Cloud-assisted Control of Ground Vehicles using Adaptive Computation Offloading Techniques

Author names & affiliation removed for the purpose of double-blind review

***Abstract*— The existing approaches to design efficient safety-critical control applications is constrained by limited in-vehicle sensing and computational capabilities. In the context of automated driving, we argue that there is a need to think *"out-of-the-vehicle"* to meet the sensing and powerful processing requirements of sophisticated algorithms (e.g., deep neural networks). The network data rate and bandwidth supported by the existing communication standards cannot adequately meet the hard-real-time requirements of core control computations. Therefore, it is important to identify parts of control computation that are amenable for offloading to the cloud (or edge)-based resources. We also need to develop mechanisms to handle network communication failure scenarios such as connectivity loss and large delays. To address the challenges, we propose an adaptive offloading technique for control computations into the cloud. The proposed approach considers both current network conditions and control application requirements to determine the feasibility of leveraging remote computation and storage resources. As a case study, we describe a cloud-hosted planning algorithm for a path following controller that is cognizant of dynamically changing road conditions using crowdsensed data. Our simulation results demonstrate improved controller performance in terms of distance and time, even in the presence of transient connectivity loss and large delays.**

I. Introduction

Modern cars increasingly rely on advances in information technology to offer safety-critical features to the users. Advanced driver-assistance systems (ADAS) help guide drivers and reduce accidents through various automated and adaptive features such as adaptive cruise control (ACC), autonomous emergency braking, and forward-collision warning. Such intelligent safety-critical aspects in vehicles are realized by running complex algorithms that process data obtained from several types of sensors such as RADAR, LIDAR, video camera, GPS, etc. The in-vehicle sense and computational capabilities have certain limitations such as sensing range and amount of energy required to perform compute-intensive tasks. The in-vehicle capabilities cannot be enhanced through additional hardware (such as duplicate sensors and powerful processors) beyond a certain limit after without increasing the system complexity and cost.

We posit that cloud computing technology can help address some of the existing limitations in terms of sensing and computation. Cloud computing allows users to acquire, scale up/down, and release computational and storage resources on-demand. Cloud resources can be treated as a utility (like electricity, for example) that can be accessed over Internet. Thus, cloud technology can be a suitable candidate to address the limitations of in-vehicle capabilities and enhance the functional and safety aspects of automotive control applications. Some of the major automakers offer several cloud-based non-control services like emergency scenario response such as remote unlocking, crash response, stolen vehicle tracking, battery health monitoring, and other diagnostic solutions.

We view cloud as a technology that not only offers an enhanced computational platform but also as an enriched information source (through means such as crowdsourcing). The enriched information essentially enhances the perception range of a vehicle by fusing together sensor data from multiple vehicles. In the context of offering advanced driver assistance features and autonomous driving, we envisage more and more automotive control applications involving cloud being developed in the future. Therefore, we think it is essential to explore various parts of automotive control application that benefit from and are feasible to be executed in the cloud. It is also necessary to develop techniques that are required to deal with potential failure scenarios so that the safety and stability of the control system is not compromised.

In this light, we have developed an offloading controller architecture that determines when it is feasible and beneficial to offload control computations, that may leverage additional data and computational resources, to the cloud. The proposed offloading approach is opportunistic and hence it is flexible to toggle between in-vehicle and cloud-based resources. We demonstrate the feasibility of our approach using a cloud-based path following controller case study in Matlab. The primitive waypoint generation algorithm for the controller is generated in cloud, leveraging crowdsourced data on current road conditions. The network communication delays are modeled using NS-3 LTE library. Our results show improved performance of the path following controller in terms of both distance traveled by the vehicle and the time taken between a given source and destination.

II. Adaptive Offloading Controller Design

In this section, we present the computation offloading controller design for automotive control applications that leverages cloud. This approach enables dynamic switching between and on-board and remote resources based on the control application specification and current network conditions.

In the mobile cloud computing area, cloud-based augmentation techniques [1], [2] and computation offloading problem [3], [4] have been extensively studied. However, major emphasis is placed on saving energy on mobile devices and

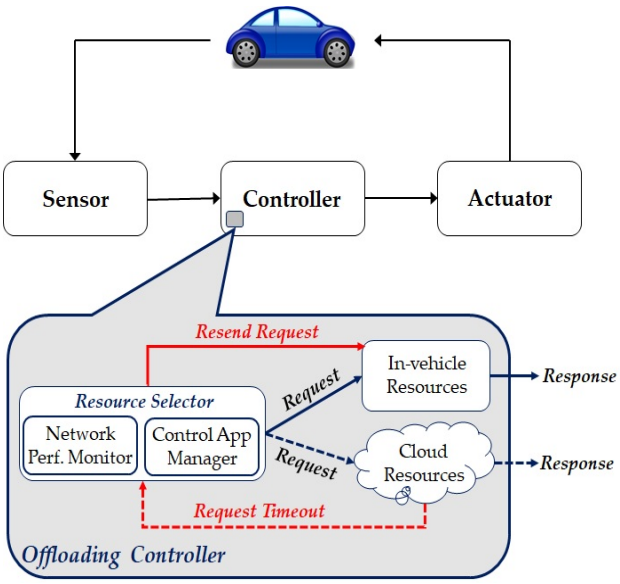

Fig. 1. Computation Offloading Controller Architecture

computation offloading is largely determined by the viability of network conditions at any given point of time. In the automotive domain, however, the most important consideration is satisfying the real-time constraints of the control applications. A typical control loop in an application like adaptive cruise control (ACC) takes 10 ms. This, given the current network communication standards, makes it infeasible to offload core control computation to the cloud. Therefore, we need to identify parts of control computation that are amenable for offloading to the cloud without compromising on the safety and stability of the system. The candidate computational part must have a delay tolerance in the range of 150-200 ms and potentially benefit from additional computation resource and/or data accessed via cloud.

Figure 1 shows our offloading controller architecture design for control applications. The basic functionality of the offloading controller is to determine whether to execute a given computational task on local or remote resource. The decision-making process relies two components, namely, Network Performance Monitor and Control Application Specification. The network performance monitor component continuously monitors the network characteristics such as available bandwidth and uplink data rate. The control application specification component identifies the self-contained parts of control computation (e.g., pre-processing sensor data, vehicle state estimation) and their computational and latency requirements. These parts are specified as tasks and this task specification along with the current network conditions guide the process of selecting a resource. Once a suitable resource for a given task is identified, the task is executed on the resource and the response is fed to the control loop.

Algorithm 1 describes the step-by-step process of resource selection and offloading. When the cloud resource is selected for task execution and the input data is offloaded to the cloud,

Algorithm 1: Computation offloading: high-level steps

/* Computation task $T_i = (S_i, C_i, L_i)$ where, */
/* S_i, S_i': size of the input & output data */
/* C_i: # of CPU cycles required for task T_i */
/* L_i: latency requirement of the task T_i */
/* C_{cloud}: # of CPU cycles available in the cloud */
/* β, β': available uplink and downlink data rate */
/* T_i^{rsrc}: resource to which the task i is assigned */

Input : Set of input tasks $\mathcal{T} = \{T_i\}, i = 1, 2, ..., n$
Output: Task execution output: T_i^{out}

while $\mathcal{T} \neq$ NULL **do**
  measure available uplink data rate for task T_i: β
  estimate network delay: $\delta_i^{net} := \frac{S_i}{\beta} + \frac{S_i'}{\beta'}$
  estimate computation delay: $\delta_i^{comp} := \frac{C_i}{C_{cloud}}$
  if $\delta_i^{net}+\delta_i^{comp} < L_i$ **then**
    T_i^{rsrc} := *cloud*
  else
    T_i^{rsrc} := *on-board*
  end if
  execute task on T_i^{rsrc} and **return** T_i^{out}
  if T_i^{rsrc} == cloud $\wedge$ timeout **then**
    execute task T_i on *on-board* resource and **return** T_i^{out}
  end if
end while

the response might not come back within a desirable timeframe (say, for example, due to temporary resource outage). In such cases, the request timeout will get triggered and the same request will now have to be forwarded to the local resource. As pointed out in Section II, due to limited understanding of the surrounding context in the local-only case, the response generated by the local resource may be less efficient (in terms of time taken for task execution and/or the quality of control output) compared to a cloud-based approach. In the tradeoff analysis, particularly in a feedback control system in the automotive domain, the emphasis is more on the stability of the system than quality of output in one control step as compromising on stability could lead to undesirable consequences.

III. Case Study: Cloud-based Path Following Controller

For communication latency modeling, we initially adopted an analytical model approach inspired from the Mobile Cloud Computing area. However, we realized that the analytical model approach was inadequate to capture the mobility aspects of the vehicles and lacks the ability to specify accurate communication link characteristics. So, we utilized the NS-3 LTE library [5] to simulate the communication channel and measure latency using UDP application traffic. We had used

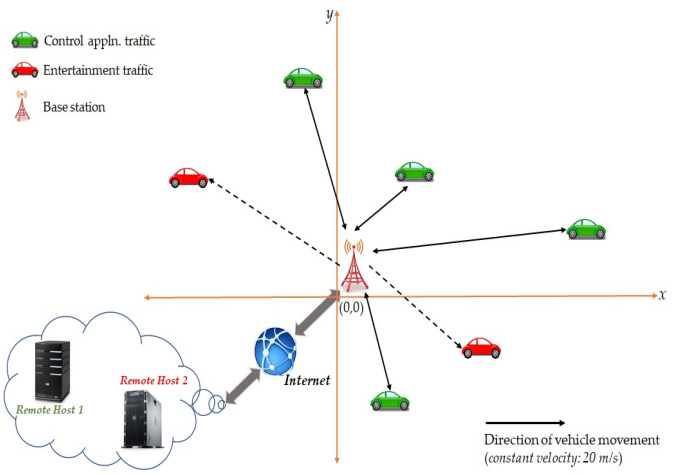

Fig. 2. Communication channel simulation setup

the measured latency values in a path following control system in Matlab and studied the behavior of the system in different scenarios and under varying network conditions.

A. Communication channel modeling using NS-3 LTE library

The simulation setup in NS-3 includes one base station (aka eNB node) and constant velocity mobility model for the vehicles. Since we used LTE communication technology, we generated two distinct categories of traffic one related to the control application and another video streaming traffic. The traffic flow in the former category is bidirectional whereas the entertainment traffic is unidirectional (from streaming servers to the end users in vehicles). The simulation setup is shown in Figure 2.

The downlink and uplink frequency bands of the cell are set to 2110 and 1710 MHz, respectively. The transmission power of the eNB node and the vehicle nodes is set to 40 and 20 dBm, respectively. In our simulation, Proportional Fair MAC Scheduler and Friis path loss model is used. The transmission method between the eNB node and vehicles is a single-hop unicast. Two distinct types of applications are configured on each remote host an UDP Echo application on Remote Host 1 and a packet generator application on Remote Host 2.

The control application traffic constitutes 80% of the total traffic and the remaining 20% is the entertainment traffic (1024 B packet every 20 ms). The packet size of the control application traffic transmitted by all but one vehicles is in the range of 128 to 2048 B every 20 ms. The remaining one vehicle, for which we are measuring the latency values, transmits 512 B packet every 20 ms. Table I shows the latency values obtained by varying number of vehicles and the network bandwidth. The latency values noted is an average of 10 runs for the vehicle the that transmits 512 B packets every 20 ms.

B. Path following controller simulation in MATLAB

We had chosen the pure pursuit controller algorithm that is available in the Robotic Systems Toolbox in Matlab [6]. The sampling interval for the pure pursuit controller meets some

Bandwidth \ # vehicles	15	30	45	60	75
5 MHz	51.44	52.09	52.92	99.49	100.66
10 MHz	37.28	42.01	42.1	68.49	89.38

TABLE I
DATA TRANSFER LATENCY (MS)

of the offloading constraints specified in Section II and hence determined as a suitable candidate to offload the computation to the cloud.

The pure pursuit controller basically implements a path tracking algorithm that keeps track of and controls the robots movements along the path. The path between a source and destination is comprised of a set of waypoints (or intermediate [x y] coordinates). The fundamental operation in the controller is the computation of linear and angular velocities for the next control step based on the current pose of the robot. The pose is specified as [x, y, theta], where (x, y) represents a point in the reference coordinate system and theta represents the heading angle of the robot.

In this work, we have extended the static waypoint generation and path following controller code (in [6]) to dynamically generate waypoints using purely on-board or remote cloud resources. The other contribution is the simulation of failure scenarios such as network connectivity loss and large delays and demonstrate switching between local and cloud resources for waypoint generation. The cloud-based waypoint generation factors in the most up-to-date information about the constantly changing road conditions and hence can compute a more efficient path for the robot.

In our work, we adopt a primitive approach for waypoint generation. We assume that the robot is operating in an environment where from any given point, there are three potential directions namely, vertical, horizontal, or diagonal – the robot can move on. Each potential direction has a weight associated with it that captures the quality of the path (e.g., if a path has a pothole or stalled traffic, it will have less weight and hence deemed as a low-quality path). The primitive waypoint generation algorithm always picks the path with the highest quality. In the local-only waypoint generation approach we assume that the weights remain static or change at much lower frequency compared to cloud (due to the large overhead incurred in V2V communication, for example). Therefore, the local-only approach might yield a path between source and destination that incurs larger distance, more time, or both.

The first set of results shown in Figure 3 demonstrates the differences in path computed fully on-board (a) and upfront (without factoring in changes in the road conditions) and path computed fully in the cloud (b) that dynamically adjusts weights based on crowdsourced data. We can see that the fully remotely computed path incurs less distance and time. The noted time values are adjusted by multiplying it with the path weight. For the cloud computation part, we have used the communication latency values obtained for N=30 vehicle case in the NS-3 simulation.

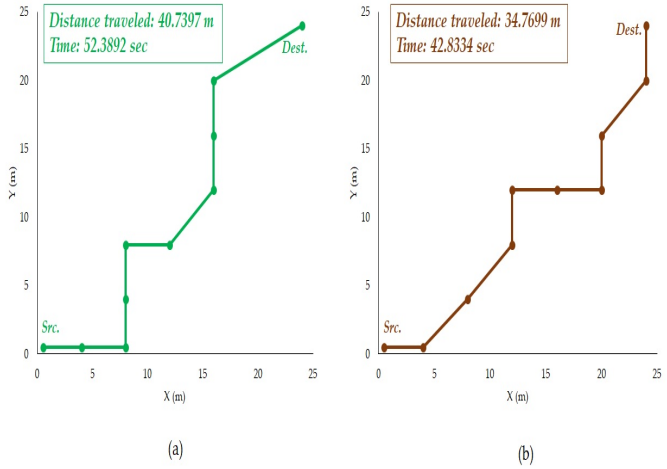

Fig. 3. Waypoint generation: (a) fully on-board without factoring in dynamically changing road conditions; (b) fully in the cloud using up-to-date data on current road conditions

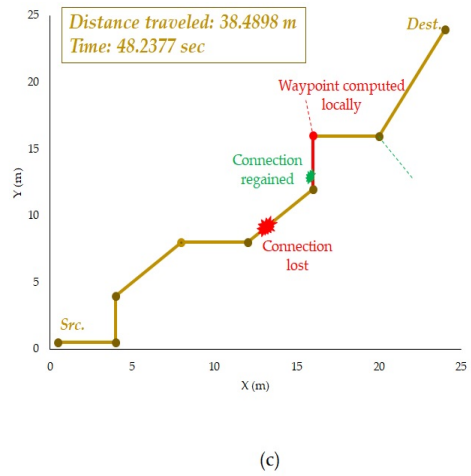

Fig. 4. Waypoint generation: (c) a scenario where both cloud and on-board resources are involved due to connectivity loss

The second set of results shown in Figure 4 highlight the switching between local and remote resources in the presence of a network communication failure. It shows the impact of connectivity loss to the cloud server. When the connectivity loss is detected by the Network Performance Monitor component of the offloading controller (c.f.: Figure 1), the resource selector component chooses to leverage local resources for the next waypoint generation. Since the local knowledge is either static (or less frequently updated) compared to cloud, it results in a less efficient path than a purely cloud-based approach.

Figure 5 (d) shows a scenario where a request is originally forwarded to the cloud but the response hasnt been received on time (say, due to temporary cloud resource outage). In this case, the timeout value for the request expires and the resource selector would choose to forward the request to the local resource (Note: here the timeout value is chosen based on the latency requirement to receive next waypoint specified in the

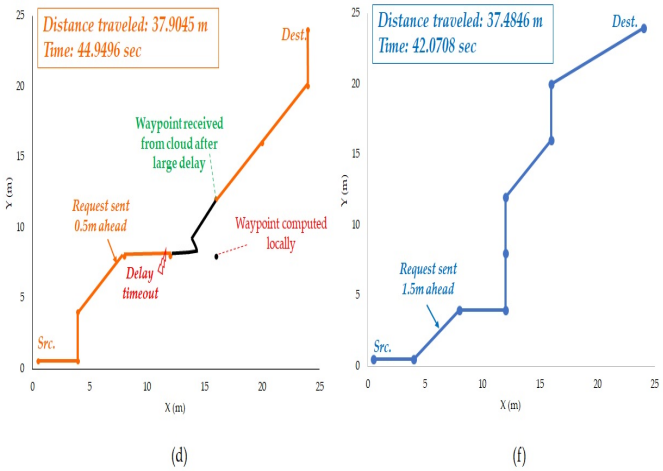

Fig. 5. Waypoint generation: (d) a scenario where both cloud and on-board resources are involved due to large delay; (e) overcome transient large delays by requesting ahead of time

Control Application Specification component in the offloading controller). Once the next waypoint from the cloud is received (after a large delay), the robotic controller would update its intermediate goal and starts moving towards that direction.

In Figure 5 (d), the request for next waypoint is sent when the robot is 0.5 m ahead of the intermediate goal. One way to overcome large delay is to increase the distance at which the request for next waypoint is sent to the cloud. Figure 5 (e) shows the performance results when the request is sent 1.5 m ahead of intermediate goal (as against 0.5 m ahead).

In this section, we have shown that a cloud-based path following controller improves the performance of the system compared a fully local resource based controller. Also, we have demonstrated the feasibility of switching between local and remote resources that effectively results in stable performance of the system despite a minor performance degradation.

IV. Related Work

A few research groups in academia considered cloud-based automotive control applications and developed prototype models involving cloud. One such work, called Carcel [7], involves development of a cloud-assisted system for obstacle detection and avoidance for efficient path planning for autonomous vehicles. The system uses infrastructure sensor data in addition to in-vehicle data to sense obstacles ahead of time. The results reported in this work show a better reaction time to obstacles compared to a non-cloud-based collision avoidance system. A survey of decision making problems in self-driving cars with a focus on motion planning is presented in [8]. Motion planning involve computing dynamically feasible trajectory computation for a given source and destination pair. This survey presents different approaches to planning and their computational complexity.

A declarative approach to detect dangerous events by combining vehicle sensor data and static road information obtained from a cloud database is proposed in [9]. The declarative rules

are defined using a sensor predicate and a cloud predicate. A cloud-based vehicle location tracking approach for urban environments, called CARLOC, is proposed in [10]. GPS systems have typically poor accuracy in urban environments. A probabilistic position that uses odometer, vehicle heading direction information from the vehicle and an online map database and landmark information is used to achieve lane level accuracy in determining the location of a vehicle.

A software architecture for cloud-based automotive control is proposed in [11]. The proposed futuristic approach involve only sense and actuate functions in the physical world whereas the control computations are preformed entirely in the cloud. This would enable offering Control as a Service (CaaS). The vehicle owners can customize and access control as a service on-demand from the cloud under the pay-per-use model. An architecture for cloud-based remote vehicle diagnostics, called AutoPlug, is proposed in [12]. The proposed architecture would enable proactively detecting software faults using diagnostic trouble codes, and allow code updates to be dispatched from a remote datacenter to vehicle electronic control units.

In contrast with the works cited above, our work aims to arrive at a generic approach to leverage cloud for control applications based on computational resource/data availability. Our approach also factors in varying communication link conditions (such as path loss and large delays) and allows dynamically switching between local and cloud resources. This ensures stability of the control system, although the control application performance may be less efficient in a local-only computation/data approach.

V. Conclusion

Cloud computing (and other remote computing technology variants such as fog or edge computing) is gaining traction in the automotive domain. To enhance in-vehicle and fault tolerance capabilities of self-driving vehicles, critical control applications can leverage such technologies. In this work, we presented our envisaged offloading controller design that would enable realization of remote resource based control applications. Using a cloud-based path following controller case study, we had demonstrated the feasibility of our approach in the presence of hostile network conditions. In the future, this work can be extended to study a more comprehensive control application and quality metrics that would allow us to perform a detailed trade-off analysis between the fully local computation and remote computation approaches. In addition, more hostile network conditions such as packet loss, corrupted packet can be simulated using NS-3 and interfaced with Matlab.

References

[1] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, and R. Buyya. Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges. *IEEE Communications Surveys & Tutorials*, 16(1):337–368, 2014.

[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 2009.

[3] K. Kumar, J. Liu, Y. Lu, and B. Bhargava. A survey of computation offloading for mobile systems. *Mobile Networks and Applications*, 18(1):129–140, 2013.

[4] K. Akherfi, M. Gerndt, and H. Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied Computing and Informatics*, 2016.

[5] NS3-LTE Library. https://www.nsnam.org/docs/models/html/lte.html, 2016. [Online; accessed Aug. 18, 2017].

[6] Path following controller. https://www.mathworks.com/help/robotics/examples/path-following-for-a-differential-drive-robot.html, 2015. [Online; accessed Aug. 18, 2017].

[7] S. Kumar, S. Gollakota, and D. Katabi. A cloud-assisted design for autonomous driving. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 41–46. ACM, 2012.

[8] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[9] Y. Jiang, H. Qiu, M. McCartney, W. Halfond, F. Bai, D. Grimm, and R. Govindan. CARLOG: A platform for flexible and efficient automotive sensing. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, pages 221–235. ACM, 2014.

[10] Y. Jiang, H. Qiu, M. McCartney, G. Sukhatme, M. Gruteser, F. Bai, D. Grimm, and R. Govindan. CARLOC: Precise positioning of automobiles. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 253–265. ACM, 2015.

[11] E. Hasan. Control as a service (CaaS) : Cloud-based software architecture for automotive control applications, 2015.

[12] R. Mangharam. The car and the cloud: Automotive architectures for 2020. *The Bridge*, 2012.