Enabling Vehicular Applications using Cloud Services through Adaptive Computation Offloading

Ashwin Ashok, Peter Steenkiste, †Fan Bai
Carnegie Mellon University, †General Motors Research
ashwinashok@cmu.edu, prs@cs.cmu.edu, †fan.bai@gm.com

ABSTRACT

There is growing interest in embedding new class of applications in vehicles to improve the user driving experience. However, the limited computational and storage resources in vehicles brings about a challenge of running computation and data intensive tasks of such applications in the vehicle's on-board unit (OBU). Moreover, embedded applications may not be easily updated by replacing hardware as upgrades in the vehicle OBUs can only happen over each vehicular life-cycle, which is of the order of 10-15 years. The advent of connectivity of vehicles to the Internet offers the possibility of offloading computation and data intensive tasks from the OBU to remote cloud servers for efficient execution. In this paper, we propose a novel architecture for bringing cloud-computing to vehicles where applications embedded in the vehicle OBU can benefit from remote execution of tasks provided as services in the cloud. We design a framework to identify and adaptively manage offloading of computation and data intensive tasks from the vehicle OBU to the cloud during application run-time. Through experimental evaluation using a preliminary prototype implementation of two computer vision applications that use our framework, we show that our approach can provide at least 3x reduction in the end-to-end application response time.

Categories and Subject Descriptors

C.5.0 [**Computer Systems Organization**]: COMPUTER SYSTEM IMPLEMENTATION—*General*

Keywords

Cloud; Offloading; Service-Oriented; Vehicular; Adaptive; Computer Vision

1. INTRODUCTION

Vehicles have long used embedded systems to control car functions (brakes, airbags), but these resources are being expanded to include new class of interactive applications (apps), such as gesture controls, traffic light detection, path mapping, etc. Such applications, in general, can be computation and/or data intensive. For example, vision based applications may involve heavy image processing and machine learning tasks, or applications may fuse multiple sensor information thus requiring access to large storage, or they may require access to a large database. The limited computation and storage resources in vehicles present a challenge to execute such computation and data intensive tasks within the vehicle's OBU. In addition, unlike phones and other mobile devices that get updated almost every 2 years, a vehicle's typical lifespan is about 10-15 years. Hence, OBUs may not have the latest software and hardware updates to sustain the increasing demands of applications over a long life-cycle.

A potential solution to address the challenges in embedding applications in vehicle OBUs is to offload the computation and data intensive tasks to the cloud. The advent of Internet connectivity to vehicles has opened up possibilities for access to remote computing servers and large amount of data in the cloud. There is already a growing interest from automotive [11] and insurance companies [3] for cloud access in vehicles to store telematic data, for emergency assistance and even logging driving behavior. Cloud can relieve vehicles from the heavy information technology resources required to execute computational and data intensive tasks. It also provides an elastic computation facility that can handle long vehicle product life cycle and thus support latest computation-intensive applications in the vehicle.

Challenges in cloud computing for vehicles. The key benefits of cloud computing in vehicles comes with its own set of system design challenges: (i) the offloading must work across heterogeneous environments such as different software and hardware architectures in the OBU and cloud, driving environments, and wireless networks, (ii) the offloading must be adaptive to the diversity of network characteristics and other parameters such as task priority, and (iii) the offloading of tasks to the cloud must happen in a seamless manner during application execution in the OBU. In addition, the system must meet the strict requirements of applications without compromising the driving experience and the overall functioning of the vehicle.

Bringing cloud services to vehicles. To address the cloud computing challenges in vehicles, we design a system that can offer the benefits of cloud computing to vehicular applications in the form of cloud services. We propose a novel *service-oriented* design where individual computation and data intensive tasks of applications are offloaded from the vehicle OBU for remote execution in a cloud server. Our design envisions that the cloud server provides the remote execution of each task as an individual software *service* to the requesting application in the vehicle. In this way, the offloading of tasks can be developed entirely in software – as libraries that can be referenced by the applications. Our proposal provides a paradigm shift from traditional cloud-offloading approaches (inspired from

MCS'15, September 11, 2015, Paris, France.
© 2015 ACM. ISBN 978-1-4503-3545-4/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2802130.2802131.

the mobile cloud computing community) that migrate custom code in virtual machine (VM) or execute parts of the application code in remote run-time compilers. The service-oriented approach can provide robustness to system heterogeneity (compatibility across different cloud architectures and parameter differences in systems) and reduction in network cost; the communication overhead can be significantly reduced as the network transactions primarily will include only data associated with the task and not its entire code or machine state.

Broadly, the goal of our work is to provide cloud computing services to vehicular applications. In this preliminary work towards our goal, using a representative example of interactive vehicular applications, we present the following contributions in this paper:

- We propose a service-based architecture for providing cloud computing services to vehicular applications. We design a framework for making adaptive decisions for offloading application tasks to the cloud.
- We prototype an end-to-end cloud offloading system that uses our service-driven framework. We implement two vehicular use-case applications (hand-gesture recognition and traffic signal light detection) on a mobile device that runs Android and demonstrate our framework of adaptive offloading to a private cloud server.
- We experimentally evaluate our prototype system in real world vehicular settings and show at least 3x reduction in application end-to-end response time through cloud offloading, when compared to executing it entirely on the mobile device.

The rest of the paper is organized as follows: Section 2 motivates the case for offloading vehicular applications; Section 3 introduces our proposed service-driven approach and highlights the key design challenges, followed by the system design details in Section 4; Section 5 describes our prototype system implementation and Section 6 discusses experimental evaluation and results; Section 7 outlines the related work in cloud-offloading and Section 8 concludes the paper.

2. THE CASE FOR OFFLOADING VEHICULAR APPLICATIONS

With the growing need to reduce driver distractions and enhance safety, there is a growing interest in making the embedded applications more interactive. The key notion of interactive applications is to use information from the environment and/or from the user to output an accurate and timely feedback to the user and/or environment, including the vehicle. Vehicle hardware and software resources get updated only with the lifespan of the vehicle (10-15 years), thus making it imperative for embedded vehicular applications to function over long life cycles. To ensure high reliability and robustness over long life-cycles such applications are becoming heavily data-driven and involve complex functions. For example, drivers may interact with the car controls – without requiring to view the dashboard console – using motion gestures identified through information from multiple sensors; integrated cameras in vehicles can help warn the driver of a red signal or traffic sign ahead through automatic detection and recognition of the signals.

Vehicle OBUs are traditionally limited in CPU cycles as well as storage resources, and hence running complex tasks locally in the OBU presents a significant challenge. Motivated by the connectivity of vehicles to the Internet, a viable alternative is to offload such complex tasks to remote cloud servers (Figure 1). Cloud can

How?
offload computation and data intensive tasks to cloud
extendable resource
reduced 'built-in' IT dependency
Users can avail an 'unlimited' number of Apps
Possible?
Connecting vehicles to the 'network' (LTE/WiFi/DSRC) is already near-future
Challenges?
what to offload?
how to offload dynamically?

Figure 1: Motivation for cloud computing in vehicular applications

provide an elastic computation and storage resource to the vehicle OBUs where complex tasks can be executed in a fast and efficient manner also while providing access to large data and expandable storage space. Unlike vehicle OBUs, the hardware and software resources in the cloud constantly evolve over time. Therefore, keeping applications up-to-date without any major modification to the vehicle unit becomes practical when using cloud computing.

3. APPROACH

We propose to bring cloud computing benefits to vehicular applications through a novel *service-oriented* approach where application task functionalities are provided as services from the cloud. Through this approach applications request for remote execution of computation and data intensive tasks in the cloud using an offloading framework where tasks deemed as *offloadable* will be executed as services provided in the cloud. Hence, by application offloading we essentially mean that, task functionalities provided as services in the cloud can be availed by the application running on the vehicle OBU at execution time. We acknowledge that such service-oriented models have indeed been in existence for a long time in multiple contexts [5]; for example, web-architecture or business logic. However, to the best of our knowledge, we believe that there is no prior work that implements remote services in the context of cloud computing for vehicular applications.

3.1 Advantages over Traditional Offloading Approaches

There are specific point solutions in the mobile cloud computing (MCC) area relevant to migrating tasks from mobile phones to the cloud. However, the mobile phone solutions may not directly translate in application to a vehicular OBU, as the latter can be regarded as a much slower device and the architectures can differ at large. Traditional approaches for cloud offloading in the MCC space primarily fall under two categories: (i) Virtual Machine (VM) approach [2, 7], where VMs that run custom middleware software, control and manage the offloading, and (ii) (ii) Run-time code execution approach [4, 9], where application code is broken into smaller units/chunks and the code corresponding to the compute intensive tasks are executed in a remote machine.

Running custom VMs or optimized application code can provide huge gains in performance as they can be independently controlled and optimized at the lower levels close to the underlying operating system (OS). However, such techniques tend to be architecture specific and hence limiting application to only homogeneous settings. Moreover, migrating VMs or application code requires transferring

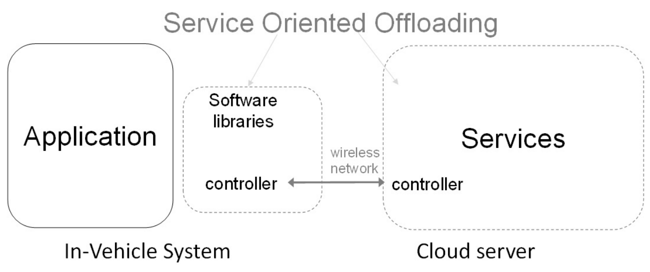


Figure 2: Service Oriented Cloud Offloading Architecture

state and other parameters of the system and application adding extra overhead to the offloaded content.

Our proposed service-oriented approach can provide the flexibility of running services across heterogeneous platforms and devices. This is possible through our approach because, the cloud services and the requesting applications are designed agnostic to the cloud operating system and its processor architecture. Moreover, migrating tasks across systems in the form of software services can be made easy through this approach, making system scaling practically possible.

3.2 Design Challenges

Designing a system for offloading application tasks from the vehicle OBU to the cloud would require addressing the following key challenges:

- Identifying computation and data intensive tasks of an application and marking them as *offloadable*,
- Packaging of tasks for offloading and providing remote execution of offloaded tasks in the cloud,
- Managing offloading of tasks during application run-time adapting to the variable parameters in the system.

Identification of *offloadable* tasks in an application has been extensively studied in the mobile cloud computing community research through application code partitioning strategies. We believe that the theory can be extended to a vehicular use-case. In this paper, we focus our efforts to address the rest of the challenges as portrayed above, that prior work has not addressed, through our system design.

4. SYSTEM DESIGN

We base our system design for offloading tasks from the vehicle OBU to the cloud using our proposed service-oriented approach. As shown in the architectural diagram in Figure 2, our proposed vehicular cloud-offloading system includes a vehicle on-board unit (client) running applications in a supporting operating system and a machine on the cloud (server) that provides application task functionalities as services. The offloading procedure involves the application requesting the server to execute one or more tasks during run-time. The offloading is handled and managed through appropriate software controllers on the client and cloud machine. Tasks are packaged as libraries that get referenced by the application during run-time and which make appropriate function calls to the corresponding tasks executing as services in the cloud. The communication between the vehicle and the cloud is handled through an appropriate wireless network such as cellular or short-range wireles (WiFi or Dedicated Short Range Communication (DSRC)). Our proposed design aims to provide adaptability to the client (vehicular) environment and where the service functions are treated as basic operational semantics. In comparison to a conventional server-client model, our proposed design is more flexible and agile. We will now discuss the aspects of our design that address the key challenges in a cloud offloading system.

4.1 Remote Execution of Tasks

Applications to be run on the vehicle will be divided into individual units called *modules*. We define a module as a block of application source code performing one or more tasks which can be reused independently across different compatible systems. Modules may be executed in sequence or in parallel depending on the application requirements and protocols. The modules are wrapped as a library of functions in a service application programmable interface (API). The service APIs are ported to cloud server machines that execute the functions when requested by the client during application execution. In this paper, we treat only the sequential execution of modules.

Porting modules in the form of API libraries eliminates the need for migrating the entire application/module source code. It also eliminates the need for relaying the application's state during its execution to the cloud. However, the premise is that the functionalities of the modules that can be offloaded are known to the server apriori. This is, however, practically viable, as such information can be embedded statically into the application code during development stage or dynamically when the client registers with the cloud service for the first time.

4.2 Adaptive Task Offloading

The service based offloading mechanism involves the application requesting the cloud to execute one or more modules. Upon acknowledgment, the cloud executes the modules(s) and relays the output back to the client. During offloading the application on the vehicle will require to send only the data corresponding to the requested service (module function) in the cloud. In this way the transactions between the client and the server involve only the data (with some low-payload meta-information) associated with the module to be offloaded. Therefore, the size of the data involved with a particular module in the application is a key parameter in the offloading process. In this regard, a key challenge is to ensure that the networking cost (monetary and latency) for communicating the data is kept minimal during the offload process. However, owing to the variability in the network conditions and the data sizes the offloading processing has to adapt dynamically to such variable parameters in the system. This requires periodic profiling of the system which is handled by the offload controller.

4.2.1 Offload Controller

The offloading controller unit periodically profiles the internal system and application parameters, and performance metrics relevant to the application. This controller may be embedded in the application (code) or can be implemented as a software middlebox within the vehicular OBU. The profiled data is used as data points to develop an optimized strategy for making decisions of which modules should be offloaded during run-time. To cater to the network variability and other parameters of the system, the offloading process must happen during execution (online) and adapt to these internal and external system variables. The challenging aspect of this adaptation is the decision making process during run-time as which modules must be offloaded at a particular instance of time to benefit the system performance. Such online decisions are handled

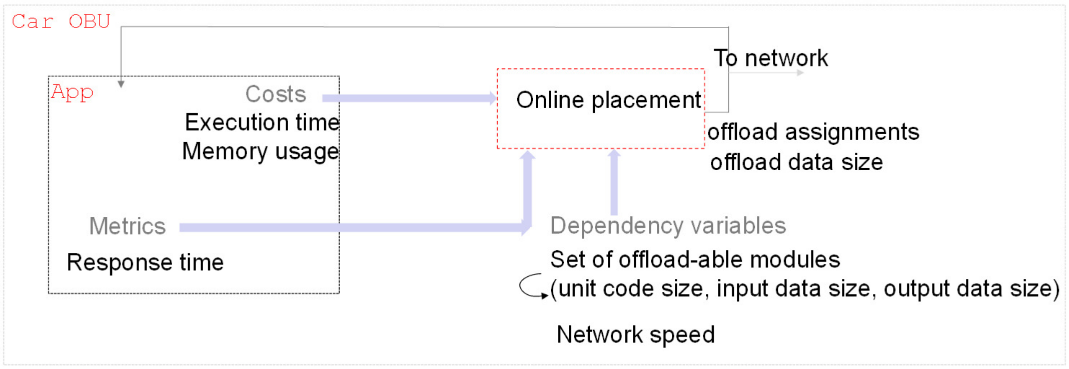


Figure 3: Online-Placement framework in our system design

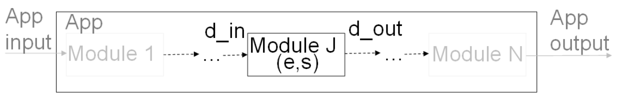


Figure 4: Model for online placement framework

by an adaptive offloading framework that we refer to as *online-placement*.

4.2.2 *Online-Placement Framework*

The goal of the online-placement framework is to make run-time decisions as to what task must be offloaded or placed in the cloud based on different system costs and metrics. We refer to the diagram in Figure 3 to illustrate how the online-placement fits in the context of task offloading in a vehicular system. The framework contains protocols that decide if a module that has been marked as 'offloadable' (whose remote execution can benefit the system significantly) should be offloaded or not when it gets evoked during application execution. The decisions are based on policies that take into account different system parameters, associated data,variables internal and external to the application on the vehicle, and metrics pertinent to the application. In this paper, though we focus only on the end-end response time as the key application metric, however, inclusion of other application metrics in the software framework is indeed straightforward.

The decision making process includes a set of variables and a set of costs. The variables in the offloading system include the set of offloadable modules, network speed, the OBU and server CPU speeds, availability of the server at the instance of offloading, and other optimization parameters if applicable; such as number of threads or processes that can help parallelize the execution. The costs primarily include the system performance metrics such as execution time of each module, their RAM usage, storage space, energy expensed in executing the module. The metrics are constantly evaluated during execution and feedback to improve the decision making.

Since the functioning of the online-placement framework is not specifically tied to the type of application, the framework can adapt to a large spectrum of variables. However, in this preliminary work, we focus only on adaptation to network bandwidth or speed. The network speed is a critical parameter in vehicular use-case as it can vary with vehicular speed; the base station/access point availability can change in each geographical area the vehicle drives into. Interference is another cause for network variability. For example, while urban spots may have good WiFi connectivity the traffic may cause network congestion.

We use the model in Figure 4 to illustrate our proposed framework. The framework executes the adaptive offloading in two phases:

- *Profiling:* The system is profiled for the input (output) data sizes to (from) each module, depicted as d_{in} (d_{out}) with unit as bytes; the network uplink (W_{up}) and downlink (W_{down}) speeds with unit as bytes/sec; the execution time of the module on the vehicle (e) and the cloud (e^*) with units as seconds, and the storage size of the module application code is depicted as s with unit as bytes.

- *Conditioning:* We define a quantity called *Module execution-time ratio*, E, expressed as the ratio of offloading execution time, which is the sum of execution time in cloud and networking time, and the local execution time,

$$E = \frac{e^* + \frac{d_{in}}{W_{up}} + \frac{d_{out}}{W_{down}}}{e} \quad (1)$$

The module is marked for offloading if-and-only-if two conditions are satisfied:

(a). Cloud execution time is less than local execution time; that is, $E < 1$. Essentially, E depicts the (scalar) gain in (reduction of) response time obtained through offloading to the cloud.

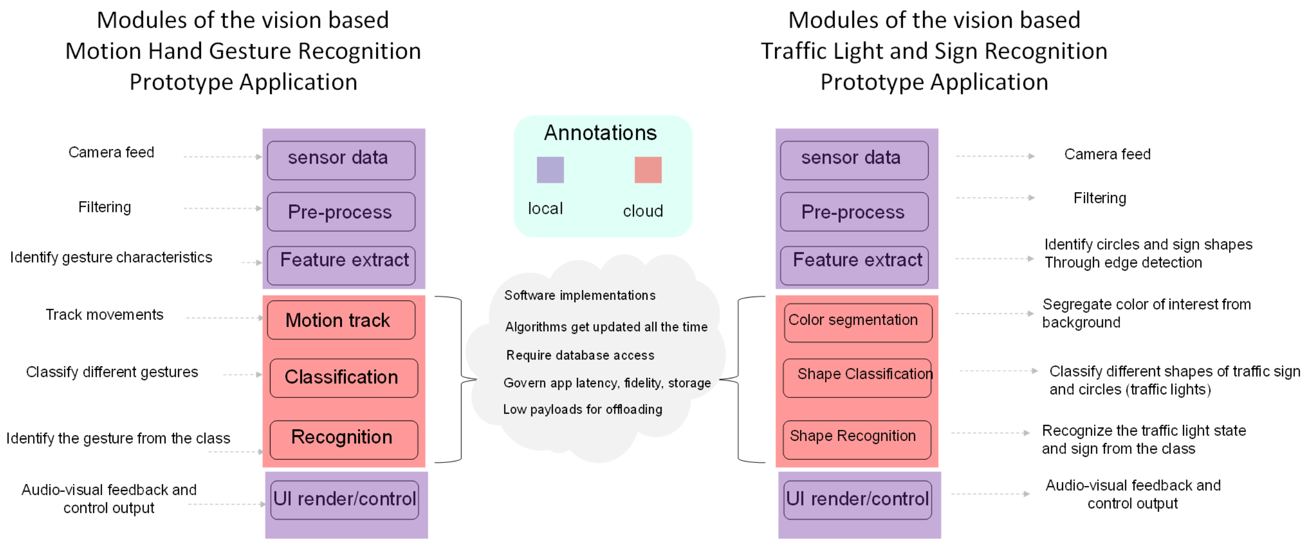
(b). The storage space s for the modules and its associated data does not exceed a threshold set by the local device storage space constraints.

5. PROTOTYPE IMPLEMENTATION

5.1 Cloud Server and Client Infrastructure

We prototyped an implementation of our service oriented cloud offloading system in a real mobile client and cloud environment. Since we could not get access to a real vehicular on-board unit we emulated its functionality in our prototype design through a smartphone device that ran Android operating system; Android runs on Linux kernel similar to a vehicular OBU that runs on an equivalent Linux kernel. We chose to develop in Java as executing Java only requires a Java Virtual Machine (JVM) which is already pervasive across systems today.

We implemented the functionality of modules as services in a cloud machine; in the form of libraries that can be run through a



Modules of the vision based Motion Hand Gesture Recognition Prototype Application
Camera feed
sensor data
Filtering
Pre-process
Identify gesture characteristics
Feature extract
Track movements
Motion track
Classify different gestures
Classification
Identify the gesture from the class
Recognition
Audio-visual feedback and control output
UI render/control
Annotations
local
cloud
Software implementations
Algorithms get updated all the time
Require database access
Govern app latency, fidelity, storage
Low payloads for offloading
Modules of the vision based Traffic Light and Sign Recognition Prototype Application
sensor data
Camera feed
Pre-process
Filtering
Feature extract
Identify circles and sign shapes Through edge detection
Color segmentation
Segregate color of interest from background
Shape Classification
Classify different shapes of traffic sign and circles (traffic lights)
Shape Recognition
Recognize the traffic light state and sign from the class
UI render/control
Audio-visual feedback and control output

Figure 5: Modules of our prototype computer vision based applications

simple function call in the application code. The client and the server machines were interfaced to the network through a TCP socket connection. The services were implemented as Java functions that were developed on a local PC machine and ported to cloud server in the form of Java libraries (.jar) files. We used a private cloud server in CMU (Pittsburgh, USA) campus as our cloud infrastructure. We selected one of the cluster nodes that contained 40 CPU cores and ran Ubuntu Linux 12.04 LTS.

We chose a Nexus 5 device running Android 5.0 as the client device. We developed applications that use our framework for offloading execution of specific modules to the cloud server. In our current implementation, we mark the modules to be offloaded manually in the Android app code.

5.2 Use-Case Applications

We chose to implement two computer vision applications as a test for our prototype framework. Since computer vision applications involve sensing and processing of large volume of data (images), we believe that such applications will provide good litmus tests of our design. Computer vision applications are also inherently interactive as they involve obtaining inputs from the user and/or sensing the physical world. We developed the following applications:

(i) A motion hand gesture recognition system, where a user makes hand movement gestures (move hand, point finger up/down, etc) that are captured by a camera. The gestures are mapped to relevant outputs for in-car usage. For example, waving corresponds to opening the maps app while moving the index finger up/down maps to initiating a phone call.

(ii) A traffic light and sign recognition system, where the camera (pointing to the road) detects the state of traffic light and recognizes traffic signs in its field-of-view.

The modules corresponding to our prototype applications are depicted in Figure 5. We mark these modules as *offloadable* based on an offline profiling of the execution time. In our current implementation we use a heuristic decision making where modules that occupied more than 50% of the execution times but yet carry less than 50% of the total data (input/output) are marked for offloading. Based on manual profiling of the applications, we mark the motion tracking, classification and recognition modules for offloading in the gesture recognition application. We mark the color classification and shape recognition modules for offloading in the traffic light and sign recognition application.

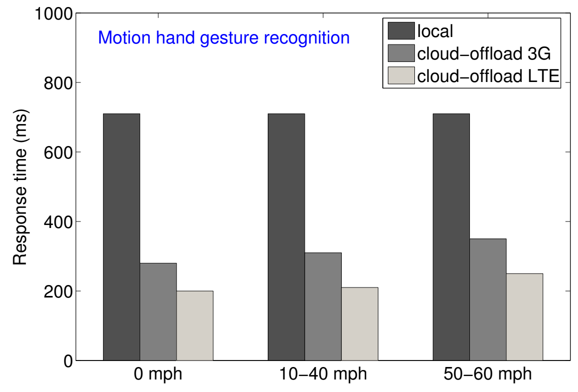


Figure 6: Motion hand gesture recognition: Comparing response time of local v/s cloud-offloaded processing at different vehicular speeds and wireless networks (LTE and 3G)

6. EVALUATION

We evaluate the performance of our service based cloud offloading system in real world vehicular driving environments using our prototype implementation. We particularly evaluate and compare the response time of the application between two cases: (i) *local*, where all the modules are executed locally (on the client device) versus, (ii) *cloud*, where only the modules marked for offloading were executed in the cloud, while the others ran on the client device. The offloading is conducted over LTE or 3G wireless.

Experiment setup. Our set up consists of the phone (client) attached to the dashboard of a car and with cellular data connection (LTE/3G) feature turned ON. The cloud server was wired to the Internet through the CMU campus network service. Both, the hand gesture recognition and the traffic light/sign recognition appli-

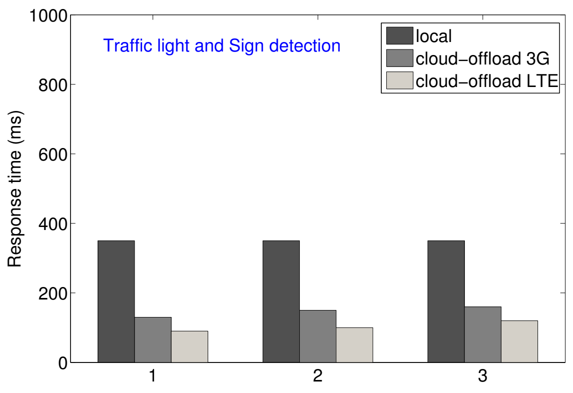


Figure 7: Traffic light and sign recognition: Comparing response time of local v/s cloud-offloaded processing at different vehicular speeds and wireless networks (LTE and 3G)

cations were developed as individual Android apps on the device. Each module of the two applications (implemented in java) were executed in the CMU private cloud server. The network interface for the client and server was maintained through a TCP socket connection. The data transactions were carried in the form of JSON (javascript object notation) strings.

Methodology. Over each trial of the experimentation, the app was fired on the phone, upon which the app relayed a camera preview on the phone screen. Upon a manual button press trigger on the screen the app started executing the application. We use the manual trigger to mark the start time of app execution. Based on our online-placement strategy the offload controller in the app conducts the offloading of the module(s) if only it is beneficial; that is, the module execution ratio (see equation 1) less than 1. We measure the response time as the time duration between the app triggering and the instance at which the app generates a valid output. We conducted the trials in different vehicular speeds and for LTE and 3G connections. We repeated the trials for a total of 50 times conducted over a period of 4 days. All the trials were conducted during day time between 11am and 5pm within 10 mile radius of the CMU campus area.

App	Local	Network	Cloud
Gesture	50	40	10
Traffic	54	36	8

Table 1: Breakdown of average response time in each application (in percentage)

6.1 Response Time Performance

In Figures 6 and 7, we compare the average app response time of local execution versus cloud offloading (over LTE or 3G) using our prototype system. We report the response time in three different vehicular driving environments where the speed of the vehicle is maintained as the variable: car parked in the CMU parking lot (within 500m of main building), car driven on a local street within a 5 mile radius at speeds of 10-40 miles/hour, and the car driven on a highway in a 10 mile radius at speeds pf 50-60 miles/hour.

We can observe from our results that, overall, a response time reduction of about 3x is being achieved through our cloud offloaded system. We also observed that the overall average response times are maintained within 250ms. At 60 mph a car displaces only about 5m in 250ms. Such response times seem practically reasonable for maintaining the interactivity requirement of these applications. We also observe that, for both applications, the cloud offloading response times monotonically increase with the speed of the vehicle. This is more likely caused by the cellular connection's inadequacy of handling Doppler effects and resulting in throughput reduction at higher speed.

In Table 1 we report the breakup of the response time in terms of the local module's execution times, the networking time, and the cloud execution time of the offloaded modules. We can observe that, the cloud execution constitutes the smallest percentage of the 'delay' and the networking time significantly governs the response time. Networking time depends largely on the size of the data communicated over the network. From our analysis of the experimental data we observed that the maximum payload data size during offload was less than 300kB. Clearly, offloading such data sizes would be more preferable than the entire image (at least an order of magnitude larger in size) as flaky cellular networks can result in high networking delays for such data sizes and thus nullifying the gains obtained through cloud computing.

We note, from our experiments, that our prototypes achieved a median 90% accuracy of recognizing correct gesture or traffic light state/sign. In this paper, we have been less concerned with improving the accuracy of a computer vision application and instead have focused on evaluating the response time gains of offloading the computation to a cloud server. However, maintaining high accuracy is also an important requirement in vehicular interactive applications. We believe that one possible way to improve accuracy in our future prototypes would be to use large sample data sets for classification and recognition databases on the cloud. However, this may increase the overall response time presenting an interesting trade-off between response time and accuracy. However, we believe that cloud computing can help bridge the gap between the response time increments and accuracy. Processing times can be significantly minimized as executing such computation and data intensive tasks on a highly efficient cloud machine can provide surplus benefits than executing them locally in the vehicle OBU.

7. RELATED WORK AND DISCUSSION

Computational offloading from mobile client devices to cloud, or mobile cloud computing (MCC) has been a topic of extensive research for over a decade. The recent years have also seen MCC work in the context of sharing computational load with other devices in vicinity through cyber-foraging [1] or even bringing the cloud closer to the mobile device through cloudlets [7]. However, the work has primarily been in the context of offloading computation from smartphone or laptop devices, and not vehicular use-cases.

MAUI [4] offloads application code at run-time to a remote machine that hosts a run-time compiler. The disadvantage of this approach is the need for a custom run-time compiler or middleware that interfaces to the remote server or cloud. In addition, MAUI uses an integer programming solver for partitioning code at run-time which not only adds computation overhead but also may not scale well to cater to multiple applications in vehicular use-cases. CloneCloud [2] proposes to optimize computation on remote machines by virtualizing the entire mobile device on the cloud machine. ThinkAir [9] proposes to perform the application partitioning at the application layer which may be more suitable for a vehicular use-case. However, the particular implementation requires cloning the entire mobile device in the cloud (by creating and destroying customized VMs at run-time), which does not translate well when a vehicle is the mobile device. Adding virtual machines and/or migrating them during run-time is tedious and in the heavily resource constrained vehicular system. COSMOS [13] proposes to manage task allocation and offload to cloud instances running on virtual machines. While the approach comes close to the idea of using cloud services, the implementation requires managing the COSMOS virtual machines developed over Android x86 platform and also customizations to the application code as well. Odessa [12] improves response time of perceptive applications by improving parallelism in the local and remote computation. However, the implementation requires heavy customizations and may not scale well in vehicular use case. Other approaches such as COMET [6] that uses shared memory resources, Cuckoo [8], are in general heavy and not easily flexible to adapt to the dynamism in the vehicular system. The need for the day is a light-weight portable and adaptive cloud offloading system. Carcel [10] comes the closest to designing a cloud offloading system for vehicular use case, in particular for autonomous driving. Carcel does not perform computational offload, however, enables the cloud to have access to sensor data from autonomous vehicles as well as the roadside infrastructure for better path planning. Our work takes a step further to design practical system for offloading computation from applications during run-time and making it adaptive to the dynamics of the vehicular environments.

8. CONCLUSION AND FUTURE WORK

We proposed a new architecture for offering cloud services to vehicular applications through a service oriented approach. Our proposed design aims to run vehicular application tasks as services in the cloud. We designed a system framework to seamlessly offload computation of application tasks at run-time to a cloud server. We implemented the application tasks as services in a private cloud that any application on the client can avail. We designed an adaptive offloading framework where computation and data intensive tasks are offloaded adapting to the network conditions during application execution in real vehicular driving environments. We showed a proof of concept implementation and conducted preliminary experiments to show the feasibility of our service based offloading approach for adaptive cloud offloading. Through experimental evaluation in real vehicular environments we showed that cloud offloading through our design can provide significant gains in the application response time.

Our current implementation required that the modules to be offloaded are statically marked by the developer during development. In addition we also considered that the execution time and storage cost on the local machine/cloud for each module are known apriori, rather than dynamically profiling the same. We aim to address these challenges in our future implementations by incorporating dynamism to the system profiling techniques.

9. REFERENCES

[1] Rajesh Krishna Balan. *Simplifying Cyber Foraging*. PhD thesis, Carnegie Mellon University, 2006.

[2] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, pages 301–314, New York, NY, USA, 2011. ACM.

[3] COMPUTERWORLD. Insurers will now be able to track driver behavior via smartphones. `http://tinyurl.com/ojf9hrr`.

[4] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: Making smartphones last longer with code offload. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[5] Thomas Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Prentice Hall, 4 2004.

[6] Mark S. Gordon, D. Anoushe Jamshidi, Scott Mahlke, Z. Morley Mao, and Xu Chen. Comet: Code offload by migrating execution transparently. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pages 93–106, Hollywood, CA, 2012. USENIX.

[7] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. Just-in-time provisioning for cyber foraging. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 153–166, New York, NY, USA, 2013. ACM.

[8] Roelof Kemp, Nicholas Palmer, Thilo Kielmann, and Henri Bal. Cuckoo: A computation offloading framework for smartphones. In Martin Gris and Guang Yang, editors, *Mobile Computing, Applications, and Services*, volume 76 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 59–79. Springer Berlin Heidelberg, 2012.

[9] S. Kosta, A. Aucinas, Pan Hui, R. Mortier, and Xinwen Zhang. Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading. In *INFOCOM, 2012 Proceedings IEEE*, pages 945–953, March 2012.

[10] Swarun Kumar, Shyamnath Gollakota, and Dina Katabi. A cloud-assisted design for autonomous driving. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 41–46, New York, NY, USA, 2012. ACM.

[11] OnStar. OnStar by GM. `https://www.onstar.com/us/en/home.html`.

[12] Moo-Ryong Ra, Anmol Sheth, Lily Mummert, Padmanabhan Pillai, David Wetherall, and Ramesh Govindan. Odessa: Enabling interactive perception applications on mobile devices. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 43–56, New York, NY, USA, 2011. ACM.

[13] Cong Shi, Karim Habak, Pranesh Pandurangan, Mostafa Ammar, Mayur Naik, and Ellen Zegura. Cosmos: Computation offloading as a service for mobile devices. In *Proceedings of the 15th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, MobiHoc '14, pages 287–296, New York, NY, USA, 2014. ACM.