust vehicular cluster using the predicted mobility behavior of vehicles. The stability of the chosen cluster is validated using a graph centrality measure, and c), Graph-based placement heuristics is developed to find the optimal placement of service graphs based on a multi-objective constrained optimization problem with the objective of efficient resource utilization. The heuristic solves an important problem of processing data generated from distributed devices by balancing the trade-off between increasing the number of service instances to have enough redundancy of processing instances to increase resilience in the service in case of node or link failure, versus reducing their number to minimize resource usage. We compare our heuristic to a mixed integer program (MIP) solution and a first-fit heuristic. Our approach performs better than these comparable schemes in terms of resource utilization and/or has a lesser service latency when compared to an edge computing-based service placement scheme.**

***Index Terms*—Fog Computing, Vehicular Fog Computing (VFC), Vehicular Cloud Computing, Intelligent Transport Systems, Flexible Service Model, Internet of Things, Service Placement, Resource Allocation.**

I. Introduction

The increasing amount of data generated from Internet of Things (IoT) devices has resulted in isolated sources of data that are not fused with other data sources and hence are not fully utilized. On the other hand, the emergence of powerful machine learning and deep learning algorithms requires large amounts of data to make accurate inferences. Most of the surveillance carried out on roads or other public places is through static cameras that send the collected data to the backend server [1]. This approach leads to increasing operational costs in deploying dedicated hardware as well as using expensive bandwidth to send this data to the cloud through dedicated links.

The concept of vehicular fog computing (VFC) [2] is derived from using the available and under-utilized vehicular resources, like in-built sensors, processors, dashboard cameras, advanced onboard units (OBUs), etc., for both crowdsensing and processing data. The VFC paradigm aims to make computation more efficient by leveraging the processing and communication capacity of the vehicles, instead of offloading computation to the edge servers or the cloud [3]. VFC extends the intelligence of mobile edge computing [4], [5] and fog computing [6] to the vehicular network, in an attempt to increase the processing capacity to meet the resource requirements for vehicular and infotainment applications. This novel system of distributed service deployment reduces the traffic at the core network, saves the network bandwidth in sending local and contextual data to the cloud, and also reduces end-to-end latency [7], [8].

VFC also reduces the need for installing infrastructure to facilitate Intelligent Transport Systems, for improving vehicular flows and reducing congestion, for recording data for road condition monitoring to detect potholes, accidents, etc., and increasing commuter safety, which has been theorized for more than a decade but has not been implemented [9]. The use of Roadside Units (RSUs) or edge servers for meeting the demands of vehicular applications has also been explored, but it has limitations due to the mobility of vehicles. These RSUs have limited coverage on the highways and have limited sojourn time with moving cars. Thus, provisioning services on RSUs can increase the control cost of service migration between different RSUs. On the other hand, the OBUs on self-driving cars have evolved in their processing capability and their ability to communicate with neighboring vehicles and keep open connections with edge or cloud servers. These vehicles also have a full System on a chip (SOC), for example, a Tesla car has 4 LPDDR4 RAM chips, with complete redundancy to have failure resistance for any system on board [10].

Many recent studies on VFC have focused on latency-sensitive applications that are safety-critical [11], [12]. In this study, we look at a very novel use-case of making opportunistic vehicle clusters in urban sections of the city and leveraging the resources on these vehicles to collect and process data. Most of the existing works on VFC either use very simplistic

K. Sharma and B. Butler are with the Walton Institute for Information and Communication Systems Science, Waterford Institute of Technology, Ireland.E-mail: kanika.sharma@waltoninstitute.ie, bbutler@ieee.org

B. Jennings is with Technological University Dublin (TU Dublin), Ireland.E-mail:brendan.jennings@tudublin.ie

K. Sharma, B. Butler and B. Jennings are with SFI CONNECT Research Centre for Future Networks and Communications, Ireland

mobility models [13], like considering a straight road segment with constant speed or consider only parked vehicles [3]. Many works also consider taxis and buses as potential fog nodes as their trajectories are more predictable [14]. We, however, want to focus on using the embedded sensors on any vehicle whose owner is willing to contribute the data. Another problem is that many existing works on VFC consider a static and composite service template, which is not suitable for the dynamic vehicular environment. We introduce distributed and flexible services that can be adapted according to the resource requirement, are suitable for the heterogeneous and distributed resources on vehicles, and can be reconfigured easily. Instead of optimizing only computation or communication resources, we jointly optimize both link and processing costs. To select vehicular nodes that are more probable to stay together, we introduce a vehicular node selection scheme and a mobility-aware service placement heuristic.

For this system to operate properly, we first introduce a distributed, graph-based service model where each component or task can be scaled to multiple task instances based on the amount of data collected. We also leverage the mobility pattern of vehicles, stuck in high density/congested traffic to estimate the ongoing availability of these vehicles to perform tasks. The distributed services we propose are scaled in real-time and deployed on the vehicle cluster such that we get a robust initial placement with less need to reconfigure services. Our model can support many applications including sensing applications like pedestrian detection to understand human engagement with coffee shops, gas stations, and other locations. The collected data can also be used to detect congestion and study usage patterns of roads, as part of building *Smarter* cities[15].

In this paper, the following contributions are made to introduce a distributed and scalable service model that can be effectively placed on a group of closely moving vehicles by leveraging their historic mobility patterns:

- A mathematical formulation leveraging the mobility-awareness of infrastructure and a novel and distributed service model is introduced. The scaling and placement of the services are modeled as a bi-objective, constrained optimization problem with an objective of efficient communication and computation resource utilization.
- Instead of focusing on widely researched latency-sensitive applications, we introduce a novel use case of initiating opportunistic clusters to collect and process data, using just macroscopic vehicular density data. We study how traffic reaches congestion levels at different occupancy rates and other traffic patterns. We then calibrate our microscopic mobility model using the real vehicle density data. The mobility model is built from our extensive work on studying predictability of vehicular flows and estimating computation and communication capacity of vehicle clusters in our previous work [16].
- We introduce a service model where each application is made of inter-related tasks and each task can also be scaled to multiple task instances, to increase resilience in the service in case of node or link failures. Two such applications are profiled in the paper to understand the resource usage of such applications.
- We then leverage a community-detection-based node selection scheme to study the collective availability of vehicular nodes in a cluster. We then introduce an effective graph-theory-based heuristic that promotes placing task instances optimally within the vehicle clusters. Our approach outperforms an MIP, and a baseline, first-fit approach. Our approach also results in lesser service latency compared to mobile edge computing-based placement.

The paper is organized as follows: Section §2 highlights the related research undertaken on task offloading in VFC and vehicular crowdsensing (VCS). Section §3 introduces the terminology used in the paper and gives a detailed system model. In section §4, we specify the two distributed application types considered in this work. We then give details on the network topology and distributed service model along with the notations used for the mathematical modeling. Section §5 covers the service scaling and placement constraints and the infrastructure constraints. The section also details the mobility model and the objective function of the constrained optimization problem. Section §6 describes the community-detection-based node selection for cluster formation and the graph-based service placement heuristics. Section §7 has a detailed evaluation of the introduced technique compared to the MIP and first-fit solution. We also show the performance of our schemes through service time and the state of the selected nodes in a cluster over time. The paper is concluded in section §8 where future work has also been suggested.

II. Related works

We discuss the existing task offloading and service placement schemes in VFC models. We highlight the challenges in implementing task offloading in a dynamic vehicular environment and discuss existing schemes addressing these challenges.

A. Task offloading in Vehicular Fog Computing

Task offloading schemes in VFC are widely researched and are designed to minimize processing latency [17], without compromising the quality-of-service (QoS) [18] also focusing on efficient resource allocation [19]. Most of these works focus on utilizing the available mobile edge computing infrastructure for carrying out compute-intensive tasks. Liu et al. [20] have introduced a three-layer service architecture for offloading vehicular applications in vehicular fog, fog server, and the central cloud. To solve the Probabilistic Task Offloading problem they introduce an alternating direction method of multipliers (ADMMs) and particle swarm optimization (PSO), to divide the problem into multiple unconstrained sub-problems that iteratively reach an optimal solution.

Liang et al. [21] suggest the use of public transport facilities like buses and taxis as fog nodes to reduce the randomness of vehicle movement with fixed bus trajectories. To solve the interruption problem caused by vehicle mobility as well as the problem of delay and reliability loss, they introduced a low-latency information distribution scheme for VFC. They study network topology dynamics to evaluate and predict connection status between fog nodes and the adjacent vehicles. Qiao et al.

[22] takes advanced driver assistant systems and autonomous driving as the use case for a distributed and collaborative task offloading scheme with a guarantee of low communication and computation latency. They work on removing redundant computation tasks based on task similarity and computation capacity. Vehicles are partitioned into the task computing sub-cloudlet to provide underutilized communication and computation resources. Vehicles with lesser similarities are partitioned into the task offloading sub-cloudlet to assign their computation tasks to edge infrastructures. In our previous work, we first introduced the problem of service mapping and service placement in vehicular networks as an Integer Linear Program (ILP) with an objective of efficient network bandwidth utilization [23]. We have built on our previous work to introduce a more realistic mobility model, utilised a more complex service model, differentiating between tasks based on their functionality, and focused on specific applications in this paper.

Lee et al. [3] introduced a reinforcement learning-based resource allocation model for the continuous and high dimensional action-space in a VFC environment. They use a simple vehicular mobility model for parked vehicles and a realistic mobility model based on Zurich traffic traces [24] to determine vehicles arriving and departing from parking lots. Iqbal et al. [25] developed a blockchain-based, distributed reputation ledger to identify malicious vehicles. They then propose a framework to handle peak workloads using nearby fog vehicles and the RSUs. The reputation score is awarded to the vehicles upon task completion to enable a decision model for task assignment.

B. Crowdsensing in vehicular networks

VCS utilizes the available sensing capability of vehicles and their mobility pattern to accomplish sensing tasks. The aim of a VCS system is different from a VFC system but has common challenges like modeling vehicular mobility and the need for an incentive mechanism for the VCS system. Zhao et al. [26] derived a long-term strategy to build a deep reinforcement learning-based incentive mechanism. They model the vehicle dynamics via a dynamic radio channel with a selection of sine, piece-wise linear, and Markov-chain channel models. Edge devices are also used for detecting parking space availability. Grassi et al. [27] presented the feasibility of deploying image-based, machine learning techniques at the network edge. They use smart cameras placed on dashboards to capture information related to parking availability without any human intervention. Zhu et al. [28] focuses on the challenging task of finding parking availability for autonomous vehicles. They collect parking information from crowdsensing and use VFC to estimate parking availability and inform client vehicles. Zhu et al. [29] also introduced a context-aware task allocation scheme to jointly optimize Quality of Information and processing latency.

C. Role of VFC and VCS

As pointed out in this section, there is a lot of active research in the field of VFC and VCS for many different application profiles and with different performance metrics as an objective. Many of these works focus on latency-sensitive applications. Our work focuses on a novel idea of utilizing opportunistic vehicle clusters for data collection and processing. Instead of focusing on the vertical scaling of services to the edge or fog nodes, and to the Cloud, we focus on horizontal scaling and placement of distributed video collection and object-detection applications on nearby vehicles. We use very limited macroscopic vehicular density data to deploy resilient services with efficient resource utilization.

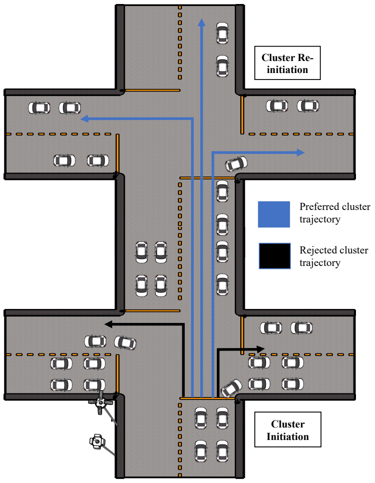


Fig. 1: Vehicle Clusters form, but membership changes over time. Clusters accept service chain placement requests from RSUs and perform service chain scaling and placement.

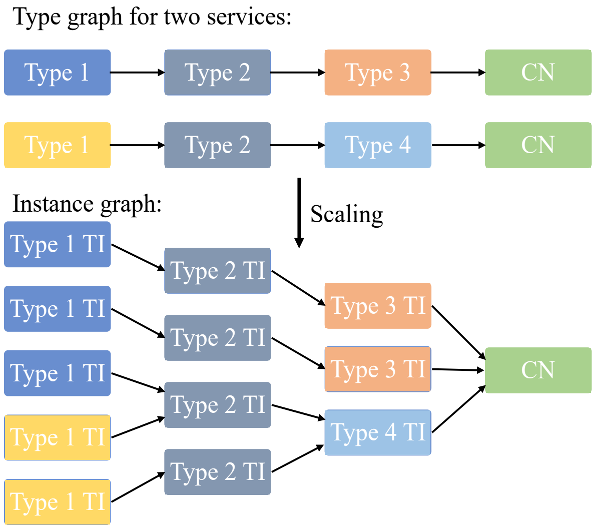


Fig. 2: Service model depicting tasks and their inter-dependencies. The Type graph is scaled to the Instance graph based on the resource state of the vehicle cluster.

III. System Model

This section first introduces the terminology used in the paper. The section also gives details of the proposed system model for selecting a well-connected cluster and managing the placement of the services on the vehicle cluster.

A. Terminology

- **Vehicle Clusters**: As vehicles slow down at intersections or busy urban routes, we recruit a group of vehicular nodes that are likely to stay together, based on their previous mobility pattern. Each vehicular node subscribes to the service of leasing its resources and has a cluster cohesion probability (CCP) based on its microscopic mobility pattern. The vehicle clusters are represented as undirected graphs. The nodes in the vehicle cluster are referred to as vehicular nodes or nodes interchangeably in the paper.
- **Control Node (CN)**: The CN coordinates the vehicle cluster(s) and routes messages to/from clients/RSUs to other nodes in the cluster. The CN is selected based on its betweenness centrality, which is a metric popularly used in social network analysis. For any node, the betweenness centrality is calculated as the fraction of the shortest paths between any pair of nodes in the cluster that pass through that node. From a network point of view, the node with the highest betweenness centrality is a critical point of information flow in a cluster.
- **RSUs**: The RSUs are edge devices with much higher resource capacity relative to the vehicle nodes. The RSU receives requests from clients and initiates a vehicle cluster at an intersection. The RSU selects both vehicle cluster and the CN using the historic mobility patterns of the registered vehicles that is also stored at the RSU. The CN maintains the state of the cluster and sends the metadata of unplaced tasks to the RSUs.
- **Task**: Tasks are the smallest unit of a video processing service. Based on the application, tasks could have different functionality like data filtering or data compression. Tasks could also be more complex and processing-intensive like local object detection. The application types and the example tasks are described in the application section (Secion IV-A). Each task is scaled to multiple task instances (TIs) to handle multiple sources of data and to process data streams in parallel. The multiple TIs also increase the resilience of the service against node failures (when vehicles leave) and link failures (when connectivity is lost).
- **Service**: Each service constitutes different types of tasks with varying functionality. The linear chain of tasks is called the Type graph. Each task can have multiple TIs as depicted in Fig.2. We place two different services on the vehicle cluster, with some shared tasks (Type 2) to promote the reuse of TIs. This ‘Type graph’is scaled to an ‘Instance graph’by both the ILP solver and the service scaling and placement heuristic proposed in this paper. Instead of the linear chain of tasks, the service model can also be modeled as a directed acyclic graph (DAG). Our service placement scheme ensures that the data flow passes through all tasks and the data is processed before it reaches the CN or RSU. We ensure this via the in-network processing constraint (modeled as Eq. 2). The DAG service model can be used when the data flow does not need to pass through each task. Theoretically, the DAG service model can be considered for the service placement model by changing service-level constraints.

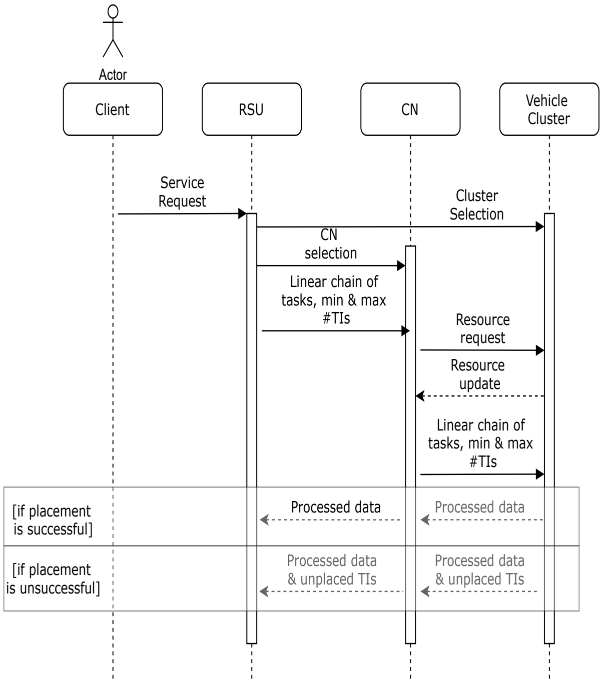

Actor
Client
RSU
CN
Vehicle Cluster
Service Request
Cluster Selection
CN selection
Linear chain of tasks, min & max #TIs
Resource request
Resource update
Linear chain of tasks, min & max #TIs
[if placement is successful]
Processed data
Processed data
[if placement is unsuccessful]
Processed data & unplaced TIs
Processed data & unplaced TIs

Fig. 3: Sequence diagram depicting the order of interactions between the client, the RSU, the CN and the vehicle cluster.

The service placement is managed with the coordination of the cluster’s CN and the RSUs. The RSU receives requests from clients to deploy services. The client could be a one-off vehicle node moving along with the cluster or a surveillance request from traffic authorities. In this paper, we assume that the service request is received and decomposed by the RSU in the form of a linear chain of tasks. As depicted in Fig.1, the RSU detects the presence of vehicles that have previously subscribed to a brokerage service and hence are prepared to lease their resources for service provisioning. The RSU also stores and updates the database of the mobility patterns of these vehicles. Each vehicle has a probability of taking a certain trajectory based on its historic mobility pattern. Based on the preferred trajectories of the cluster, each vehicle node has a certain probability of following the cluster trajectory. A weighted graph is created where each link is weighted by the probability of two nodes staying together for a duration of time, called the CCP. Based on this graph, the most well-connected cluster is selected, and the vehicular node with the highest connectivity is elected as the CN, based on a graph centrality measure.

Once the vehicle cluster and the CN are selected the process of service placement begins. The process of service placement and the order of interactions between the client, the RSU, the selected CN, and the vehicle cluster is represented in Fig.3. The RSU sends the linear Type graph to the CN in the form of docker images. The CN also collects the updated resource information from the vehicular cluster. Initially, the minimum number of instances of each task is placed on the cluster, to process multiple data flows in parallel. Based on the number of

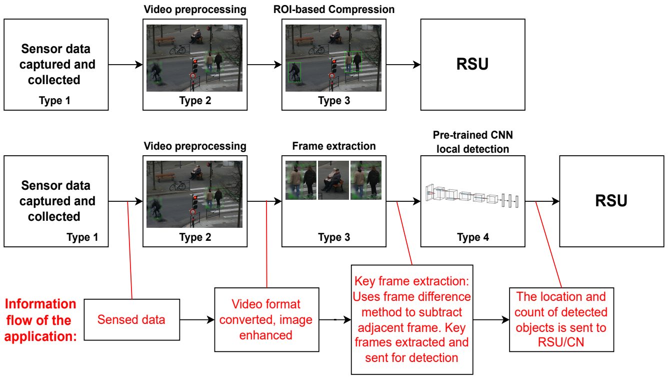

Video preprocessing
ROI-based Compression
Sensor data captured and collected
Type 1
Type 2
Type 3
RSU
Video preprocessing
Frame extraction
Pre-trained CNN local detection
Sensor data captured and collected
Type 1
Type 2
Type 3
Type 4
RSU
Information flow of the application:
Sensed data
Video format converted, image enhanced
Key frame extraction: Uses frame difference method to subtract adjacent frame. Key frames extracted and sent for detection
The location and count of detected objects is sent to RSU/CN

Fig. 4: Two types of applications of chain length 3 and 4.

TABLE I: Task Function, Techniques and Commonly-Used Algorithms for Application 1 Tasks

Task Type	Task Function	Task Techniques	Algorithms
Type 1	Real-time video capturing		
Type 2	Video preprocessing	Image Enhancement, Noise reduction	Gaussian blur, Simple thresholding
Type 3	Video compression	non RoI-based compression RoI-based compression,	Huffman encoding

TABLE II: Task Function, Techniques and Commonly-Used Algorithms for Application 2 Tasks

Task type	Task function	Task techniques	Algorithms
Type 1	Video collection		
Type 2	Video pre-processing	Image enhancement/ Noise reduction	PCA tranformation Noise removal using Wiener filter
Type 3	Feature Extraction	Regularization/ Dimensionality reduction	Independent Component Analysis
Type 4	Object detection	Local, pre-trained model for object detection	Faster R-CNN, YOLO, tinyYOLO

video TIs and the amount of processing required, the heuristic scales to more processing TIs by requesting TI images from the CN. If there are no more potential nodes left to place new TIs, the collected data is sent to the nearest RSU with the remaining TIs in the linear chain left to be placed for processing. The RSU can request a re-initiated cluster to place the remaining processing TIs on it.

IV. Model

A. Application Type

We consider a linear chain of video collection and processing tasks. We place multiple video collection TIs to increase the scale of video collection which results in better accuracy for object detection applications. To process multiple video streams, we scale all the processing tasks to multiple TIs, to utilize the limited processing capacity of vehicles. The multiple TIs also increases the resilience of the service in a dynamic vehicular network where mobility of vehicles increases node and link failures. We present two distributed services that follow our distributed service model:

1) **Data Collection service**: For this kind of service, an initiated cluster acts as “moving sensors” and only pre-processing and compression is carried out at the cluster. Most of the application-specific processing is carried out on more powerful edge computing nodes at RSUs. As an example, we take an application where Type 1 is a video capturing TI, Type 2 is a video pre-processing TI. Type 3 is a video compression TI.

   To profile this application, at Type 1 TI, we first capture the video using OpenCV. At Type 2 TI, the video is pre-processed using Gaussian blur and simple thresholding. For Type 3 TI, the images are compressed using RoI-based image compression. The task function, techniques and commonly-used algorithms are tabulated in Table I for this data collection application.

   This collected data is sent to the mobile edge for an application that requires specific data over time like traffic monitoring and traffic management. Similarly, a large amount of data can also be collected for applications that require inference from more sophisticated Deep Neural Network (DNN) models that demand powerful cloud computing devices. For example, multiple 3D road maps can be generated from multiple sources of video data. This kind of application requires real-time and local information. Our service model sends pre-processed and compressed data, reducing the overhead of data transmission, for the data-intensive map generation task that can then be executed on mobile-edge computing devices.

2) **Object Detection application**: The aim of building this distributed object detection application is to deliver a prompt and communication-efficient data processing service leveraging the resources available in moving vehicles. For this kind of application, all the processing of the collected data is executed on the vehicle cluster. Such applications have local context and scope, for example, using object-detection techniques to identify vulnerable pedestrians and alerting drivers in the vicinity. The Type 1 TIs for this application are of video collection type. The Type 2 TIs are the same pre-processing TI as Type 2 in application I. Type 3 TI is a frame extraction type that transforms video stream to images based on an extraction rate. For slow-moving pedestrians, the extraction rate is low which reduces the computation intensity of the task. For Type 4 TI, we use a pre-trained object detector called YOLO [30] which determines if the object of interest is in the frame, from a detectable object pool. The task function, techniques and commonly-used algorithms are summarised in Table II for this application type.

   If the Type 4 TI finds unknown objects, the images can be transmitted to back-end servers, which can thereby update the inference model of local detectors, but the global knowledge and cloud involvement are not in the scope of the applications we are defining for local detection.

We use the linear model described in [18] to determine the memory usage for video streaming. For the case of video streaming the memory usage ranges between 110-220 MB. The data size for each frame is given in the range of 2.7-33.7 KB for five different video resolutions (1920 * 1080, 1280 * 960, 960 * 720, 640 * 480, 320 * 240). For the object detection application, we ran pre-trained YOLOv3 [30] model on an Linux OS system with 8GB RAM and i7-6500U running at 2.50 GHz and got a processing latency of 7.417 seconds. The detection has 16-18% of CPU usage. We also ran Tiny YOLOv3 and got a processing latency of 0.31277 seconds with lower confidence scores. The detection uses 4-5% of CPU and can be used on resource constrained on-board units for non-safety related applications that can afford lower accuracy.

B. Network Topology

The network topology consists of moving nodes that halt at an intersection and the roadside unit (RSU) that receives application requests from clients. The cluster is initiated by selecting nodes based on their mobility pattern and resource availability. This information of vehicles willing to lease their resources is stored and updated at the RSU. A vehicle cluster is initiated by detecting a density of $\mathbb{I}$ nodes subscribed to provide their onboard computing and camera resources. The RSU represented as $\mathbb{I}_{\mathbb{RSU}}$ collects mobility and resource information from the $\mathbb{I}$ subscribed nodes. The mobility of each node is presented as the CCP, represented as p_I, which is the probability of the node staying at a particular road segment in a time interval from $[t_1, t_2]$. We also derive the communication link probability between two vehicles as the joint CCP for two vehicles to communicate over a period of time $[t_1, t_2]$.

The selected cluster of vehicles is represented as a directed graph $G(V, E)$. Each node of the graph $i \in V$ has K resource types. The available processing capacity, for each resource type k, is represented as $C_k(i)$. Each node i has a probability of staying on a road segment for a time period $[t_1, t_2]$, represented as $P_{(t_1,t_2)}$. The CCP of the RSU is equal to 1 as it is stationary and is always available from the viewpoint of mobility. The available link capacity between two nodes i_1 and i_2 is represented as $B(i_1, i_2)$ Kb/s. The joint probability between two nodes i_1 and i_2 depicts the probability of both nodes to stay together in a road segment for a period of time $[t_1, t_2]$ and is represented as $P_{t_1,t_2}(i_1, i_2)$. This is crucial for placing TIs that depend on other TIs for input data for task completion.

C. Distributed Service Model

The service model is composed of tasks, denoted as s_p, each with a different processing function or type, represented as p. Due to the limited resource capacity in each vehicle node, a service is composed in a distributed manner as a linear chain of tasks. Due to the dynamic nature of the vehicular network, each task can be scaled to multiple TIs, represented as s_{pj} where p represents the type of each TI and j represents the number of TIs. The number of TIs for each task s_p is N_{s_p} and the maximum number of allowed TIs for each type is given as $N_{s_p^{\max}}$. The objective of scaling tasks to instances is to increase the robustness in the service model, especially because of the link and node failure due to the wireless connectivity and vehicle mobility. The resource requirement of type k for each task type p is represented as D_{kp}, where $k \in \{1, 2, 3, 4\}$ for CPU, memory, GPU and video camera. The incoming flow from task types s_{p_1} to s_{p_2} is given as $F(s_{p_1}, s_{p_2})$.

The objective of this optimization is to find nodes that have a higher probability of staying together over a period of time. We then place two services of varying chain lengths (3 and 4), as described above. We model the mobile infrastructural resource constraints and the constraints required for placing a flexible and scalable service on the infrastructure. We then optimize resource usage in the service placement on the vehicular cluster through node and link cost, which is the sum of processing resources on vehicles and the communication cost for the data flow between the distributed tasks. The resource utilization is normalized to total available resources and weighted by the CCP to take into account the mobility of vehicle clusters.

V. Service scaling and placement constraints

We define the service scaling and placement problem as a constrained optimization problem. We first define the constraints for the distributed service scaling, which are given as:

*1) **Flow capacity constraint:*** The processing requirement for a flow from TI s_{p_1j} to s_{p_2j} is given as $C(F(s_{p_1j}, s_{p_2j}))$. The constraint 1 ensures that each TI has enough processing capacity for the incoming flow. The constraint is given as:

$$\forall i_2 \in \{1, \ldots, \mathbb{I}\};$$

$$\sum_{\forall p_1, j; p_2; p_1 \neq p_2} M(s_{p_2}, j, i_2) C(F(s_{p_1j}, s_{p_2j})) \leq C(s_{p_2j}) \tag{1}$$

where $C(s_{p_2j})$ is the available processing capacity at TI s_{p_2j}. Here, $M(s_{p_2}, j, i_2)$ is a binary mapping variable which is 1 when the TI s_{p_2j} is mapped to node i_2 and is 0 otherwise.

*2) **In-network processing constraint:*** Constraint 2 ensures that the flow is processed at each TI before being sent to the CN. To ensure that, we calculate the ratio of incoming to outgoing flow which should be equivalent to the data processing factor of each TI. The processing factor is given for each task type p and is given as α_p. The constraint is presented as:

$$\sum_{\forall p,j;} F(s_{pj}, s_{(p+1)j}) \alpha_p \leq F(s_{(p+1)j}, s_{(p+2)j}) \tag{2}$$

where $0 \leq \alpha_p \leq 1$. The data processing factor is 1 for forwarding nodes as it does not process the incoming data flow. The incoming flow from s_{pj} to $s_{(p+1)j}$ is given as $F(s_{pj}, s_{(p+1)j})$. The outgoing flow from $s_{(p+1)j}$ to $s_{(p+2)j}$ represents the flow that has to be processed at the TI $s_{(p+2)j}$.

*3) **Service Scaling constraint:*** The constraint 3 ensures that the TIs are scaled to the maximum number of TI specified for each task type p. This constraint also ensures that there is at least one TI for for each task type. This constraint is given as:

$$\forall p \quad N_{sp_{min}} \leq N_{sp} \leq N_{sp_{max}} \tag{3}$$

where N_{sp} is the number of TI of task type p. The maximum allowed TIs for the task type p is given as $N_{sp_{max}}$ and the minimum number of TIs for task type p is given as $N_{sp_{min}}$ which is set to 1 for our model.

A. Infrastructure constraints

The infrastructure constraints ensure the the node and link placement meets the resource constraints for the service placement. The infrastructure constraints are given as:

*1) **Node Resource constraint**:* The resource requirement for a TI is represented as D_{pk} where p is the type of task and k is the resource type where $k = 1$ is CPU cycles requirement, $k = 2$ is memory capacity requirement, k = 3 is video camera resource requirement and k = 4 is the GPU availability. A decision variable $M(p, j, i)$ is used if TI s_{pj} is mapped to node i. The node resource capacity ensures that there is enough available capacity at a node to support a TI s_{pj}. The constraint is given as:

$$\forall i \in \{1, \ldots, \mathbb{I}\}, k \in \{1, \ldots, \mathbb{K}\}, \sum_{\forall p,j} M(p, j, i) D_{pj,k} \leq C_k(i) \quad (4)$$

where $C_k(i)$ is the available capacity at node i for resource k.

*2) **Bandwidth constraint**:* The bandwidth constraint ensures that the bandwidth requirement between two TIs, given as $F(s_{pj}, s_{(p+1)j})$, is less than the available bandwidth capacity over the entire path between two task instances s_{pj} and $s_{(p+1)j}$. The path between two nodes i_1 and i_n is a list of bandwidth of variable length. It stores available capacity over all forwarding nodes between i_1 and i_n, if there is no direct link between the two nodes and the available bandwidth link between the two nodes if they are directly connected. The bandwidth capacity of the path is represented as $path[B(i_1, i_2), \ldots, B(i_{n-1}, i_n)]$. We enable this to support multihop clusters for cases where nodes might not be connected through a direct path but are connected over multiple hops. The constraint is given as:

$$\begin{aligned} &\forall i_1 \in \{1, \ldots, \mathbb{I}\}; i_2 \in \{1, \ldots, \mathbb{I}\}; i_1 \neq i_2 \\ \sum_{\forall p_1, j_1; p_2, j_2; p_1 \neq p_2} &M(p_1, j_1, i_1) F(s_{pj}, s_{(p+1)j}) M(p+1, j_2, i_n) \\ &\leq \min(path[B(i_1, i_2), \ldots, B(i_{n-1}, i_n)]) \end{aligned} \quad (5)$$

B. Mobility modeling

Each vehicle node has a certain probability of choosing a road segment based on its historic mobility pattern. The mobility history for each vehicle node is stored in the RSU along with the time stamp. We aim to select vehicles with the highest probability of staying at the selected road segment which is RS_j in our case, depicted in Fig. 5. A transition probability matrix stores the mobility probability for different road segments for all the vehicles registered to lease their resources and participate in the crowdsourcing service. New vehicles registering for the first time are also added to the table.

For discovering participating vehicles for the service deployment, the RSU broadcasts probe messages for participation requests. If already registered vehicles with known transition probability responds, they are given a priority over newer vehicles that want to participate. The newest participants are given the least confidence score. The confidence score is not issued according to the performance of the deployed task. It is a simple measure of updating confidence score if the vehicles follows its historic mobility trajectory. The confidence score is updated as the number of times the vehicle followed a preferred trajectory, over the total number of trips registered by the vehicle. Once the RSU updates its participants list, the RSU then runs the community detection-based node selection algorithm on a group of participants with a confidence score above a pre-decided threshold value.

We consider each road segment to be a Markov state. The vehicle transitions in the Markov process when moving from one road segment to the next. The vehicles follow a Markov memory-less property, wherein the node transitions from state n to n+1 and is independent of state n-1. We record the transition of a vehicle from state RS_i to RS_j as the number of times a vehicle transition to segment RS_j given the vehicle was at RS_i in its previous state. The probability is given as:

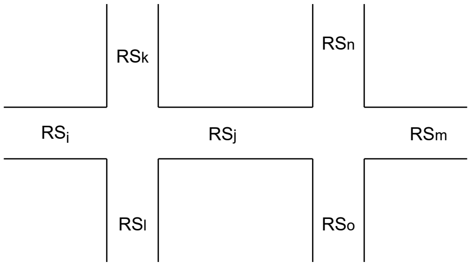


Fig. 5: Mobility modeling for selected road segments at intersections.

$$\begin{aligned} P_{t_1,t_2}(i)\{VS_{n+1} = RS_j | VS_n = RS_i\} \\ = \frac{\#(RS_i to RS_j)}{\#(RS_i)} \end{aligned} \quad (6)$$

where $[t_1, t_2]$ is the time interval selected based on the traffic state of the selected road segments. If the traffic is in a free-flow state, the time is chosen to be a 5-minute interval, and if there is queuing and the road is congested, the time interval is taken as 10 minutes. These values are inferred from our detailed experiments based on studying the predictability of vehicular flows at the Dublin intersection in our previous work [16]. To make the microscopic model more realistic, we use the macroscopic data from Transport Infrastructure Ireland Traffic Data[1] to calibrate the microscopic probabilities for each vehicle at the selected Dublin intersections, using the SUMO simulator.

C. Objective Function

We use node and link utilization cost as a measure to analyze the quality of placement of tasks on mobile vehicle nodes from the point of view of resource utilization. We aim to minimize the node and link utilization cost which is defined as:

1) **Obj1** Node Utilization Cost: is defined as the ratio of total used computational capacity to the total available computational capacity for the service placement. The ratio is weighted by the CCP of the node. This considers mobility of nodes rather than placing tasks on nodes with high processing capacity but with very low CCP to stay with the other vehicles in the cluster. The node utilization cost is given as:

$$\begin{aligned} \text{NodeCost}(i) = \sum_{\forall i,j,p;} \\ (1 - P_{(t_1,t_2)}(i)).(D_{pj,k}/C_k(i)).M(p, j, i) \end{aligned} \quad (7)$$

[1] https://trafficdata.tii.ie/publicmultinodemap.asp

where $P_{(t_1,t_2)}(i)$ is the CCP of the node with the TI s_{pj} placed on it. As the CCP of a node increases, the cost of placing the TI on that node decreases.

2) **Obj2** Link Utilization Cost: The link utilization cost is defined as the ratio of total used link capacity to the total available link capacity. This ratio is weighted by the CCP of the link. The CCP of the link defines the joint probability of two nodes to stay together during the time window $[t_1, t_2]$. The link utilization cost is given as:

$$\text{LinkCost}(i_1, i_2) = \sum_{\forall i_1, i_2; i_1 \neq i_2} (1 - P_{(t_1,t_2)}(i_1, i_2)). \tag{8}$$
$$(F(p_1, p_2)/\sum path[B(i_1, i_2)])\Big(M(p, j_1, i_1).M(p, j_2, i_2)\Big)$$

where $P_{(t_1,t_2)(i_1,i_2)}$ is the joint CCP of two nodes to stay together and $path[B(i_1, i_2)]$ is a list of available bandwidth on the path between two nodes i_1 and i_2 with placed TIs. The total bandwidth is summed over the path between nodes i_1 and i_2.

3) **Obj3** Chain length/hop count: The number of hops for the distributed service can have a significant impact on both communication overhead and service reliability. To keep the hops between two placed TIs to the minimum, we make sure to minimize the length of path between two nodes i_1 and i_2 with placed TIs. The network distance between two placed TIs is given as:

$$\forall i_1 \in \{1, \ldots, \mathbb{I}'_{(pj)((p+1)j)}\}; i_2 \in \{1, \ldots, \mathbb{I}'_{(pj)((p+1)j)}\}; i_1 \neq i_2$$
$$H(i_1, i_2) = \sum_{\forall p_1, p_2} M(p, j, i_1), M((p+1), j, i_2)$$
$$len(path[B(i_1, i_2)]), \tag{9}$$

where $H(i_1, i_2)$ is the hop count between two nodes i_1, i_2 with placed TIs. To minimize the service transmission latency, one can minimize the latency directly by calculating its value or minimizing the number of hops in the transmission path as a proxy indicator. The proposed model has been generalized for any application, with different latency requirements. To keep the model independent of the nature of the application, instead of latency, the number of hops in the transmission path is used as a proxy indicator. In our model we assume the cost per hop is the same for all services, steering clear from specific components of end-to-end latency like decoding time, encoding time, processing time at the entry or exit of the path.

The multi-objective function aims to minimize Obj1, Obj2, and Obj3. Each objective is weighted by λ_1, λ_2 and λ_3 such that λ_1 + λ_2 + λ_3 totals to 1 and each objective is weighted equally (but this can be changed to reflect operational requirements). The objective function is given as:

$$\min \sum_{\forall i_1, i_2; i_1 \neq i_2} \lambda_1 H(i_1, i_2) + \lambda_2 \text{LinkCost}(i_1, i_2) + \lambda_3 \text{NodeCost}(i_1) \tag{10}$$

where $H(i_1, i_2)$ is the hop count between two nodes i_1, i_2 with placed TIs.

VI. Mobility patterns of vehicles in highly congested urban areas

Even though it is known that vehicular congestion is a major problem in both urban sections of the cities and busier freeways, it is crucial to study the mobility patterns of vehicles to decide in which sections of the city can vehicular clusters be initiated for data collection and processing. Our service model requires vehicles to be closely spaced to each other to deploy the distributed data-dependent applications. Hence, we strictly focus on very slow-moving vehicles as a high-speed vehicle cluster would lead to more service failures and require many service re-configurations, increasing the management overhead. The traffic congestion is estimated using either the density or the occupancy of a road segment. Density is a measured, spatial quantity that represents the number of vehicles averaged over a spatial distance(per lane or mile), whereas occupancy is an observed value collected by detectors [31]. Occupancy is calculated as the percentage of time in which the vehicles are passing over the detector. We chose occupancy as a measure to see how closely spaced vehicles are and at what occupancy do the vehicles become slow-moving or reach a complete breakdown condition.

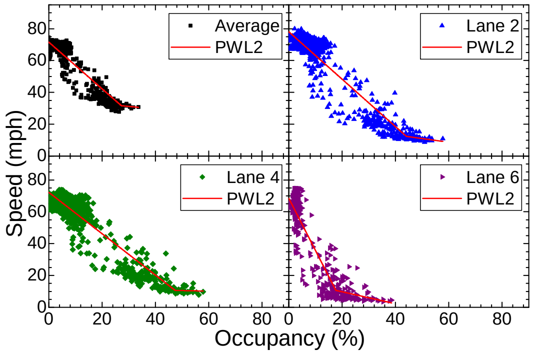


Fig. 6: Speed versus occupancy graph for a detector on the I-405 freeway for seven consecutive days. The data is taken from the Caltrans dataset [31].

The speed versus flow (number of vehicles passing a detector) and the occupancy versus flow graphs are standard traffic theory plots that are commonly used in transport research. However, we plot the speed versus occupancy graphs to understand the correlation between how closely spaced vehicles are and if there is a threshold speed at which vehicles come to a halt. We take the data from the California Department of Transport (Caltrans) dataset from detectors on the I-405 freeway, known to be one of the busiest freeways in California [31]. We plot the speed versus occupancy graphs from a detector on the freeway. As can be seen in Fig.6, the speed of vehicles tends to zero for lane 6 from 20% to 40% occupancy, whereas for lane 4, the speed gets at 10 mph after 50% occupancy. Similarly, for lane 2, the speed gets lower than 10 mph after 40 % occupancy. We do piece-wise fitting for the speed and can see a breaking point after which vehicular speeds stabilizes to slower speeds for all lanes. This highlights that there are enough vehicles that are closely spaced in busier road segments, moving at very slow speeds. This also helps in identifying where and when clusters can be initiated based on the available vehicular occupancy. However, the threshold speed of vehicles slowing down and congesting is different for different road sections.

VII. Heuristic-based Solution

Algorithm 1 Service Placement

```
Input: LG Linear type graph, Vehicle cluster graph
Input: ($UL_{Type_{i+1}}, LL_{Type_{i+1}}$): Upper and lower limit for number of Type i+1 TI
Output: Successful/Unsuccessful service placement
procedure ServiceMapping($LG, VC$)▷ The g.c.d. of a and b
    while $Type_1$ do                                  ▷ For all Type 1 instances
        for ($Type_1, CN$) ∈ TI_pairs do
            for i ∈ TI_pairs($Type_1, CN$) do          ▷ Ensures placement of full chain for each $Type_{i+1}$ instance
                TI_placement(i,VC,($UL_{Type_{i+1}}, LL_{Type_{i+1}}$))
                if placement is successful then
                    Success
procedure TI_placement(i,VC,(($UL_{Type_{i+1}}, LL_{Type_{i+1}}$)))
    while (i) do                                       ▷ for all available $Type_1$ TCI
        for ($Type_i$,$Type_{i+1}$) ∈ i do
            $node_1$ ← location of $Type_i$
            if $Type_{i+1}$ instances exist on VC then  ▷ To re-use instances
                $node_2$ ← List of location of $Type_{i+1}$
                for j $in\ node_2\_location\_list$ do
                    if resource at j ≥ resource required for $Type_{i+1}$ then
                        $pathtonode_2$ ← Get path from $Type_i$ to $Type_{i+1}$
                    $sortedpath$ ← sorts path based on path length
                    for k in $sortedpath$ do
                        if $required\_datarate \leq min(k\_path\_datarate(i,j))$ then
                            place $Type_{i+1}$ on $node_2$
                            return TI $Type_{i+1}$ placed
                            break
            else
                if length($node_2\_location\_list$) ≥ $UL_{Type_{i+1}}$ then
                    return Not enough resources on vehicle cluster
                else
                    $CN_{location}$ ← location of CN
                    paths ← weighted path from $Type_{i+1}$ to CN
                    $sorted\_paths$ ← sort paths from highest to lowest path weight
                    for i in $sorted\_paths$ do
                        if $required_{datarate} \leq min(i\_path\_datarate(x,y))$ then
                            while nodes available in i do
                                v ← next node on i
                                if resource on v ≥ resource required by $Type_{i+1}$ then
                                    Place $Type_{i+1}$ on v
                                    break
                                if no node available on path($Type_i$,CN) then
                                    return Unable to place TI on cluster
```

Due to the nature of vehicular networking, it is required to scale services and find efficient service placement in a very short time, compared to the time required to solve the full MIP. Consequently, we propose a node selection and service placement *heuristic* solution. We first leverage the historic mobility patterns of vehicles to select a group of vehicles that are more probable to stay together for a period of time using the principles of community detection.

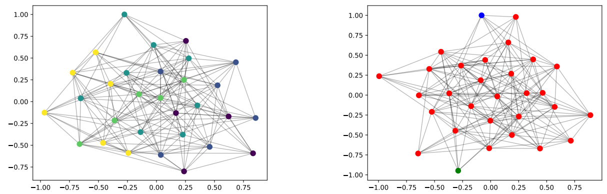
The mobility of vehicle nodes is constrained by the underlying road topology. We model the available vehicle cluster as a graph with their joint CCP as the edge weight for each edge, depicting how probable are two nodes to stay together in the next time segment. The use of community detection using mobility behavior helps in identifying the most connected vehicular nodes that play a crucial role in the successful deployment of our service model with data-dependent TIs. Due to the mobility of nodes, the services can fail because of link and node failures due to vehicles leaving the cluster. The identification of communities helps in reducing service failures and subsequent service reconfiguration, by selecting a group of closely connected vehicles. Even if nodes or links fail within the selected community, there will be alternate paths available within the community. The chances of a complete breakdown between any two nodes in the community are low.

A. Vehicular Node Selection

The first step of the service placement problem, requires selecting vehicular nodes that are more probable to stay together for a certain period of time. This is quantified using the CCP of the vehicles. At this stage, the scheme does not consider the available computation or communication capacity. We aim to find a sub-group of vehicles that have similar mobility patterns and follow a similar trajectory. We use community detection, which is the process of discovering cohesive groups or clusters in a network, to determine vehicular nodes that have better connectivity between them than the rest of the network. Using community detection algorithms, we partition the vehicular network graph into communities and, the biggest community is chosen for service placement. Due to the data-dependency between TIs in the service model, all TI's are promoted to be placed in the same community of nodes. Even if nodes leave or link fails within the selected community, there are alternate paths available in a well-connected community.

We analyze two community detection algorithms for selecting a vehicular cluster for the service placement. We use vehicular nodes and nodes interchangeably in the text We first use the modularity score-based Louvain algorithm [32] that initially starts with $|V|$ communities where each vehicular node is considered to be a community in the first iteration. Modularity is defined as the density of edges inside the community with respect to edges outside the community. In each iteration, every node is moved to its neighboring community and the gain in modularity is calculated. If the gain is positive, the node does not return to its previous community. The iterations of the heuristic stop when the modularity gain, between any two iterations, does not exceed a specified threshold value. The algorithm has the complexity of $O(VlogV)$ where V is the number of nodes in the graph. In our experiment, the modularity obtained in a cluster of 30 vehicles was 0.1428.

We also considered the hierarchical clustering-based Girvan and Newman algorithm [33] which derives a community tree or a dendrogram with a specified depth [34]. The connectivity of a community increases as the depth of the derived dendrogram increases. The method first removes the edge with the highest edge betweenness centrality. The edge betweenness centrality is the sum of the fraction of the shortest paths that cross the edge. Each iteration splits every existing community into two new communities. The disconnected sub-graphs undergo the same procedure until the entire graph is split into isolated nodes. The complexity of the algorithm is $O(E^2V)$, where E is the edges of the graph and V represents the nodes. The modularity score for the same graph using this method is 0.00186. We, therefore, prefer the Louvain method as it results in a higher modularity score, which is more useful in this context, and Louvain's computational complexity is also lower. Thus, the strongest selected community is the vehicle cluster that is used for the service placement problem.



(a) Louvain method

(b) Girvan and Newman method

Fig. 7: Using two different method for community/cluster detection.

B. Service placement heuristic

After the vehicle cluster is selected based on the CCP of nodes using the community detection method, we then place the TIs by utilizing a graph-based heuristic. We first give as input, 1) the selected vehicular cluster which is the strongest detected community, 2) the linear type graph to be placed and 3) the upper $(\text{UL}_{\text{Type}_i})$ and lower limit $(\text{LL}_{\text{Type}_i})$ for the number of TIs of each type to be placed. The LL for all the tasks is 1 as we want to make sure at least one TI of each type is placed. The UL for each TI is equal to the number of video sources or Type 1 TIs, ensuring each stream gets one processing TI, in case the available processing capacity at individual nodes is very low.

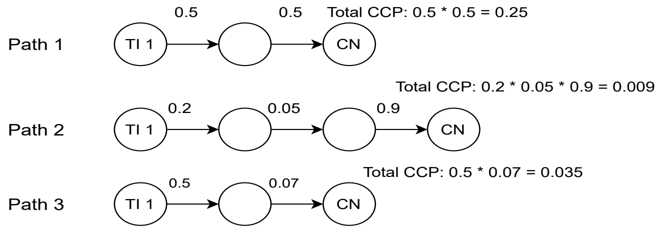

Path 1
TI 1
0.5
0.5
CN
Total CCP: 0.5 * 0.5 = 0.25
Path 2
TI 1
0.2
0.05
0.9
CN
Total CCP: 0.2 * 0.05 * 0.9 = 0.009
Path 3
TI 1
0.5
0.07
CN
Total CCP: 0.5 * 0.07 = 0.035

Fig. 8: Joint CCP-based path selection for service placement.

We modify a heuristic algorithm inspired by the work of [35], where VNFs are placed along the shortest path with the smallest bottleneck value. Instead of placing TIs on the shortest path, we consider the joint CCP of the path from a source TI (of Type 1) to the CN, as depicted in Fig.8. We then place TIs along the path with the highest joint CCP. As we intend to place a long chain of TIs along this path, choosing a longer chain increases the possibility of placing most TIs on the path to the CN. The heuristic may also randomly choose the shortest path, in terms of hop count, if the combined CCP of the path is the highest. From the three example paths shown in Fig.8, the heuristic will choose path 1 as it has the highest combined CCP, even though it is shorter than path 2. Choosing path 2 will result in placing TI on two nodes linked by very low CCP, 0.05 in this case. Path 3 is the same length as Path 1 but has lower joint CCP in comparison.

TABLE III: OPTIMALITY GAP PERCENTAGE FOR THE LINK UTILIZATION COST FOR DIFFERENT NUMBER OF TYPE 1 TIs

Number of Type 1 instances	Optimality gap (%)
1	0
2	0
3	11.03
4	13.05
5	10.997
6	17.83

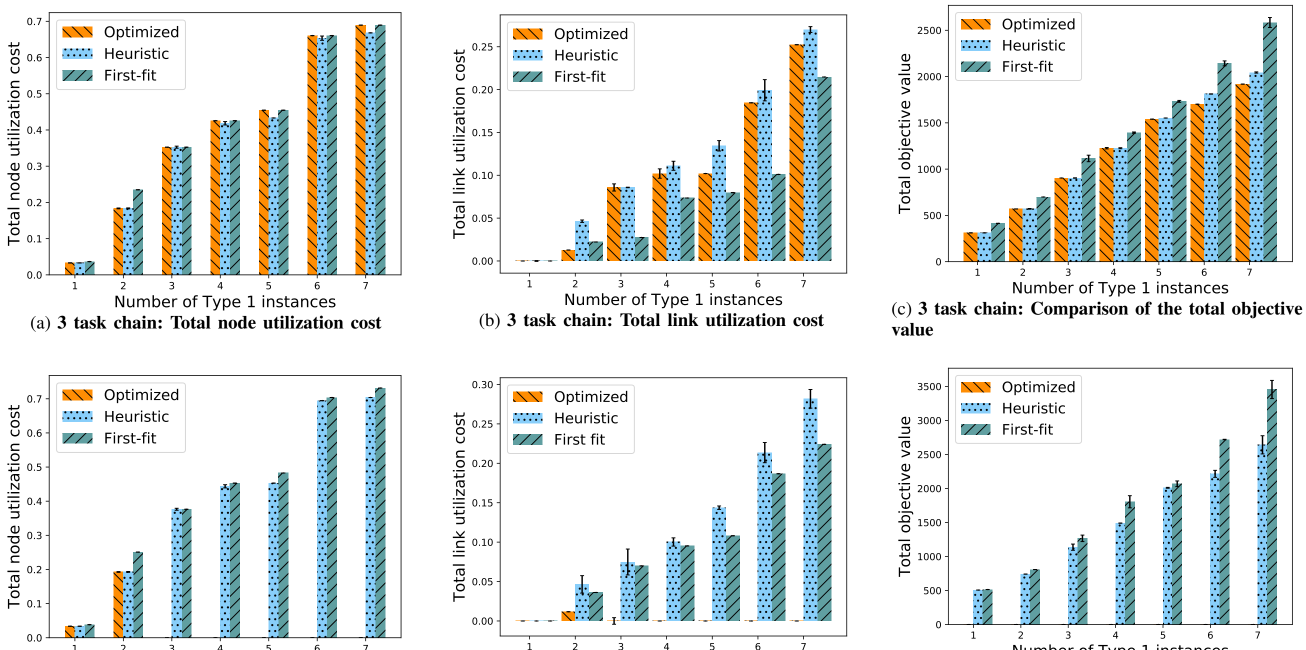
The heuristic is described as Algorithm 1. Similar to [35], we place TIs in a pairwise way, as the service model has dependent TIs. In our model, every TI of any type has a common endpoint as the CN. In line 3, we iterate over all TI pairs from Type 1 to the CN. In line 5, each TI pair is sent to the TI_PLACEMENT function along with the UL and LL for the TI. In line 11, the location of the Type 1 instance is detected. On line 12 it is checked if the next TI, of type $Type_{i+1}$, exists on the vehicle cluster. If it exists, say at node j, and the resource capacity at node j meets the capacity constraint for TI $Type_{i+1}$, all the paths from $Type_i$ to $Type_{i+1}$ are stored in the list $sortednode_2$. In line 18, all the available paths are iterated over and the bottleneck edge capacity for each path is compared to the required available capacity between the two TIs. If the constraint is met, $Type_{i+1}$ TI is reused for the flow. If $Type_{i+1}$ does not exist on the cluster, it is checked if the upper limit for the TI type is met (on line 24).

If the upper limit is not reached, all the paths are explored from Type$_i$ to CN of the cluster. All the paths are sorted based on the path weight, which in our case is the total CCP of the path. In line 31, all the paths are iterated over, and the bandwidth capacity requirement is checked for the path. If the bandwidth requirement is met and the resource capacity requirement for the node is met, then Type$_{i+1}$ is placed on the node v. If there are no more available nodes on the path to the CN, a failed placement is registered. Thus, this approach aims to send the collected data to the CN and tries to place processing TIs in-network when possible. An example of the service placement heuristic has been added to the github repository[2].

VIII. Evaluation

In this section, we evaluate the performance of the MIP, the proposed heuristic, and the first-fit approach through resource

[2] https://github.com/kannonical/graph-based-service-placement



(a) **3 task chain: Total node utilization cost**

(b) **3 task chain: Total link utilization cost**

(c) **3 task chain: Comparison of the total objective value**

(d) **4 task chain: Total node utilization cost**

(e) **4 task chain: Total link utilization cost**

(f) **4 task chain: Comparison of the total objective value**

Fig. 9: Total node utilization cost, total link utilization cost and total objective value for task chain of lengths 3 and 4.

utilization metrics for bandwidth (link) and processing (node). We also compare the MIP, the first-fit, and the heuristic for the total optimal value. We then evaluate the average chain length and the total instances that the heuristic scales to analyze the performance of the heuristic in terms of not overprovisioning resources. We then use total response time as a QoS measure and compare our placement approach to a static mobile-edge computing approach, which does not use task replication in the form of multiple instances of the same task. We also evaluate the performance of the selected vehicular cluster using centrality measures. We use vehicular mobility on a Fog-computing-based simulator, to study the evolution of the selected cluster through its lifetime.

We first use Gurobi, a standard MIP solver to solve the multiple objectives, constrained optimization problem. The configuration of the computing capacities ranges between three resource profiles: 1) Large node type: 5 CPUs, 500Mb disk, 6MB/s bandwidth; 2) Medium node type: 3 CPUs, 250Mb disk, 4MB/s bandwidth; and 3) Small node type: 2 CPUs, 100Mb disk, 2MB/s bandwidth. Vehicular OBUs may have higher computation capacity, but in our model, we assume that vehicles would lease only part of their resources, in return for some incentive, for over-the-top services. We used Veins-LTE to simulate the data collection application, described in section IV on page 5. VeinsLTE is a vehicular network simulator based on two simulators: Omnet++, an event-based network simulator, and SUMO, a road traffic simulator.

We find an optimal solution for placing both service types of different chain lengths on a selected vehicle cluster. We evaluate the solution for the node utilization cost, link utilization cost, and the total objective value for placing multiple applications of chain lengths 3 and 4. We vary the number of video instances from 1 to 7 to evaluate the scalability of the experiment. The applications are defined in Section III, where the 'data collection application' type is rich in data flow and has low processing requirements, whereas the 'object detection' application type is more compute-intensive and has less bandwidth requirement. The applications are of variable chain length. We use the first fit approach as the baseline approach. It sorts all the available paths from the data collecting TI to the CN and sorts them based on the highest available resource capacity. It then places TIs on all the available vehicle nodes on that path.

We first place two applications of the first type on the selected vehicular cluster. For the case of the 3 task chain, our heuristic gives better node utilization cost as compared to both optimal and first-fit solutions, as shown in Fig.9a. Our heuristic gives comparable link utilization cost in comparison to the optimal solution for 1-3 Type 1 TIs, but it becomes less efficient for a higher number of Type 1 TIs, as shown in Fig.9b. This is due to prioritizing paths with higher CCP which may result in selecting longer routes between dependable TIs.

For the total objective value, our heuristic performs as well as the optimal solution for the 1-5 Type 1 TIs, as shown in Fig.9c. For more number video TIs (Type 1), our heuristic under-performs when compared to the optimal solution. The optimality gap percentage has been summarised for the case in Table III. The worst-case optimality gap is 17.83% for the case of 7 Type 1 TIs. The first-fit algorithm performs poorly for any number of Type 1 TI, irrespective of the scale. In terms of execution time, the MIP solves the problem in 1 second for 1 Type TIs, whereas the heuristic solution takes 0.1154 seconds. Each run is being performed on the same hardware, an Intel i7-6500U running at 2.50 GHz, not optimised for performance.

For the case of 7 Type 1 TIs, the MIP solution takes 15.60 seconds whereas the heuristic solution takes 1.235 seconds.

For the second case, we place both applications of chain length 3 and 4 TIs for both services. The MIP solver fails to give a solution in a reasonable time for applications with a longer chain length. We get a solution from the solver in seconds for 1-2 Type 1 TIs. But as the number of Type 1 TIs increases, the solver does not converge to a solution even after hours. We, therefore, compare our heuristic solution to the baseline approach. Our heuristic performs better than the baseline approach for any number of Type 1 TIs (from 1 to 7), as shown in Fig.9d. The baseline approach outperforms the heuristics solution for the link utilization cost for the second case, as shown in Fig.9e. This is due to choosing paths that have higher joint CCP, to increase the robustness of the service placement. This results in more bandwidth utilization as a tradeoff to selecting more robust paths. Our approach outperforms in minimizing the total objective value as compared to the first-fit approach, as depicted in Fig.9f. The baseline approach fails in minimizing the total objective for the higher number of Type 1 TIs.

As our heuristic solution does not select the shortest path but the most reliable path, it might select very long paths with multiple hops between the Type 1 and the CN TIs. We present both the aggregate hop count and the total number of scaled processing TIs (of Type 2, 3 and 4) corresponding to the number of Type 1 TI, presented in Fig.10a. For any number of TIs, the hop count of all the paths between Type 1 TI and CN ranges between 5 and 6. The total number of processing TIs for each placement is also plotted in the same figure and it ranges from 4 to 15.

To compare the number of scaled processing TI's, we run the optimization problem with the objective of minimizing the number of processing TIs, to analyze the least number of processing TIs required for meeting the application demands. We calculate the minimum number of Type 2, 3, and 4 TIs required for a successful service placement without any TI being rejected a placement on the vehicular cluster in Fig.10b. We compare this to the number of TIs scaled by our heuristic, plotted in Fig.10c. As shown, our heuristic scales are close to the minimum number of required TIs. It places 2-4 more TIs in comparison to the least number of required TIs. But the heuristic chooses more reliably connected nodes to place the TIs.

A. Comparison of placement techniques in terms of service time

We have evaluated our placement approach from a resource utilization point of view. We now look at a QoS-based metric to compare our approach to a mobile-edge computing-based solution. The service demands are still generated by moving vehicles, but the mobile-edge computing approach places all the TIs on static edge servers. For the real-time performance of the vehicle cluster, we use a fog computing simulator called Yet Another Fog Simulator (YAFS) [36] for modeling the mobility and estimating the real-time performance of the selected cluster. YAFS is a python-based discrete event simulator that supports resource allocation policies in fog, edge, and cloud computing. The simulator has a distributed data flow application model that could be easily adapted to our use case. The simulation provides dynamic service selection, placement, and replacement of services that we have customized for our requirements. The support of mobility of nodes, which can also be treated as processing nodes makes the simulator a good fit for our case.

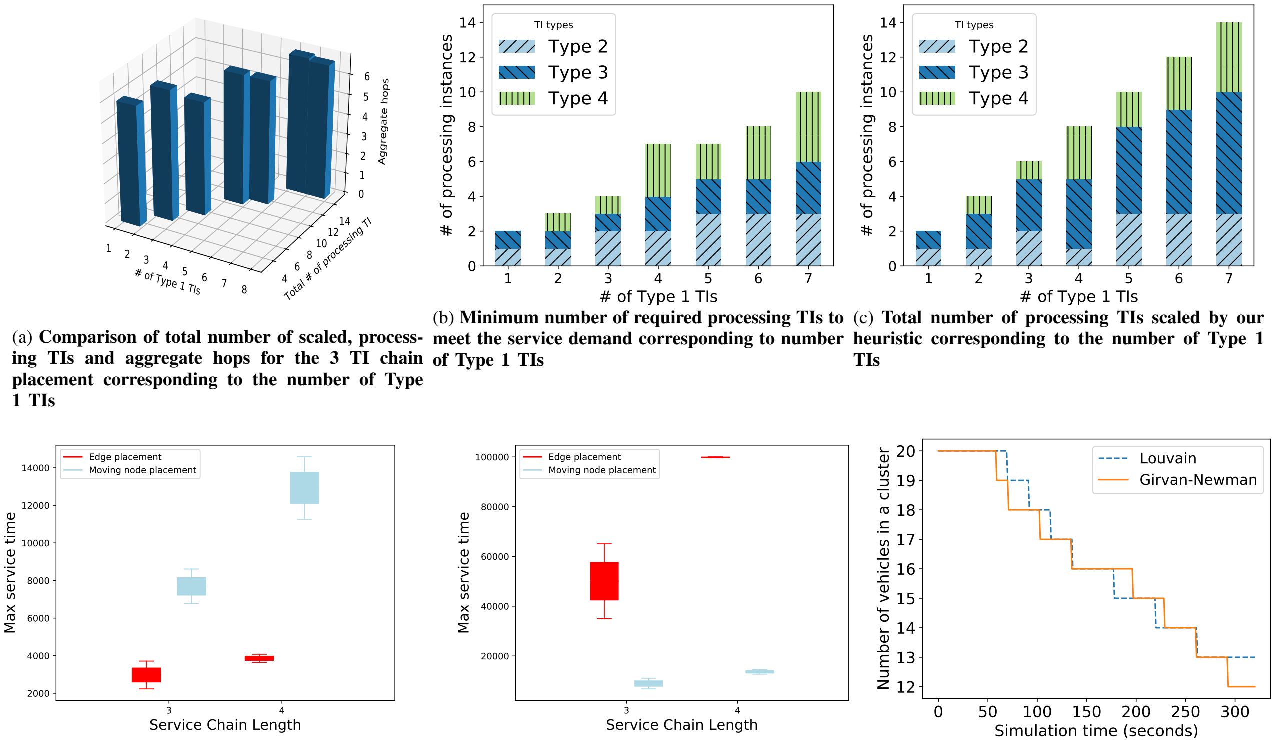
We consider the service time as the total time it takes for a service to execute, including both processing and link latency. We observe the minimum and maximum service time for the two services of different chain lengths for the mobile-edge placement versus our approach of placing multiple TIs on a moving vehicle cluster. We observe that the minimum service time is significantly less for the cloud placement, in comparison to our approach, in Fig.10d. Our approach places multiple TIs on different vehicles, thus the delay in the execution of one TI can result in a significant delay in service execution time. For the case of maximum service time, as can be seen in Fig.10e, the mobile-edge placement is significantly high. This is because of the delay in sending all the collected data from moving vehicles to the edge server or RSU. The service time is also higher for the chain length of 4 for the mobile edge placement approach. In comparison, our approach approximately takes the same time for a chain length of 3 or 4 in the worst-case scenario as the minimum service time in Fig.10d. Thus, even if an optimal placement is not achieved, on average our approach performs better in terms of service time, in comparison to the mobile-edge placement approach.

B. Evaluation of the selected cluster over time

We also analyze the performance of the selected cluster by evaluating the number of nodes in the cluster that stay together over a period of time. For the two community detection-based node selection approaches, out of the 20 selected nodes, 12 to 14 nodes make it till the end of the simulation, as depicted in Fig.10f.

We then evaluate the quality of the selected cluster in terms of the nodes that stay till the end of the simulation by using a resilience score. We use the betweenness centrality as a measure to check the importance of a node, in terms of connectivity in the graph. The *betweenness centrality* calculates the shortest weighted path between every pair of nodes in a graph. Each node gets a betweenness centrality score (BCS) based on the number of shortest paths that pass through the node. The resilience score is calculated as the total BCS for all the nodes that made it till the end of the service time upon the total BCS of all the nodes in the selected cluster. The higher resilience score shows that nodes with higher BCS stayed with the cluster, thereby reducing the need for rerouting flows or re-configuring service chains due to the absence of a forwarding node or a path between two data-dependent TIs. We evaluate the communities detected for two community sizes, of 15 and 30, using the Girvan-Newman and the Louvain approaches. We observe a higher resilience score for the Louvain approach for communities of either size, as depicted in Fig.11.

We observe a higher resilience score for the bigger cluster, of 30 nodes, as we observe more number of nodes with higher



(a) **Comparison of total number of scaled, processing TIs and aggregate hops for the 3 TI chain placement corresponding to the number of Type 1 TIs**

(b) **Minimum number of required processing TIs to meet the service demand corresponding to number of Type 1 TIs**

(c) **Total number of processing TIs scaled by our heuristic corresponding to the number of Type 1 TIs**

(d) **Minimum service time for different chain length**

(e) **Maximum service time for different chain length**

(f) **Cluster state through the simulation**

Fig. 10: Different performance metrics to evaluate the performance of the proposed heuristics, service time comparison for the proposed approach with edge placement and evaluation of the cluster state throughout the states of the simulation.

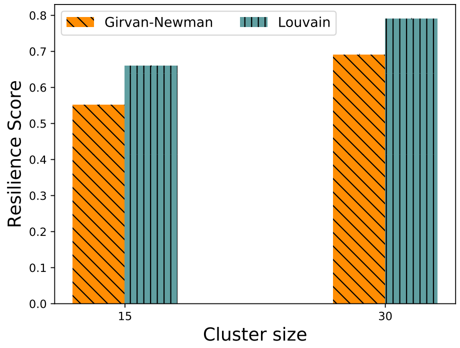


Fig. 11: Calculated betweenness centrality score for the two community-detection approaches.

BCS stay with the cluster till the end of the simulation time. However, this might not always be the case. A bigger cluster may not always end up being more resilient than a smaller cluster. The resilience score depends on the BCS, which is based on the importance or the role of a node in the graph in terms of the flow of communication. For example, the BCS of any vertex in a complete graph is zero since no vertex lies on the shortest path, as every node is connected to the other by a unique edge.

IX. Conclusion and Future Works

The paper aims to solve the problem of placing video collection and object-detection applications on moving vehicle clusters. The first application executes pre-processing tasks on the vehicle cluster whereas the second application executes a pre-trained local object-detection service, which is computation-intensive. We then model the problem as a multi-objective, constrained optimization problem. We introduce a vehicular node selection and service placement problem with the novelty of placing scalable and distributed services on mobile infrastructure.

We also evaluate the performance of the service placement heuristic using other resource utilization measures like the number of scaled TIs and the average hop-count for placing the distributed services. We then consider a QoS-level parameter called service time to analyze how our approach performs compare to a mobile-edge placement approach. We also emulate the service placement using a Fog simulator. We analyze how the node selection approach performs in terms of the life of the selected cluster. We introduce a betweenness centrality-based resilience score to evaluate the performance of the chosen cluster, in terms of the quality of nodes that make it to the end of the execution time.

Whilst the MIP delivers an optimal solution it requires significant computational resources; on the other hand, the heuristic delivers near optimal solutions using less resources, making it viable for deployment in a real system. Our approach also outperforms the baseline first-fit solution, because the mobility-aware strategy ensures that the cluster cohesion is higher, increasing the resilience of the system. We also compared our vehicular fog computing approach to edge computing-based placement. Our placement technique results in better worst-case performance, with much lower maximum service time that is a measure of the time taken in service execution, including both processing and link latency.

In future work, we plan to extend our theoretical treatment of mobile service placement on vehicles in urban traffic by collecting data from a smart city testbed and analysing how well our algorithm performs in practice. We also intend to consider service migration and not just the initial service placement problem. We intend to add the role of multiple RSUs to manage the life-cycle of both the CN and the vehicle cluster.

References

[1] M. M. Rathore, A. Paul, S. Rho, M. Khan, S. Vimal, and S. A. Shah, "Smart traffic control: Identifying driving-violations using fog devices with vehicular cameras in smart cities," *Sustainable Cities and Society*, vol. 71, p. 102986, 2021.

[2] Z. Ning, J. Huang, and X. Wang, "Vehicular fog computing: Enabling real-time traffic management for smart cities," *IEEE Wireless Communications*, vol. 26, no. 1, pp. 87–93, 2019.

[3] S. S. Lee and S. Lee, "Resource allocation for vehicular fog computing using reinforcement learning combined with heuristic information," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10450–10464, 2020.

[4] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7944–7956, 2019.

[5] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[6] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All one needs to know about fog computing and related edge computing paradigms: A complete survey," *Journal of Systems Architecture*, vol. 98, pp. 289 – 330, 2019.

[7] R. Yadav, W. Zhang, O. Kaiwartya, H. Song, and S. Yu, "Energy-latency tradeoff for dynamic computation offloading in vehicular fog computing," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 14198–14211, 2020.

[8] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylae-Jaeaeski, "Fog following me: Latency and quality balanced task allocation in vehicular fog computing," in *2018 15th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–9, 2018.

[9] A. Thakur and R. Malekian, "Fog computing for detecting vehicular congestion, an internet of vehicles based approach: A review," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 2, pp. 8–16, 2019.

[10] "Teslas new hw3 self-driving computer its a beast." https://cleantechnica.com/2019/06/15/teslas-new-hw3-self-driving-computer-its-a-beast-cleantechnica-deep-dive/. Accessed: 2021-10-19.

[11] I. W. Ho, S. C. Chau, E. R. Magsino, and K. Jia, "Efficient 3d road map data exchange for intelligent vehicles in vehicular fog networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 3, pp. 3151–3165, 2020.

[12] H. Du, S. Leng, F. Wu, X. Chen, and S. Mao, "A new vehicular fog computing architecture for cooperative sensing of autonomous driving," *IEEE Access*, vol. 8, pp. 10997–11006, 2020.

[13] Z. Zhou, H. Liao, X. Wang, S. Mumtaz, and J. Rodriguez, "When vehicular fog computing meets autonomous driving: Computational resource management and task offloading," *IEEE Network*, vol. 34, no. 6, pp. 70–76, 2020.

[14] S. Ge, M. Cheng, and X. Zhou, "Interference aware service migration in vehicular fog computing," *IEEE Access*, vol. 8, pp. 84272–84281, 2020.

[15] C. M. Kanaka Sri Shalini, Y. M. Roopa, and J. S. Devi, "Fog computing for smart cities," in *2019 International Conference on Communication and Electronics Systems (ICCES)*, pp. 912–916, 2019.

[16] K. Sharma, B. Butler, and B. Jennings, "Scaling and placing distributed services on vehicle clusters in urban environments," *IEEE Transactions on Services Computing, to appear*, 2022.

[17] J. Zhang, H. Guo, J. Liu, and Y. Zhang, "Task offloading in vehicular edge computing networks: A load-balancing solution," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 2, pp. 2092–2104, 2020.

[18] C. Zhu, J. Tao, G. Pastor, Y. Xiao, Y. Ji, Q. Zhou, Y. Li, and A. Ylä-Jääski, "Folo: Latency and quality optimized task allocation in vehicular fog computing," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4150–4161, 2019.

[19] F. Lin, Y. Zhou, G. Pau, and M. Collotta, "Optimization-oriented resource allocation management for vehicular fog computing," *IEEE Access*, vol. 6, pp. 69294–69303, 2018.

[20] Z. Liu, P. Dai, H. Xing, Z. Yu, and W. Zhang, "A distributed algorithm for task offloading in vehicular networks with hybrid fog/cloud computing," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–14, 2021.

[21] J. Liang, J. Zhang, V. C. Leung, and X. Wu, "Distributed information exchange with low latency for decision making in vehicular fog computing," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[22] G. Qiao, S. Leng, K. Zhang, and Y. He, "Collaborative task offloading in vehicular edge multi-access networks," *IEEE Communications Magazine*, vol. 56, no. 8, pp. 48–54, 2018.

[23] K. Sharma, B. Butler, B. Jennings, J. Kennedy, and R. Loomba, "Optimizing the placement of data collection services on vehicle clusters," in *2018 IEEE 29th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC)*, pp. 1800–1806, 2018.

[24] M. . Kuran, A. Carneiro Viana, L. Iannone, D. Kofman, G. Mermoud, and J. P. Vasseur, "A smart parking lot management system for scheduling the recharging of electric vehicles," *IEEE Transactions on Smart Grid*, vol. 6, no. 6, pp. 2942–2953, 2015.

[25] S. Iqbal, A. W. Malik, A. U. Rahman, and R. M. Noor, "Blockchain-based reputation management for task offloading in micro-level vehicular fog network," *IEEE Access*, vol. 8, pp. 52968–52980, 2020.

[26] Y. Zhao and C. H. Liu, "Social-aware incentive mechanism for vehicular crowdsensing by deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–12, 2020.

[27] G. Grassi, K. Jamieson, P. Bahl, and G. Pau, "Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[28] C. Zhu, A. Mehrabi, Y. Xiao, and Y. Wen, "Crowdparking: Crowdsourcing based parking navigation in autonomous driving era," in *2019 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, pp. 1401–1405, 2019.

[29] C. Zhu, Y.-H. Chiang, Y. Xiao, and Y. Ji, "Flexsensing: A qoi and latency-aware task allocation scheme for vehicle-based visual crowdsourcing via deep q-network," *IEEE Internet of Things Journal*, vol. 8, no. 9, pp. 7625–7637, 2021.

[30] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[31] "An Introduction to the Calfornia Department of Transportation Performance Measurement System (PeMS)." https://pems.dot.ca.gov/, Feb 2020. [Online; accessed 05-April-2022].

[32] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, p. P10008, Oct 2008.

[33] M. E. J. Newman, "Analysis of weighted networks," *Phys. Rev. E*, vol. 70, p. 056131, Nov 2004.

[34] I. Lera, C. Guerrero, and C. Juiz, "Availability-aware service placement policy in fog computing based on graph partitions," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 3641–3651, 2019.

[35] S. Dräxler and H. Karl, "Specification, composition, and placement of network services with flexible structures," *International Journal of Network Management*, vol. 27, no. 2, p. e1963, 2017. e1963 nem.1963.

[36] I. Lera, C. Guerrero, and C. Juiz, "Yafs: A simulator for iot scenarios in fog computing," *IEEE Access*, vol. 7, pp. 91745–91758, 2019.