Vehicular and Edge Computing for Emerging Connected and Autonomous Vehicle Applications

Sabur Baidya[1], Yu-Jen Ku[1], Hengyu Zhao[2], Jishen Zhao[2], Sujit Dey[1]
[1] Department of Electrical and Computer Engineering, [2] Department of Computer Science and Engineering
University of California, San Diego, CA, USA
Email :{sbaidya, yuku, h6zhao, jzhao, dey}@ucsd.edu

***Abstract*—Emerging connected and autonomous vehicles involve complex applications requiring not only optimal computing resource allocations but also efficient computing architectures. In this paper, we unfold the critical performance metrics required for emerging vehicular computing applications and show with preliminary experimental results, how optimal choices can be made to satisfy the static and dynamic computing requirements in terms of the performance metrics. We also discuss the feasibility of edge computing architectures for vehicular computing and show tradeoffs for different offloading strategies. The paper shows directions for light weight, high performance and low power computing paradigms, architectures and design-space exploration tools to satisfy evolving applications and requirements for connected and autonomous vehicles.**

Keywords—Vehicular computing, autonomous vehicles, edge computing, task partitioning

I. Introduction

The recent evolution of vehicles has significantly magnified the complexity of vehicular computing and control systems. Previous generation vehicles mainly relied on sensors, actuators, and microcontrollers for independent tasks. However, with the advent of connected and autonomous vehicles (CAV) [1, 2], the enormous volume of real-time data made available by modern sensors, e.g., cameras, radars, LiDARs and other sensing devices, has enabled a varied range of advanced vehicular applications e.g., object detection and tracking, navigation, path planning, localization, autonomous or assisted controls as shown in Fig. 1. Some of these applications are highly compute-intensive, e.g. involves complex and heavy machine learning algorithms, while some are much more latency-sensitive because of the need for low closed loop control delay whereas some have high reliability requirements.

The performance demands are more stringent with the increasing degree of autonomy from level-2 to level-5 autonomous vehicles [3]. Additionally, the control decisions in the connected and autonomous vehicles often involve complex sensor fusion algorithms and thus enhances the challenge in multiscale computations and synchronizations.

In this paper, we look into the essential performance metrics required to be satisfied to support the emerging applications, e.g. LiDAR perceptions or object detections from camera frames, which help in making control decisions in the CAVs. We investigate the optimal scheduling of the complex computation algorithms for a given computing architecture and also show directions to find optimal computing

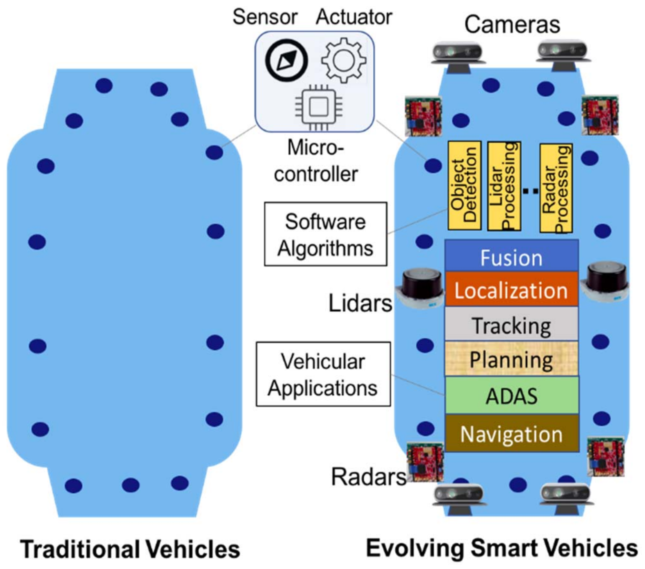

Fig. 1. Evolution of vehicular sensors, computing tasks and applications from traditional to connected and autonomous vehicles.

architectures for given application requirements. We have performed preliminary experiments with real-world computing hardware and software, and shown the tradeoffs for the optimal choices in the aforementioned scenarios. We also present the feasibility of using the edge computing paradigm [4] for vehicular computations which extends the computing capabilities of vehicles with more powerful computing servers installed at road-side units (RSU), however with additional overhead of wireless communications. We show that efficient task partitioning and computation offloading can provide optimal performances in vehicular applications.

In light of future autonomous and connected vehicles, herein, we show the feasibility of efficient, lightweight, low-cost computing configurations and resource allocations that can satisfy the computing demands without compromising the safety, accuracy, and other performance requirements.

II. Vehicular Computing Architectures

Computing architectures in emerging smart vehicles are rapidly evolving in recent years. In the following, we discuss the state-of-the-art approaches to computing system design of autonomous vehicles, and how emergent needs and connectivity will pose challenges and opportunities for rethinking future vehicular computing.

A. Autonomous Vehicles

Most modern autonomous vehicle developers and platform suppliers, such as Waymo, Tesla, Nvidia, Mobileys, and Uber, strive to build self-sufficient driving platforms, i.e., a platform that (i) is equipped with all of the necessary driving operations on a self-contained system and (ii) relies on itself to perform all of the driving tasks and ensure safety. The driving computing system is an especially critical component in high (level-4) and full (level-5) automation autonomous vehicles (AVs) where the computing system will act as the driver, even in the face of emergency. For instance, a level-4 AV needs to drive itself almost all the time without any human input, except for unmapped areas or during severe weather. Therefore, we need to carefully design both software and hardware of AV computing systems. While low-level AVs still adopt embedded computing systems, modern high-automation AV systems adopt server-grade heterogeneous hardware and software. For example, state-of-the-art level-4 AVs employ Intel Xeon CPU, Nvidia Volta GPU, terabytes of SSDs, FPGAs, and Ubuntu operating systems to accommodate the sophisticated computation and storage demand [5]. However, given the limited area, power budget, and system redundancy inside vehicles, as well as the dynamic environment and computation demand changes, it is important to manage the hardware and software resources the same way as traditional servers. For example, it is difficult to minimize the latency of all computing tasks due to the interdependency of executing software modules [5]. As such, it is critical to rethink computing system design to meet the unique demands and constraints introduced by AV applications to be further discussed in this paper.

B. Connected Vehicles

With the development of C-V2X communication technologies [6], connected vehicles will have high throughput and low latency connections with other vehicles or road-side infrastructures. This will provide an opportunity for vehicular edge computing (VEC) [4], which enables a vehicle to improve the performance of vehicular applications by utilizing edge computing resources from neighboring vehicles [7, 8] or RSUs [4, 7, 9-11]. Task offloading and computing resource sharing methods are being investigated to improve the latency [4, 7-9], power consumption [10], or both [11]. For example, [8] investigates the task offloading and scheduling methods among multiple neighboring vehicles for optimizing the latency performance. However, how to efficiently partition and offload the computation tasks according to the current channel conditions and the corresponding local and edge computing architectures is still an open topic. In Section V, based on our preliminary study [12], we will present the possible framework for efficient task partitioning and offloading, and provide end-to-end delay analysis between different offloading strategies.

III. Performance Metrics

With the degree of autonomy of CAV starting to increase to a higher level, it becomes crucial to ensure the performance of vehicular applications, as they will have increasing responsibility for driving experience and safety. In this section, we introduce different performance metrics that the computing architecture of the evolving smart vehicles should consider.

A. Latency/Safety

Safety is one of the most critical metrics when evaluating CAVs. In the automotive industry, safety is categorized to (a) functional safety and (b) nominal safety. We will discuss functional safety in Section III-C. Nominal safety enforces that CAVs always make correct and timely decisions. To make correct driving decisions, the developers need to design sound algorithms to perform various driving tasks. For example, Mobileye and Nvidia develop their own safety models, the responsibility-sensitive safety model [13] and the safety-force-field model [14], respectively. The two safety models focus on how to design software algorithms to ensure correct decision making in various driving scenarios. However, software correctness is insufficient to guarantee nominal safety, because we still need to ensure that CAVs interpret various driving scenarios (including emergencies) and make driving decisions in a timely manner to avoid collisions. For example, without a timely traffic interpretation and driving decision making, CAVs may not have sufficient time to apply a hard-brake in an emergency -- that will be extremely dangerous. The timely decision-making requirement motivates us to investigate how we can improve the vehicular computing system design to optimize the system latency in terms of response time to various driving scenarios. One way of estimating the latency requirement to ensure nominal safety is to consider a threshold based on LiDAR sampling cycle, which is typically 100ms. Many modern high-automation vehicles rely on LiDAR for perception to understand the instant driving scenario, which will guide the driving decision making. If the computing system cannot process a LiDAR input within 100ms, the next cycle of data will arrive, further slowing down perception by an accumulation of perception latency.

B. Accuracy

Accuracy of vehicular computing can be defined in terms of the types of computation processes, e.g., object detection, localization, navigation, path planning, etc. Usually, the accuracy of a computing algorithm is measured with respect to a ground truth. For example, the accuracy of an object detection can be measured in terms of intersection over union (IoU) of the matching objects' bounding boxes detected in the algorithm vs. those corresponding objects in the ground truth. Now, depending on the complexity of algorithms, the accuracy usually is inversely related to the speed of the algorithm. As an example, for object detection, we can use several algorithms, e.g. Tiny yolo v3 [15] or SSD MobileNet v2 [16], which are in the increasing order of speed but decreasing accuracy of object detection as shown in Fig 2.

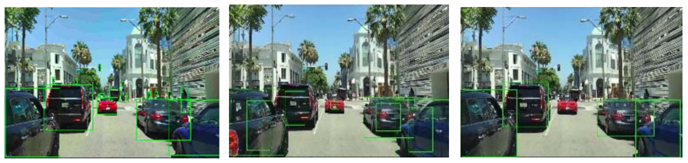

Fig. 2. Varying bounding boxes of detected objects in (a) Ground truth (b) Tiny yolo v3, and (c) SSD MobileNet v2 show varying accuracy

C. Reliability

The AV reliability, also called functional safety, is quantified by ISO26262 [17]. It ensures that CAVs always maintain system integrity, to be free or immune of software bugs and hardware failures. In current designs, we list three opening questions in terms of CAV reliability: (1) what if the system crash happens when the vehicle is operating, as current design does not consider this point because the human safety driver will take over the car if any hazards happen; (2) how to ensure the inaccurate deep learning models do not affect reliability, because even state-of-the-art deep learning models cannot promise a 100% accuracy; (3) how to monitor the CAV system in real-time so that it is practical to prevent it from system crashes. In fact, to ensure CAVs' reliability, it is necessary for us to take both hardware and software designs into account.

D. *Power Consumption*

Power consumption is another critical metric that hampers CAVs' performance, because high power consumption not only limits the endurance of electric vehicles, but also results in heat dissipation issues that increases the probability of hardware failures. In terms of high-level AVs, such as level-4 or even level-5 AVs, the computation requirement is huge, so they adopt server-grade processors. Such a computing platform consumes significant energy, e.g., Nvidia's AV hardware, called Pegasus, consumes 500 watts per hour [18].

In this paper, we primarily focus on the latency and power consumptions of the vehicular applications.

IV. In-Vehicle Computing

As vehicular applications are becoming increasingly compute-intensive, it is essential to manage in-vehicle computing resources carefully, so as to ensure the required CAV application performances. Although such performance requirements can be potentially met by deploying in-vehicle very powerful computers, the corresponding cost and power consumption may be prohibitive. We will first discuss how to allocate available in-vehicle computing resources according to the dependency graph of the applications, such that specific performances can be guaranteed. Next, we will discuss how to select an optimal computing architecture for a vehicle, given its required application capabilities and performances.

A. *How to optimally allocate computing resources?*

With finite hardware resources and interdependently executing software tasks in CAV computing systems, it is impossible to simultaneously minimize the latency of all computation tasks to maximize nominal safety. Therefore, we need to determine the computation hardware resource allocation of each software task or module and the priority of software module execution. Yet, the choice of resource allocation schemes highly depends on a given optimization goal. Various optimization goals will lead to different optimal hardware resource allocation strategies.

As an example, Fig. 3 shows a typical perception module in state-of-the-art CAVs, which employs three sensors - LiDARs, cameras and radars - to capture the surroundings. LiDARs use lasers for sensing, having great 3D data measurement ability. Cameras capture color information, important for objects like traffic lights. Radars detect other objects' distances and velocities. Among the three sensors, while LiDARs are the most critical sensor for emerging CAVs, the LiDAR tasks can

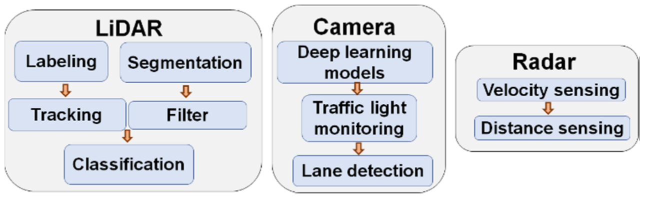

Fig. 3. A general perception module using three sensors in CAVs.

have long latencies. We run the five LiDAR tasks shown in Fig. 3 on a NVIDIA Jetson TX2 board when all tasks are executed only on CPU. Fig. 4(a) shows four tail latencies (50th-percentile, 95th-percentile, 99th-percentile and maximum latency) for those five LiDAR tasks respectively. We observe that the maximum latency of even a single segmentation task is longer than 100ms. This shows it can be challenging to meet the real-time requirements if we run an end-to-end LiDAR module on a single Jetson TX2 board without a proper resource management scheme to efficiently utilize the available resources (e.g., CPUs, GPUs) of the vehicle's computing architecture

Case study. To improve performance and ensure safety, we hope to develop efficient resource allocation schemes [5]. We propose two principles for the design. First, we always run the bottleneck task (the task with the longest maximum latency) on GPUs. Second, we utilize the potential parallelism. If two

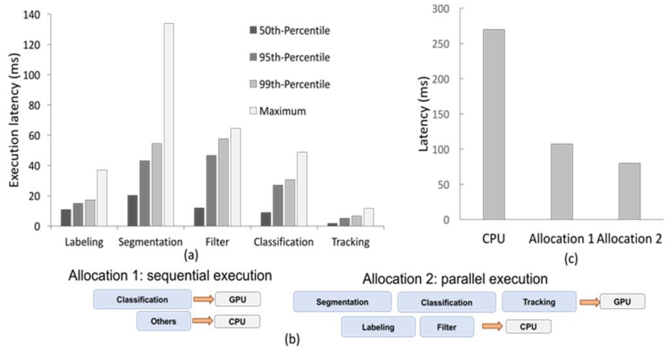

Fig. 4. (a) Tail latency for Lidar tasks on CPU, (b) Different allocation schemes, (c) Latency comparison among three allocations

modules do not have dependency, we offload them to CPUs and GPUs respectively. Then we demonstrate a simple case study which includes two resource allocation schemes: allocation 1 and allocation 2 as shown in Fig. 4(b). In Fig. 4(c), we compare the latency achieved by the two allocation schemes, along with a scheme which allocates all the tasks to CPU. It shows that allocation 2 with more efficient resource scheduling outperforms others in terms of latency. We conclude that different resource allocation schemes can achieve different latencies, and finding the optimal resource allocation scheme can be important to achieve the desired latency performance.

B. *What is the optimal computing architecture?*

Since future vehicles will have a widely varying set of capabilities and requirements, an important question comes up: what is the optimal computing architecture for a particular vehicle class. For example, given a number of camera sensors that need to be supported for a vehicle, and hence the number of machine vision (e.g., object detection and tracking) instances that need to be executed simultaneously, and the desired latency-power requirements of the instances, what may be the

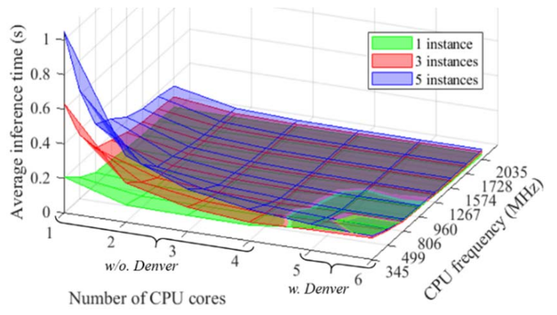

Fig. 5. Variation of average inference time for different application requirements with respect to different system configurations

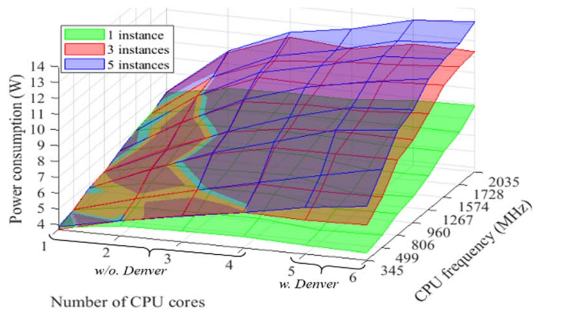

Fig. 6. Variation of power consumptions for different application requirements with respect to different system configurations

TABLE I. Performance comparison between Cortex and Denver cores

Core (clock frequency = 345 MHz)	Inference time (s)	Power consumption (W)
Cortex A57	0.089	4.56
Denver 2	0.137	4.25

optimal computing architecture (number of CPU/GPU cores and their frequencies) that can satisfy the capability and the requirements. In this section, we explore the above question by conducting a preliminary experiment using an Nvidia Jetson Tx2 board to execute multiple SSD MobileNet v2 [16] (object detection) instances, corresponding to the number of desired cameras for the vehicle, instances. We will show in the following paragraphs the corresponding average inference time of all the instances and power consumption of the Jetson board.

To emulate different computing architectures, we manually change the hardware architecture of the Jetson board, including changing the number of CPU cores, the CPU clock frequency, and the GPU clock frequency. According to [19], Nvidia Jetson Tx2 board has 4 ARM Cortex A57 cores and 2 Denver cores. In this experiment, Denver cores are utilized when the available number of cores is more than 4. On the other hand, the value of the CPU clock frequency is among 8 predefined frequencies ranging from 345 MHz to 2.035GHz. The GPU of the Jetson Tx2 is always on and its clock frequency can be set to either 624 MHz or 1.3 GHz.

Fig. 5 and 6 show the average inference time and power consumption for running 1, 3, 5 SSD MobileNet v2 instances on Nvidia Jetson Tx2, respectively. Due to the visualization limitation, in these two figures, we keep the GPU clock frequency at 1.3 GHz and show different architectures resulting from varying the number of CPU cores and CPU clock frequencies. The performance of different required capabilities, 1, 3, 5 instances, are shown by green, red, and blue surfaces, respectively. Fig. 5 shows the relationship between the average inference time, capability, and the chosen architecture, with the inference time mostly reducing with increasing number of ARM cores and increasing core frequency, except when Denver cores are included in the architecture, the latter producing overlapping regions. Based on our observation in TABLE I, obtained by running an SSD MobileNet v2 instance on ARM and Denver cores at 345 MHz clock frequency, respectively, the average inference time performance of the Denver core is longer than the Cortex core. Therefore, the overlapping regions happen when the task scheduler schedules the only instance to the Denver cores in 1 instance case.

TABLE II. Available architectures for average inference time < 0.08 s when running 5 instances

Architecture (CPU cores, freq*), (GPU freq*)	Power consump-tion (W)	Architecture (CPU cores, freq*), (GPU freq*)	Power consump-tion (W)
(3, 1267), (1300)	9.50	(5, 1267), (1300)	11.4
(3, 1574), (1300)	10.4	(5, 1574), (1300)	12.2
(3, 1728), (1300)	10.7	(5, 1728), (1300)	12.4
(3, 2035), (1300)	11.4	(5, 2035), (1300)	12.9
(4, 960), (1300)	9.59	(6, 806), (1300)	10.4
(4, 1267), (1300)	10.7	(6, 960), (1300)	10.8
(4, 1574), (1300)	11.2	(6, 1267), (1300)	12.0
(4, 1728), (1300)	11.4	(6, 1574), (1300)	12.4
(4, 2035), (1300)	12.0	(6, 1728), (1300)	12.6
(5, 806), (1300)	9.98	(6, 2035), (1300)	13.2
(5, 960), (1300)	10.6		

*clock frequency in MHz

The power consumption for running SSD MobileNet v2 instances on Nvidia Jetson Tx2 is shown in Fig. 6. Note that the power consumption for running 1 and 3 instances at low clock frequencies reduces when the Denver cores are present. The reason is that the Denver core consumes less power than the ARM core for running the SSD MobileNet v2 application, as shown in Table I. These overlapping regions in Fig. 5 and 6 point out the need of a task scheduler which is more aware of the application type and the architecture choices (e.g. between Cortex and Denver cores) given the performance requirements.

The data presented in Fig. 5 and 6 show the possibility of developing models to estimate inference time and power consumption based on given requirements (instances) and architecture choices (number of cores, type of cores and frequency). Moreover, the models constructed can be used to help identify an optimal architecture for given capabilities and performance requirements of a set of vehicle applications. For example, when the requirements are using 5 cameras and thereby 5 object detection instances, with maximum average inference time requirement of 0.08 s, the available architectures are listed in Table II based on the data shown in Fig. 5. Table II also shows the corresponding power consumption observed in Fig. 6. Among all the architectures in Table II, if the objective is to minimize the power consumption, then the optimal architecture will be using 3 CPU cores at 1.267 GHz clock frequency, as the row highlighted shows. Following the same method, we can derive Table III, which shows the optimal computing architecture, including various GPU clock frequency, for different capabilities to 1) minimize average inference time under the 10 W power consumption constraint, and 2) minimize power consumption while satisfying the 0.08s average inference time constraint. The optimal system architecture is shown in the format of: (Number of CPU cores,

TABLE III. Optimal architecture for different capabilities and objectives

Ins-tances	Minimizing inference time (Power consumption < 10W)		Minimizing power consumption (Inference time < 0.08 s)	
	Optimal architecture (CPU cores, freq*), (GPU freq*)	Avg. Inference time (s)	Optimal architecture (CPU cores, freq*), (GPU freq*)	Power consump-tion (W)
1	(4, 2035), (1300)	0.0244	(2, 499), (624)	4.05
2	(2, 2035), (1300)	0.0337	(3, 499), (624)	4.85
3	(4, 1267), (1300)	0.0451	(2 , 960), (624)	5.53
4	(5, 960), (1300)	0.0583	(4, 1267), (624)	6.50
5	(5, 806), (1300)	0.0716	(3, 1267), (1300)	9.50

*clock frequency in MHz

CPU clock frequency), (GPU clock frequency) in Table III. The results show the tradeoff between changing the number of CPU cores as well as CPU and GPU clock frequencies to achieve the required objective. For example, when the capability changes from 2 to 3, the optimal architecture has a reduction in the number of cores but increase in the CPU clock frequency for minimizing the power consumption. The above experiments and observations demonstrate that the optimal architecture varies significantly depending on the requirements, and hence the need for design-space exploration to identify the optimal architecture.

V. Collaborative Computing for Connected Vehicles

In the era of multi-scale computing, vehicular applications can leverage the advantage of more powerful data processing nodes distributed remotely. Edge computing is one such promising computing paradigm that is one hop wireless distance away from the sensors to ensure lower latency than traditional cloud computing. In the context of vehicular edge computing (VEC), the edge computing nodes can be located at the RSU, which can communicate with the vehicles over wireless networks [9-12]. The motivations for distributing vehicular computing tasks between vehicular local computing (VLC) units and VEC nodes can be multifold, including the ability of vehicles with limited computing capability to benefit from advanced compute intensive vehicular applications. Moreover, the vehicle makers can add new applications to their existing vehicles without requiring upgrading their existing computing platform and constantly re-designing their vehicle computing platform for future vehicles.

Fig. 7 shows an example of two connected vehicles utilizing the VEC resources from RSU to execute some computing tasks, where the black circles represent the connected vehicles' computing tasks and red circles present the RSU's computation tasks. Black circles in the purple boxes are the tasks executed by VLC units and that in the orange boxes are the tasks offloaded to the VEC node. With the knowledge of the dependency graph of a vehicular application like fig. 3, connected vehicles can have more flexibility on offloading. The performance can be further improved by partitioning the tasks and offloading some of them to VEC node according to the current channel condition and VEC/VLC computing resource availabilities.

However, the decision for partitioning and offloading tasks is nontrivial as it depends on several factors, including computing capability of the VLC unit, communication bandwidth, VEC node's computing capacity, and the achievable task partitioning within the vehicular applications.

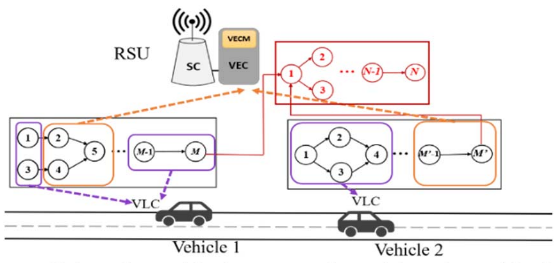

Fig. 7. Collaborative vehicular computing with task partitioning and offloading to edge computing server connected to the RSU.

A. *Case Study*

In [12], we had done some preliminary work developing algorithms and analyzing benefits of task partitioning and offloading between connected vehicles and RSUs. In this paper, we show in real-world and implementation-wise how vehicular applications can benefit from collaborative computing by conducting a case study using our testbed which consists of vehicles with cameras and NVIDIA Jetson-based VLC nodes, and RSUs with NI USRP radios [20] and Jetson-based VEC nodes attached. The vehicles communicate to the RSU over 4G LTE communication channels. In our case study, we focus on object detection algorithms, whose tasks can be partitioned such that some tasks can be executed on the VLC node and some can be computed at the VEC node. For our experiments, we consider the following two partitions of the three tasks - T1: image read and resize tasks, and T2: object detection task. Note that object detection models are trained to use a specific size of the image. Here we use SSD MobileNet v2 [16], which uses input size as 300x300. Depending on the computing capacity of the VLC unit, the delay for T1 and T2 can vary significantly. But, if T1 can be done on VLC unit, it reduces the size of the data to be transmitted over wireless networks to the VEC node, and hence reducing the communication latency. Here we chose to execute the same object detection model on the VLC unit and VEC node, which can provide the same level of accuracy but different inference times. Our decision for optimal offloading is based on the lowest achievable inference time with respect to the available resources. We tested with real-word vehicular image frames of average data size 45 KB, collected from the front camera of our testbed vehicle, using different VLC unit computing capabilities and communication bandwidth conditions. The VEC node has a CPU with frequency 2 GHz and a GPU with frequency 1.3 GHz.

1) *Preliminary Results: Need for Optimal Task Partitioning*

We varied the VLC capacities and measured the delay for partitions T1 and T2 and corresponding communication delays with respect to the bandwidth. Fig. 8 shows the variation of end-to-end delays when the VEC node is twice as powerful as the VLC unit. The individual color shows the specific delay components. The three bars indicate: (i) no offload, i.e. T1 and T2 both are done at VLC unit, (ii) partition and offload, i.e., T1 at VLC unit, communication of resized image and T2 at VEC node, and (iii) full offload, i.e. communication of full image, then T1 and T2 both at VEC node. When the communication speed is low (5 Mbps), no offloading is the best as it has the minimum end-to-end delay. As the communication speed improves (10-20 Mbps), partitioning and offloading seems to be the best choice as full offloading still has high enough

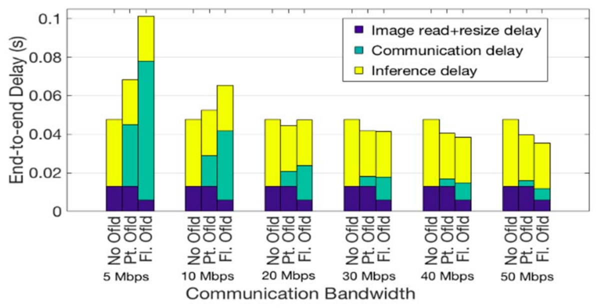

Fig. 8. End-to-end delay for no offloading, partition and offloading and full offloading showing different delay components w.r.t different communication bandwidths. The computation power of VLC here is half of the VEC computation capability.

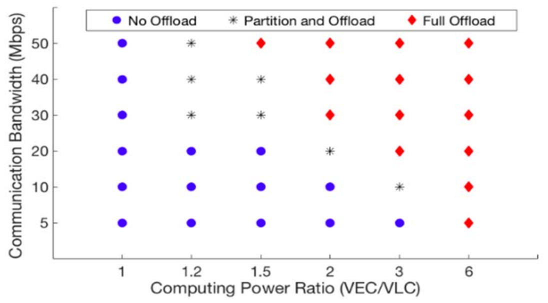

Fig. 9. Optimal Offloading decisions based on minimum-end-to-end delay with respect to different VLC computing capability and available communication bandwidths

communication delay. However, when the network quality improves further (> 30 Mbps), full offloading is the best choice. However, the choice of offloading decision further varies with the variation in the VLC configurations.

We further varied the VLC configurations in terms of number of cores, CPU frequency, and GPU frequency proportionally and measured end-to-end delay of the tasks in different offloading decisions. Fig. 9 shows the optimal offloading decisions (in terms of minimum end-to-end delay) for varying VEC/VLC computing power ratio. It shows that when VLC unit is as powerful as VEC node, then irrespective of communication conditions, no offloading is the best policy. Similarly, when the VLC capability is six times inferior than the VEC, full offloading is the best policy even in weak network conditions e.g. 5 Mbps. For other computing power ratios, there are tradeoffs and the optimal choice can vary among three offloading policies.

The experimental results show that depending on the computing and communication resource availabilities, one can make optimal choices leveraging the VEC nodes. In more complex future applications in CAVs, there will be possibilities for more complex and enhanced task partitioning scenarios, necessitating the need for algorithms which can explore the large search space and make optimal decisions depending on varying computing and communication availabilities.

VI. Conclusion

In this paper, we have introduced a vision for the evolving smart vehicle's computing needs and architecture considerations. We have introduced the needs, efficacy and research opportunities of optimally allocating existing computing resources for emerging vehicular applications, modeling vehicular application performance for different capabilities and computing architectures, and tools to help develop optimal in-vehicle computing architectures as well as collaborative computing system with edge computing nodes. Although this paper mainly focuses on a centralized architecture for in-vehicle computing, a distributed architecture is feasible, which can add more resiliency and parallelisms, especially with simultaneous computations of multi-sensor data, albeit with the overhead of handling synchronizations and causality.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1619184, the UC San Diego Center for Wireless Communications, and the Smart Transportation Innovation Program (STIP).

References

[1] S.-C. Lin, *et al.*, "The architectural implications of autonomous driving:Constraints and acceleration," in *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2018, pp. 751–766.

[2] N. Lu, *et al.*, "Connected Vehicles: Solutions and Challenges," in *IEEE Internet of Things Journal*, vol. 1, no. 4, pp. 289-299, Aug. 2014.

[3] A. Driving, "Levels of driving automation are defined in new SAE international standard J3016: 2014," SAE International. Google Scholar

[4] K. Zhang, *et al.*, "Mobile-Edge Computing for Vehicular Networks: A Promising Network Paradigm with Predictive Off-Loading," in *IEEE Vehicular Technology Magazine*, vol. 12, no. 2, pp. 36-44, June 2017.

[5] H. Zhao, *et al.*, "Towards Safety-Aware Computing System Design in Autonomous Vehicles" arXiv:1905.08453, 2019.

[6] S. Chen et al., "Vehicle-to-Everything (v2x) Services Supported by LTE-Based Systems and 5G," in *IEEE Communications Standards Magazine*, vol. 1, no. 2, pp. 70-76, 2017.

[7] J. Feng, Z. Liu, C. Wu and Y. Ji, "Mobile Edge Computing for the Internet of Vehicles: Offloading Framework and Job Scheduling," in *IEEE Vehicular Technology Magazine*, vol. 14, no. 1, pp. 28-36, March 2019.

[8] C. Chen, *et al.*, "Delay-Optimized V2V-Based Computation Offloading in Urban Vehicular Edge Computing and Networks," in *IEEE Access*, vol. 8, pp. 18863-18873, 2020.

[9] Y. Dai, D. Xu, S. Maharjan and Y. Zhang, "Joint Load Balancing and Offloading in Vehicular Edge Computing and Networks," in *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4377-4387, June 2019.

[10] Z. Zhou, *et al.*, "Energy-efficient workload offloading and power control in vehicular edge computing," *2018 IEEE Wireless Communications and Networking Conference Workshops*, Barcelona, 2018, pp. 191-196.

[11] K. Zhang, *et al.*, "Mobile Edge Computing and Networking for Green and Low-Latency Internet of Things," in *IEEE Communications Magazine*, vol. 56, no. 5, pp. 39-45, May 2018.

[12] Y. Ku and S. Dey, "Sustainable Vehicular Edge Computing Using Local and Solar-Powered Roadside Unit Resources," *IEEE 90th Vehicular Technology Conference (Fall)*, Honolulu, HI, USA, Sep. 2019, pp. 1-7

[13] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," arXiv:1708.06374, 2017.

[14] "Planning a Safer Path: NVIDIA Safety Force Field Protects Against Real-World Traffic", Nvidia Co., CA, USA. (Accessed: Apr. 9, 2020.) https://www.nvidia.com/en-us/self-driving-cars/safety-force-field/

[15] J. Redmon, *et al.*, "You Only Look Once: Unified, Real-Time Object Detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, 2016, pp. 779-788.

[16] M. Sandler, *et al.*, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, 2018, pp. 4510-4520.

[17] Junko Yoshida, "AV safety ventures beyond ISO 26262", EETimes, 2019. (Accessed: Apr. 9, 2020.) https://www.eetimes.com/document.asp?doc_id=1334397#

[18] J. Stewart "Self-Driving Cars Use Crazy Amounts of Power, and It's Becoming a Problem", Wired.com. (Accessed: Apr. 9, 2020.)

[19] https://www.wired.com/story/self-driving-cars-power-consumption-nvidia-chip/ "NVIDIA Jetson TX2 Series System-on-Module", Nvidia Co., CA, USA. Datasheet v1.2, 2014, (Accessed: Apr. 9, 2020) https://developer.nvidia.com/embedded/jetson-tx2

[20] "USRP B210", Ettus Research, TX, USA. (Accessed: Apr. 9, 2020) https://www.ettus.com/all-products/ub210-