CloudSLAM: Edge Offloading of Stateful Vehicular Applications

Kwame-Lante Wright*, Ashiwan Sivakumar*†§, Peter Steenkiste*, Bo Yu‡, Fan Bai‡
*Carnegie Mellon University, †AT&T Labs Research, ‡General Motors

2020 IEEE/ACM Symposium on Edge Computing (SEC) | 978-1-7281-5943-0/20/$31.00 ©2020 IEEE | DOI: 10.1109/SEC50012.2020.00018

***Abstract*—**
Vehicular applications are becoming increasingly complex and resource hungry (e.g. autonomous driving). Today, they run entirely on the vehicle, which is a costly solution that also imposes undesirable resource constraints. This paper uses Simultaneous Localization and Mapping (SLAM) as an example application to explore how these applications can instead leverage edge clouds, utilizing their inexpensive and elastic resource pool. This is challenging as these applications are often latency-sensitive and mission-critical. They also process high-bandwidth sensor data streams and maintain large, complex data structures. As a result, traditional offloading techniques generate too much traffic, incurring high delay. To overcome these challenges, we designed CloudSLAM. It partitions SLAM between the vehicle and the edge. To manage the complex, replicated SLAM state, we propose a new consistency model, Output-driven Consistency, that allows us to maintain a level of consistency that is sufficient for accurate SLAM output while minimizing network traffic. This paper motivates and describes our offloading design and discusses the results of an extensive performance evaluation of a CloudSLAM prototype based on ORB-SLAM.

***Index Terms*—simultaneous localization and mapping, connected vehicles, edge computing**

I. Introduction

Modern vehicles are increasingly relying on IT technology for tasks such as enhancing driver experience, driver assistance, and even safety-critical applications. These applications typically run entirely on the vehicle, requiring a substantial in-vehicle IT infrastructure. In-vehicle execution has the advantage that (1) applications are simple to develop and (2) always work with predictable response time without needing wireless connectivity. However, this approach is not sustainable for several reasons. First, the on-board IT infrastructure increases the complexity and cost of the vehicle. Second, while mobile devices (e.g. smartphones) are updated every few years, vehicles have a typical lifetime of 10-15 years. This can result in outdated computing infrastructures that cannot support evolving application requirements.

Cloud offloading, which has been studied extensively for mobile devices [1], [2], [3], is an obvious alternative. While this is possible for non-mission-critical infotainment applications [4], [5], more critical applications like autonomous driving use a multitude of sensors (e.g. video cameras, LIDAR, radar) that generate large volumes of data. However, cellular bandwidth is a limited and expensive resource. In addition, these applications are often latency sensitive, typically requiring response times in tens or a few hundreds of milliseconds. Due to these issues, traditional offloading techniques are not applicable since the large sensor inputs would result in very high network loads and excessive delays.

In this paper, we propose an architecture for offloading sensor-rich, real-time vehicular applications, using Simultaneous Localization and Mapping (SLAM), specifically ORB-SLAM, as an example application. SLAM is a widely used technique in autonomous driving systems. Inspired by previous research [3], [4], [5], we investigate an offloading approach in which we *partition* the automotive application between the vehicle and the edge. In our approach, called *CloudSLAM*, we execute latency-sensitive SLAM tasks on the vehicle to ensure their timely completion, while less latency-critical tasks are delegated to an edge cloud. In this design, offloaded tasks only use pre-processed sensor data, thus reducing communication costs and latency. Using the edge also assists with latency by providing for low, predictable latency [6], [7], [8].

While function partitioning is an attractive and viable solution, it raises the question of how the distributed tasks share information. Applications such as SLAM maintain a large state map that is used and updated by all the tasks throughout the trip. This state map must be replicated across the vehicle and the edge, but using traditional methods to ensure consistency is impractical because of the high frequency and complexity of the updates. Previous works have focused on partial replication [9] and differential updates [10]. However, to address this challenge, we note that the *only* output of ORB-SLAM is the *pose* (location and orientation) of the vehicle. The ORB-SLAM state is used to compute the pose but the state itself is not an output. Based on this observation, we propose output-driven consistency, a new, relaxed consistency model to manage consistency for this type of application. Output consistency keeps the vehicle and edge state sufficiently consistent for the vehicle to calculate accurate output poses. Our implementation of output-driven consistency minimizes bandwidth use by adapting the rate of state updates while assuring the quality of SLAM's pose output.

This paper makes the following contributions:

- We define an offloading architecture for sensor-driven, stateful vehicular applications based on partitioning;
- We introduce the concept of output-driven consistency, a relaxed state consistency model in which the degree of consistency is driven by the desired quality of the output of the application;
- We describe CloudSLAM, a distributed implementation of ORB-SLAM that applies output-driven consistency,

§Work done when author was affiliated with Carnegie Mellon University

978-1-7281-5943-0/20/$31.00 ©2020 IEEE
DOI 10.1109/SEC50012.2020.00018

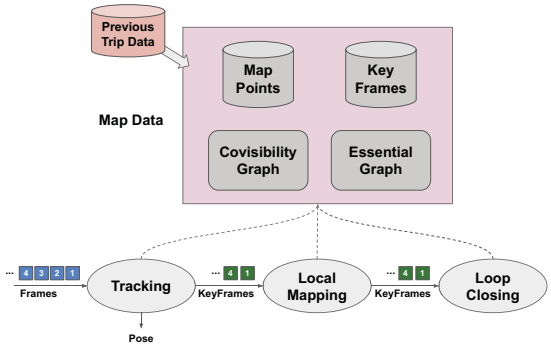


Fig. 1. System Overview of ORB-SLAM

focusing on the mechanisms that adapt the bandwidth utilization based on the desired output quality;
- We present an extensive evaluation that shows the effectiveness of the output-driven consistency adaptation mechanisms and their impact on the accuracy.

The rest of the paper is organized as follows. Section II gives an overview of ORB-SLAM. We present a high-level CloudSLAM design in Section III and elaborate on how we manage the replicated SLAM state using output-driven consistency in Section IV. In Section V we discuss how we balance network use and SLAM accuracy. Cloud-vehicle coordination is described in Section VI and Section VII presents our evaluation results. Finally, we discuss related work and conclude in Sections VIII and IX, respectively.

II. ORB-SLAM

Simultaneous Localization And Mapping (SLAM) is a technique for estimating the pose of a vehicle (location and orientation). It is a critical component of autonomous driving systems [11]. In SLAM, precise localization is achieved using a feature map of the traversed environment that is constructed and updated throughout the trip using inputs from sensors such as a stereo camera or LIDAR. In this paper we use ORB-SLAM [12], specifically ORB-SLAM2 [13], as an example of a sensor-driven, stateful vehicular application. It is a state-of-the-art implementation of visual SLAM that has high accuracy and good performance. In this section, we provide the background needed on ORB-SLAM to understand our offloading design.

ORB-SLAM consists of three main modules, namely Tracking, Local Mapping, and Loop Closing, as shown in Figure 1. These modules run as separate threads on a single processor.

1) Tracking: The Tracking module uses a stream of stereo frames to incrementally localize the camera on a frame-by-frame basis. This is accomplished by extracting unique features from the images and establishing *map point* correspondences between frames. Map points are points in space that are believed to have a fixed position. By comparing the locations of these map points in consecutive frames using the map data structure described below, it is possible to reason about the motion of the camera and to incrementally estimate its pose. The Tracking module also determines when to generate new *keyframes*. Keyframes are frames that contain enough new information to be worth saving and they are further processed by the Local Mapping module.

2) Local Mapping: The Local Mapping module is responsible for creating and maintaining a model of the environment. Local Mapping takes as input the keyframes that are forwarded by the Tracking module and builds a set of data structures that are used to calculate an accurate pose output. First, it stores the keyframes and their associated camera pose. Second, it keeps track of map points which are features tracked across multiple frames. The map point information includes a features descriptor, its 3D position, and the mean moving direction across the frames. Finally, the Local Mapping module also builds and maintains a set of graphs that link keyframes based on the map points they contain. These data structures (keyframes, map points, and graphs) constitute what we call a *map*.

As new keyframes arrive, the Local Mapping module also performs a small-scale optimization step referred to as a *local bundle adjustment* or local BA. Local BA uses the information provided by each new keyframe to improve the position estimates of previous keyframes in the spatial locality of the new keyframe. Consequently, this improves the map in general which will improve the accuracy of future pose outputs as well.

3) Loop Closing: The Loop Closing module determines whether the camera has revisited a previous location. For example, imagine an employee who commutes to an office building for work. At the end of the day she commutes back home, thereby "completing a loop" by revisiting the place where she started. This information can be used to improve the quality of the map using a technique called *global bundle adjustment* or global BA. Global BA is based on the observation that the start and end points of the loop must be in the same location. However, SLAM's pose output for the start and end points may in fact be different since Tracking and Local Mapping alone may result in significant accumulated *drift*, an intrinsic problem of visual odometry. The Loop Closing module can reduce the effect of drift by forcing the start and endpoints to be co-located and by correcting the pose of all keyframes and map points in the map accordingly.

4) Using previous trip data: Note that global BA is performed after the loop is completed, so it cannot go back in time and improve the accuracy of the pose output during the trip itself, what we refer to as the live trajectory. However, the map that was optimized using global BA can be used by ORB-SLAM to improve the accuracy of ORB-SLAM in future trips. The idea is simple: when a driver starts a new trip, instead of starting with an empty map, the map generated during earlier trips can be used as the initial map for the new trip. Since it has benefited from global BA, this map that contains very accurate information on map points that can improve the accuracy of the pose output. When using a map with previous trip data, Local Mapping continues to apply local BA to further improve the pose output. Figure 2 shows an extreme example for a very

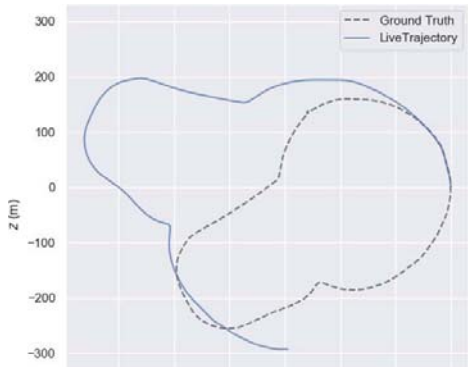


Fig. 2. ORB-SLAM output with and without access to previous trip data

challenging loop to illustrate the importance of using a map with previous trip data. We see that without the benefit of loop closure or previous trip data, the ORB-SLAM output suffers from significant drift. Note that while loop closing *at the end* of the loop will significantly improve the accuracy, the result is unacceptable *during* the trip itself.

III. CloudSLAM Overview

While it is possible to run ORB-SLAM entirely on the vehicle, this has several disadvantages. First, embedding high-performance compute infrastructure in a vehicle is complex and costly. Second, while one can limit cost by simplifying the SLAM code, sensor-driven applications such as SLAM and object tracking typically involve trade-offs between compute resources and result quality. Resource constraints on the vehicle can affect the quality of the output, and loss of accuracy will get worse as applications become more sophisticated. Finally, a map with previous trip data will generally become large enough to make storage on a vehicle impractical. For example, using the KITTI dataset [14] as a reference, we estimate that it would take on average 200 MB of memory per mile of road to store an ORB-SLAM-compatible map. With 6000 miles of roads, it would take 1.2 TB of memory to store a map of New York City. This map is critical to achieving good accuracy and will need to be constantly updated.

In this section, we describe the high-level design of CloudSLAM, a modified version of ORB-SLAM that supports real-time offloading of SLAM processing to the inexpensive, elastic resource pool available in the cloud. Since SLAM output is time-critical, we focus on the use of an edge cloud. We identified the following goals for CloudSLAM. The primary goal is reducing the CPU and memory requirements on the vehicle. Second, CloudSLAM's accuracy should be similar to that of ORB-SLAM running on the vehicle, both in terms of accuracy and timeliness of pose updates. Finally, we want to minimize the use of the network since cellular bandwidth is limited and expensive.

A. Offloading versus Partitioning

The simplest way to leverage the cloud is to simply run ORB-SLAM on an edge server. The vehicle can send the stereo video stream to the edge, which computes the pose using a map with previous trip data to achieve high accuracy, and returns the pose to the car. The problem is that this approach requires significant cellular bandwidth. Not only can this design be expensive (in monetary cost), but the cellular bandwidth is limited and variable, which can lead to long transfer times for video frames, delaying the pose outputs and affecting accuracy.

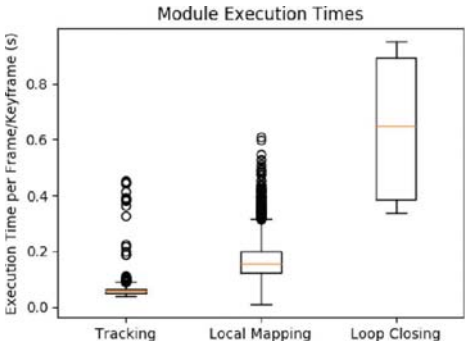


Fig. 3. ORB-SLAM execution times for Tracking, Local Mapping, and Loop Closing for the KITTI-05 trace. This box plot shows the five-number summary of statistics for each of the modules (circles are outliers).

(a) ORB-SLAM's requirements per module

Module	Latency Constraint	Frequency
Tracking	Real-time	every frame
Local Mapping	Real-time	every keyframe
Loop Closure	No exact deadline	once per loop

(b) ORB-SLAM's average performance on KITTI-05

Module	# of Frames	Avg. Time (s)
Tracking	2761	0.058
Local Mapping	725	0.168
Loop Closure	3	0.644

TABLE I
LATENCY CONSTRAINTS AND EXECUTION TIME OF ORB-SLAM MODULES

In order to better understand the computational requirements of ORB-SLAM, we measured the execution times of the three ORB-SLAM modules on one of the traces of the KITTI visual odometry dataset [14]. We conducted this test on a workstation computer running an Intel i7-6700K 4GHz processor with 16 GB of RAM. Figure 3 shows the execution times for the modules, while Table I provides more information on the properties of each module. The results are shown per processed frame and averaged across five runs. Each run on average processes 2761 frames, 725 keyframes, and 3 keyframes in the Tracking, Local Mapping, and Loop Closing modules, respectively. Note that while ORB-SLAM is multi-threaded, execution is sequential on a single core.

B. Module and Map Placement

The above results suggest that CloudSLAM should use partitioning, with the Tracking and Local Mapping modules executing on the vehicle, and the Loop Closing module executing in the edge. The Tracking and Local Mapping modules

have relatively low execution times and, while there are some outliers, their execution times are fairly predictable (Figure 3). In addition, as shown in Table I, both must execute in real-time since they generate the pose output that is used by applications running on the vehicle. Finally, they process either all the frames (Tracking) or keyframes (Local Mapping) and sending all of frames to the edge consumes a large amount of bandwidth and delays the pose output.

In contrast, the Loop Closing module (when fully executed) has a longer execution time that is less predictable since it depends strongly on the size of the map, i.e. the length of the loop at the time it is triggered. This makes it a good candidate for offloading. Loop Closing is also less time critical than Tracking and Local Mapping, so we can afford the extra latency associated with remote execution. However, as we will show in our evaluation, it is critical that we offload to an edge cloud as opposed to an arbitrary server.

Finally, due to its size, the map with previous trip data is only stored in the edge. We refer to this map as the *global map* and refer to the map on the vehicle as the *local map*. The local map only includes information relevant to the current trip.

This high-level design helps address many of our design goals. However, this design involves three significant challenges:

- **State management:** Since all modules read and write to the complex map data structure, how do we manage the consistency of the copies of the map on the vehicle and in the edge?
- **Limiting bandwidth use:** How do we further limit network bandwidth use since this design still requires keyframes to be sent to the edge?
- **Maintaining accuracy:** The difficulty of generating an accurate pose depends heavily on the mobility and physical environment. How do we maintain accuracy given that both can change significantly during a trip?

In the next two sections we describe how we address these challenges using a set of mechanisms that result in the CloudSLAM design shown in Figure 4.

IV. Managing State

ORB-SLAM was designed to run on a single processor and all modules expect to operate on a single copy of the map data structure. As described previously, the local and global maps in CloudSLAM are very different since the global map is initialized with previous trip data while the local map starts from scratch each trip.

A. Consistency Requirements

From our initial experimentation with ORB-SLAM, we realized that maintaining strict state consistency between the local and global maps is not necessary. What we really care about, for any SLAM system, is the pose output. While the accuracy of the pose output depends on the accuracy of the map, small differences in map state will have a limited impact on the pose output.

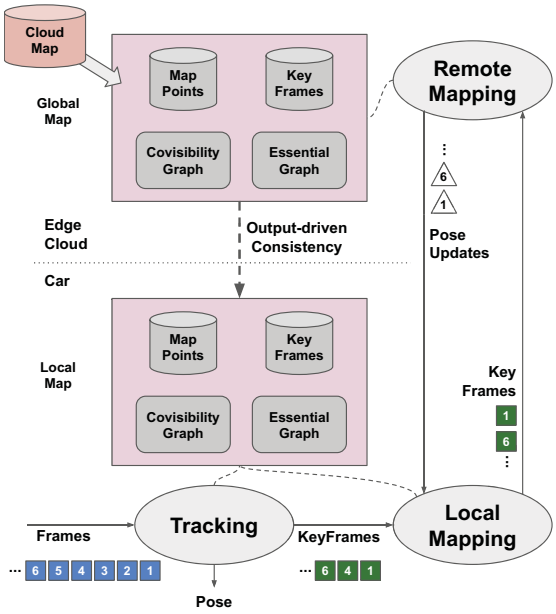


Fig. 4. CloudSLAM vehicle-edge partitioning

This is acceptable for two reasons. First, the execution of ORB-SLAM is not repeatable due to its multi-threaded design [13], so two executions of ORB-SLAM using the same video input will generate a slightly different trajectory each time due to the non-deterministic scheduling of the modules. Secondly, the construction of the map is based on sensor data, which itself can be noisy. Given this, it is not realistic to expect that the CloudSLAM output will be exactly identical to the output of a corresponding ORB-SLAM execution. However, they can be kept similar and we observe that occasional updates are sufficient to accomplish this. This suggests that in CloudSLAM, differences in the local and global maps are acceptable and strong consistency is not a requirement. What matters is that the local and global maps must both be accurate enough to generate an accurate pose compared with ORB-SLAM. SLAM output always has some error and the amount of error that is acceptable depends on the application that uses the output.

B. Local and Global Map Consistency

Based on the above observations, we implement a relaxed consistency model that focuses on the accuracy of the pose output instead of the consistency of the state. We refer to this as output-driven consistency and it is discussed in more detail below. Supporting output-driven consistency while maintaining pose accuracy with limited network bandwidth requires several new mechanisms, resulting in the design in Figure 4.

First, instead of using the original Loop Closing module which performs an expensive global optimization over the entire map, the edge server processes the keyframes using the more efficient *Remote Mapping* module that we developed, reducing the response time. It uses the same loop closing

function, but only applies the optimization to a smaller number of frames that are relevant to the keyframe being processed. Since the global map was initially created by using loop closing on loops from previous trips, we found that Remote Mapping can still generate an accurate pose output using a more constrained set of matched keyframes. In addition, while other works send a new local map [9] or map updates [10] to the vehicle, the edge server in CloudSLAM only sends a pose update to the vehicle, reducing communication overhead. On the vehicle, the updated pose is used by the Tracking module to calculate future poses, but also by the Local Mapping module to back-propagate the correction and improve the accuracy of the local map by adjusting the map within a small time window. This is the part of the map that is most critical for future pose calculations.

Second, since CloudSLAM is bandwidth limited, we cannot send all keyframes to the edge during a trip, so CloudSLAM only sends a subset of keyframes. The challenge is that the difficulty of calculating an accurate pose depends on the physical environment and speed. For example, visual SLAM often has trouble with camera rotation in turns [15], increasing the risk of drifts. To deal with this challenge, we adapt the rate at which keyframes are sent to the edge based on an estimate of the pose error relative to ORB-SLAM (Section V).

Third, since CloudSLAM only sends a subset of the keyframes to the edge, the Remote Mapping module cannot fully enhance the global map with the data from loops that are part of the current trip using global BA since it is missing many keyframes. However, loop closing can be done at the end of the trip, when the car has an opportunity to upload more keyframes cheaply. In CloudSLAM, we incorporate the missing keyframes of the current trip in the *cloud map* at the *end of the trip*. The cloud map is archival and it is used to initialize the global map before a trip, so enhancements will be automatically picked up. Also, by delaying the global bundle adjustment until after the trip, we can use high-speed wireless (e.g. WiFi) to transfer keyframes to the edge, so there is no need to use valuable cellular bandwidth during the trip (Section VI). Note that with this design, the global and local map are ephemeral state, so limiting updates to the part of the map that is relevant to future pose outputs is acceptable.

Next, we define output-driven consistency more precisely and describe its implementation in CloudSLAM. We then describe how we adapt the keyframe rate in Section V and how all the mechanisms work together in Section VI.

C. Output-driven Consistency

Based on the discussion above, the only consistency requirement for the cloud and vehicle maps in CloudSLAM is that the pose estimates generated by CloudSLAM are similar in quality to those generated by ORB-SLAM. We refer to this type of consistency as *output-driven consistency*. To use the terminology defined by Aguilera et al. [16], output-driven consistency is a form of operation consistency as opposed to state consistency.

Let us more precisely define output-driven consistency using the following simple formulation. Let x_1 and x_2 represent the state of two nodes in a distributed system, node 1 and node 2, respectively. Let $f(\cdot)$ represent the processing done on such states by some application. For a strict form of state consistency, when there are no outstanding update operations, we get state equality (Equation 1) which yields output equality (Equation 2).

$$x_2 = x_1 \tag{1}$$

$$f(x_2) = f(x_1) \tag{2}$$

In contrast, for output-driven consistency we want the states to be sufficiently similar (Equation 3) so that the difference between the outputs of the function $f(\cdot)$ applied to the states x_1 and x_2 is small (Equation 4).

$$x_2 \approx x_1 \tag{3}$$

$$|f(x_1) - f(x_2)| < \epsilon \tag{4}$$

The goal of CloudSLAM is to generate a pose output that is very similar to that of ORB-SLAM. Of course, we cannot run ORB-SLAM during the trip. However, since the Remote Mapping module running in the edge has access to the global map with previous trip data, we expect that the pose it generates will be very similar to that of ORB-SLAM, assuming it receives enough keyframes. Based on this, we can express the consistency model implemented by CloudSLAM as follows. Consider that the states m_L and m_G correspond to the local and global maps, respectively, and the functions $f_v(\cdot)$ and $f_e(\cdot)$ correspond to the components of CloudSLAM running on the vehicle and edge server, respectively. We want to keep the part of the two maps that impact the pose output qualitatively similar (Equation 5), so that the error of the pose output generated by the vehicle $f_v(m_L)$ is similar to the (estimated) ORB-SLAM pose $f_e(m_G)$.

$$m_L \approx m_G \tag{5}$$

$$|f_v(m_L) - f_e(m_G)| < \epsilon \tag{6}$$

This model can be implemented by having the vehicle monitor the difference between its local pose estimate and the adjustments returned by the edge. This difference can then be used as a metric by which to adjust the rate at which keyframes are sent to the edge, as described in Section V.

We believe that a consistency model that focuses on the output accuracy, rather than state, can be useful in other sensor-driven applications. For example, Google Maps has released an augmented reality (AR) feature [17] that provides directions while users navigate through an environment using their phone's camera. This application applies similar computer vision techniques to that of SLAM and it is also latency-sensitive as well. We believe that enabling the application to dynamically leverage both a local and global map as we do in CloudSLAM can improve its performance.

V. Minimizing Network Use while Maintaining Accuracy

In this section we describe how we balance the trade-off between minimizing bandwidth use and maintaining the accuracy of the pose output. This is done by limiting the rate at which we send keyframes to the edge and by adjusting this rate based on the pose adjustments returned to the vehicle by the Remote Mapping module (Section IV-B).

A. Minimizing Network Bandwidth

We first focus on how we implemented fixed-rate strategies in CloudSLAM. We consider two strategies.

The simplest fixed-rate strategy is a *periodic strategy*, which sends keyframes periodically to the edge server. However, ORB-SLAM's selection process for keyframes is data-driven and unpredictable so we cannot send keyframes with an exact fixed period. We instead approximate periodic transmissions by enforcing a maximum keyframe transmission rate, as opposed to a fixed one, by specifying a minimum time window between consecutive keyframes that are sent to the edge. For example, if the desired period is one frame/second, then after a keyframe is sent to the edge, the next keyframe that is sent is the first keyframe that is generated by the vehicle after one second has elapsed.

We also evaluate a second fixed-rate strategy, called the *distance strategy*, in which we pick keyframes based on a distance threshold, rather than a time threshold. This is based on the observation that when a vehicle is not moving, or moving slowly, it is not necessary to upload as many keyframes to the edge since the content of the video frames changes more slowly compared with a fast moving vehicle. In other words, distance can be a better predictor of how quickly frames change than time. This effect is captured by using a distance metric, i.e. after a keyframe transmission the next keyframe that is sent is the first keyframe generated after the vehicle has tracked itself to have moved at least x meters. We compare both of these strategies in our evaluation in Section VII.

B. Maintaining Accuracy through Adaptation

Sending keyframes to the edge using a fixed strategy is easy to implement once we have the right rate. Picking a rate that is too high will waste network bandwidth while a rate that is too low will result in a loss of accuracy of the pose output and cause drift. In practice, the ideal number of keyframes needed to obtain an accurate output depends strongly on the physical environment. In an environment with many distinct features that can be used by the Tracking module to incrementally generate a new pose, fewer keyframes are needed. The rate at which features change across frames and the type of motion are also factors. For example, pure camera rotation is a difficult challenge for visual SLAM [15].

For the above reasons, CloudSLAM adapts the rate at which keyframes are sent to the edge, but a metric is needed to control the adaptation. We considered two metrics. The first metric is the number of features that match between successive frames. When the number of matches starts to drop, we would expect that the pose estimates will start to diverge significantly from the actual trajectory. The second metric is the magnitude of the pose corrections that the edge returns based on keyframes that were sent up. Since the Remote Mapping module uses a global map based on previous trip data, we expect that its pose estimate will be very similar to that of ORB-SLAM. As a result, the difference between the poses generated by the edge and by the vehicle are a direct estimate of the error in the pose estimates generated on the vehicle.

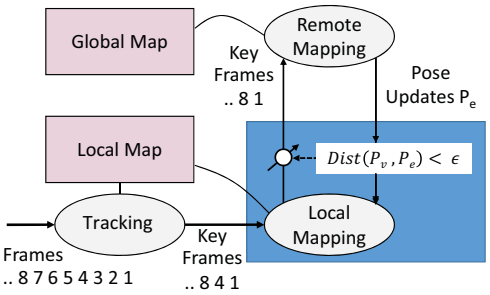


Fig. 5. Adapting the rate of keyframes sent to the edge

In order to evaluate how effective the metrics are in predicting the error introduced by the vehicle's local map tracking, we calculated the error in the pose output of ORB-SLAM without previous trip data (i.e. using just the Tracking and Local Mapping modules, similar to the vehicle) relative to ORB-SLAM's optimized output (i.e. with Loop Closing). For both metrics we found a good correlation. For the feature-based metric, we found that the number of feature matches between consecutive frames is generally lower as the error increases. For the pose adjustment metric, we found that the pose adjustment returned by CloudSLAM is higher for road segments where ORB-SLAM output has higher error. We conclude that either metric could be used but decided to use the pose adjustment metric for two reasons. First, the number of matches between consecutive frames depends strongly on the video trace, making it hard to define an appropriate threshold for adaptation. Second, the pose adjustment metric is a direct estimate of the error in the output of CloudSLAM. This is likely to be more robust than an internal metric (such as feature matches) and it also has the benefit that the adjustment threshold can be intuitively picked by users of CloudSLAM based on their error tolerance, e.g. 2 meters.

Figure 5 shows how the adaptive strategy is implemented. The Local Mapping module compares the pose returned by Remote Mapping with the vehicle's pose estimate and adjusts the keyframe rate accordingly. This design can be used to adapt the rate for both the periodic and distance fixed-rate strategies. As a proof of concept we implement the adaptive strategy on top of the distance metric. We use a decrease multiplier of 0.9 and an increase multiplier of 1.01 for the distance threshold. The motivation is that if the pose correction is high, we want to increase the rate more aggressively to avoid a long-term

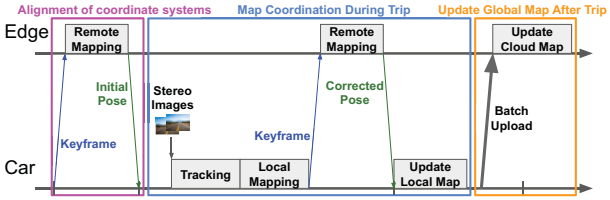


Fig. 6. CloudSLAM timeline showing vehicle-edge interactions

drift. When the correction is low, the results are likely accurate so we slowly explore lower keyframe rates that save network bandwidth. We use this simple multiplicative-increase, multiplicative-decrease strategy to evaluate the effectiveness of adaptation to maintain accuracy, but more research is needed to develop more robust strategies that can, for example, deal with a sudden change in the environment, e.g. when moving from an urban road with many distinct buildings into a park with a uniformly green leafy environment.

VI. Cloud-Vehicle Coordination

Figure 6 shows how the various mechanisms of CloudSLAM work together during a trip.

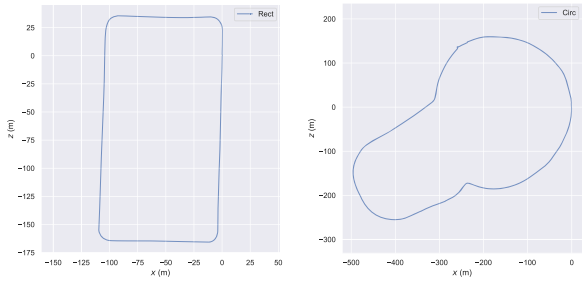
At the start of a trip, the vehicle sends its first frame to the edge. The Remote Mapping module will try to match it with a keyframe in the global map. When a match is found, the module provides the vehicle with its pose based on the frame's placement in the global map, which was initialized using an archival cloud map. This aligns the coordinate systems of the local and global maps, which simplifies the exchange of keyframes and updates during the trip. The vehicle then proceeds with Tracking and Local Mapping as usual. The vehicle continues to send keyframes to the edge as the strategy in use specifies.

During the trip, the vehicle sends keyframes to the edge at a rate determined by the adaptation strategy, discussed in Section V-B, to achieve output-driven consistency. For each keyframe, Remote Mapping does loop closing on a limited set of frames similar to the new keyframe. This updates the cloud map with new information that the keyframe might contain and provides an optimized pose estimate for the keyframe. This optimized pose is sent to the vehicle, which uses it to improve both the future poses computed by the Tracking module and also the local map updated by Local Mapping.

After the end of the trip, we assume that the vehicle can use an inexpensive high-bandwidth wireless link (e.g. WiFi) to upload all keyframes of the trip to the cloud. CloudSLAM processes these frames using full loop closure to help grow and keep the archival cloud map up-to-date so that it remains useful for future trips, even as the environment changes. For this procedure, CloudSLAM applies the more time consuming global BA to fully benefit from the information in the keyframes.

VII. Evaluation

We present a trace-driven evaluation of the end-to-end performance of CloudSLAM using a prototype implementation. We also explore the accuracy-bandwidth trade-off of output-driven consistency and compare the keyframe transmission strategies that we developed.



(a) Rectangular Trace (b) Circular Trace

Fig. 7. The Two Traces

A. Experiment Setup

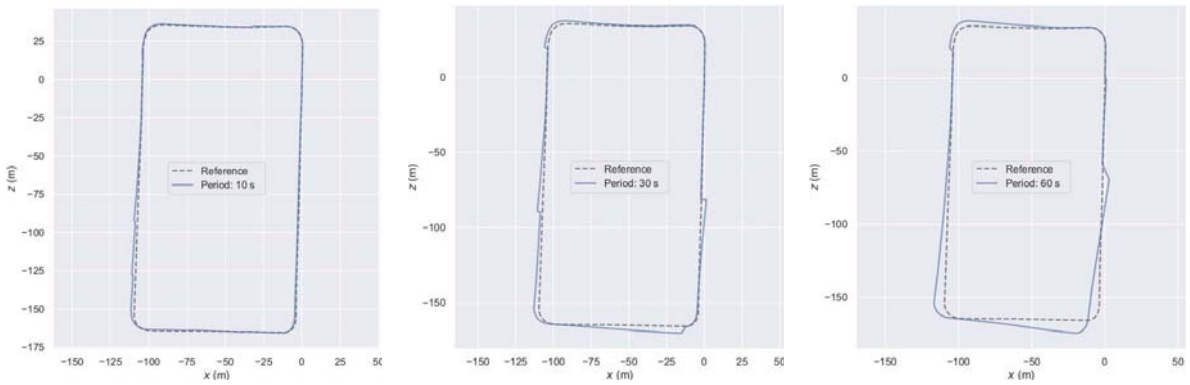
1) Hardware Setup: Our experimental setup for CloudSLAM evaluation consists of two networked computers representing a vehicle and an edge server. We use a Dell PowerEdge R720 with a 2.8 GHz Intel Xeon E5-2680 v2 to represent the vehicle and a Dell Precision Tower 3620 with a 4 GHz Intel Core i7-6700K to represent the edge server. From our experimentation with the unmodified ORB-SLAM, we observed that the software performance is CPU-bound. We believe assigning a faster CPU to the edge server is representative of a realistic environment, where we expect that edge servers will have relatively more resources than vehicles.

2) Approach: As described earlier, our focus is on using ORB-SLAM as an example application to develop an architecture for offloading stateful, latency-sensitive applications using output-driven consistency, so we are *not* trying to develop a better visual SLAM system. As a result, we use the live trajectory output of ORB-SLAM as the baseline for our experiments and evaluate how well CloudSLAM replicates the ORB-SLAM output.

3) Traces: Up until now we've been using the KITTI vision benchmark suite [14] to inform our development decisions. KITTI is widely used for evaluating SLAM but unfortunately, none of the KITTI traces include substantial loops so they are not suitable for evaluating CloudSLAM.

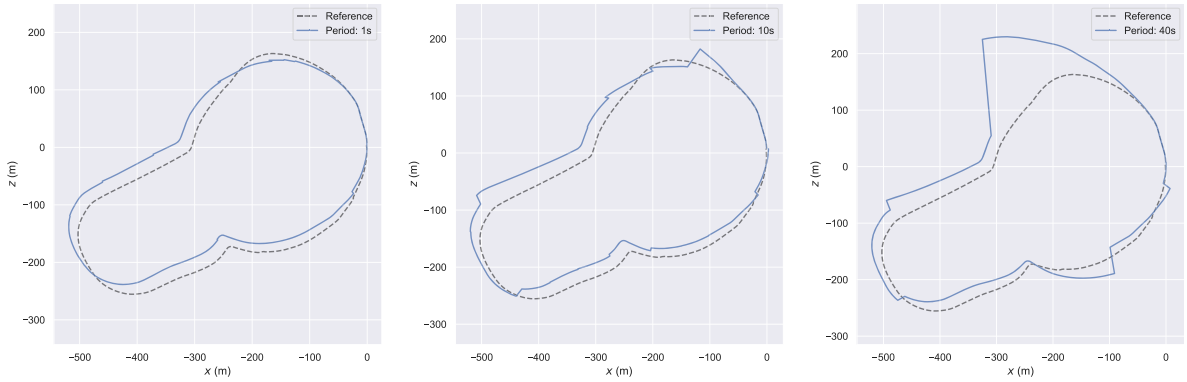
For that reason, we collected two traces consisting of long loops. As shown in Figure 7, the first trace is rectangular in shape and was captured on an urban campus, while the second trace is more circular and was captured in a suburban community. Images for the traces were recorded by a camera with 1280x720 resolution, operating at 60 fps (frames per second). We downsampled the traces to 15 fps so they can be processed in real-time by ORB-SLAM and CloudSLAM. The stereo camera has a 12 cm baseline.

The rectangular and circular traces are important for two reasons. Firstly, the traces consist of two laps along the



(a) Period: 10s, RMSE: 0.99m (b) Period: 30s, RMSE: 2.12m (c) Period: 60s, RMSE: 6.22m

Fig. 8. Example trajectories using the periodic strategy for the rectangular trace



(a) Period: 1s, RMSE: 17.88m (b) Period: 10s, RMSE: 19.13m (c) Period: 40s, RMSE: 48.42m

Fig. 9. Example trajectories using the periodic update strategy for the circular trace

Traces	# Frames	Path Length (m)	Duration (s)	Top Speed (mph)
Rectangle	1922	598	128	15
Circular	2984	1450	200	24

TABLE II
DETAILS ABOUT SECOND LOOPS OF CUSTOM DATASET (BOTH ARE 15 FPS AT 1280X720 RESOLUTION)

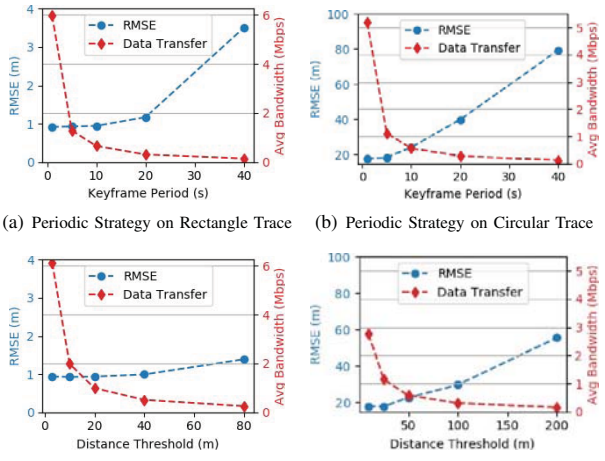
same path. We use the first lap as "previous trip" data for both ORB-SLAM and CloudSLAM. The previous trip data is optimized using loop closure and global bundle adjustment. The second lap is used for evaluation purposes. Details on the second lap are shown in Table II. Secondly, the traces are challenging enough that ORB-SLAM experiences significant drift when no previous trip data is available, as can be seen in Figure 2. In other words, the Tracking and Local Mapping modules alone are not sufficient to get good performance. This allows us to evaluate the impact that offloading has on CloudSLAM's performance relative to ORB-SLAM. The circular trace is especially difficult because long continuous rotation is fundamentally challenging for SLAM [15].

4) Comparing Trajectories: To measure the error of our trajectory output against the ORB-SLAM baseline, we use the root mean square error (RMSE) of the pose output as our metric, which is commonly used for comparing trajectories. A trajectory consists of a finite set of points representing the poses determined for each frame of the video trace. First, we use Euclidean distance to determine the pose error between the corresponding points of the ORB-SLAM and CloudSLAM trajectories. Then, we combine the error across all of the points using RMSE. Figure 8 and Figure 9 show examples of CloudSLAM output overlaid on the ORB-SLAM baseline (dashed line). It can be observed that the RMSE increases as the CloudSLAM trajectory deviates more from the ORB-SLAM one.

Fig. 10. Map of the circular trace*

*Base map and data from OpenStreetMap and OpenStreetMap Foundation

It is challenging to provide a proper reference for the circular trace due to the difficulty of the trace. The reason is that the ORB-SLAM trajectory generated by global BA after the first lap has a high error relative to ground truth. We do not have ground truth in the form of GPS coordinates for the trace, but Figure 10 shows a map of the roads used for the circular trace (Verona Cir. and Fulton Pl.). We can see that the shape of the reference trajectory for the circular trace (dashed line in Figure 9) is quite different from the actual road in the map. The reason is that global BA distributed the error uniformly across the loop. In fact, with the period parameter of 1 second, the output of CloudSLAM after the second lap (blue trajectory in Figure 9) matches the map much better, so CloudSLAM (and ORB-SLAM) improve the accuracy of the trajectory relative to the actual road shape during the second lap. Due to the poor quality of the reference frame, the RMSE results for the circular trace suffer from a large offset. A cloud map that has benefited from multiple previous trip data sets would not suffer from this problem regarding how ORB-SLAM handles large amounts of rotation [15].



(a) Periodic Strategy on Rectangle Trace (b) Periodic Strategy on Circular Trace

(c) Distance Strategy on Rectangle Trace (d) Distance Strategy on Circular Trace

Fig. 11. RMSE and data transfer plots for each strategy and trace, averaged over five runs.

B. RMSE versus Bandwidth Tradeoff

Figures 11(a) and 11(b) show the results of using CloudSLAM with the fixed-rate Periodic Strategy on the Rectangle and Circular traces, respectively. The figures show both the RMSE (blue circle) and the Average Bandwidth (red diamond) as a function of the keyframe period. The Average Bandwidth is calculated based on all data sent in both directions between the vehicle and the edge during the trip. This value is directly correlated with the number of keyframes transmitted. Each keyframe is about 500 KB on average while pose updates from the edge are about 90 bytes. As a baseline, vehicle-only operation for the Rectangle trace results in an RMSE of 25.3 m, whereas sending all keyframes to the edge results in an RMSE of 0.899 m at 35.3 Mbps.

The results show a trade-off between accuracy and bandwidth use of CloudSLAM. In Figure 11(a), when the keyframe period is small, the error is low but the network utilization is high. On the other hand, when the period is higher, around 20 seconds, the network utilization drops below 5 MB as only around 9 keyframes are transmitted, corresponding to about 0.310 Mbps on average for the duration of the trip. This trend is more pronounced in the Circular trace, Figure 11(b), which is a result of the Circular trace being much more challenging than the Rectangle trace.

Figures 11(c) and 11(d) show the same results for the fixed-rate Distance Strategy for the two traces. We observe that similar to the periodic scheme, the distance scheme enables a trade-off between accuracy and bandwidth, but there is a significant difference. For low data transfer rates, the RMSE is noticeably lower for the Distance Strategy than for the Periodic Strategy, suggesting that the Distance Strategy is more effective at deciding when to send keyframes to the edge server.

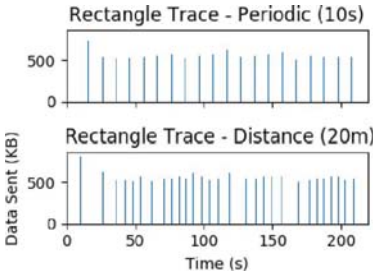


Fig. 12. Timeline of keyframe transmissions for the Rectangle trace

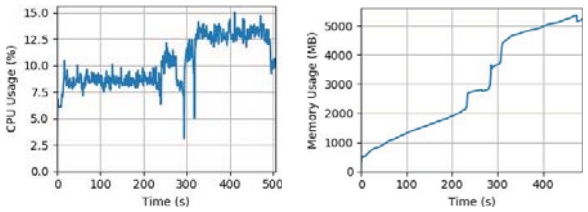
C. Comparing Periodic and Distance Strategies

During our initial experimentation with the KITTI dataset, we observed that drift is more a function of distance than it is of time. We believe that this observation explains the performance difference between the Periodic and Distance Strategies. The Periodic Strategy is based on sending keyframes at fixed time intervals so it does not consider the motion of the vehicle. In contrast, the Distance Strategy sends keyframes based on how far it believes the vehicle has moved, so it implicitly adapts to the degree of change in the frame content. Figure 12 shows a graphical representation of this for the Rectangle trace.

Fundamentally, what this means is that the two strategies really only differ when the vehicle's speed changes during the trip, in which case the distance between keyframes sent to the edge by the Periodic Strategy varies as well. If the distance traveled varies between keyframes that are sent to the edge server, so will the potential amount of drift incurred that can be corrected. Therefore, the Distance Strategy results in more efficient control of which keyframes are sent to the edge by sending keyframes based on distance rather than time.

Between the Periodic and Distance Strategies, we conclude that the Distance Strategy is typically the best to use since it adapts to a vehicle's speed. On the other hand, the Periodic

Strategy may be preferred if predictable utilization of the wireless link is desired.



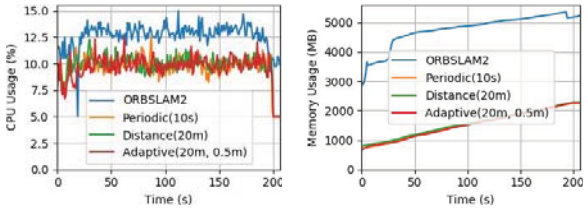
(a) CPU Usage (b) Memory Usage

Fig. 13. ORB-SLAM resource usage for Rectangle trace

D. Resource Usage

Figure 13 shows ORB-SLAM's CPU and memory usage while processing both laps of the Rectangle trace. For the first half of the trace, the CPU usage remains fairly steady as only the Tracking and Local Mapping modules of ORB-SLAM are fully active. At around 250 seconds and again at around 300 seconds, the CPU usage spikes because the Loop Closing module triggers a global bundle adjustment. The Loop Closing module is activated because ORB-SLAM has determined that it has returned to a previous location. In terms of memory, it can be seen in Figure 13(b) that ORB-SLAM's usage grows continuously as new keyframes are added to the map. This is clearly not a sustainable level of growth.

Figure 14 compares the CPU and memory usage of CloudSLAM and ORB-SLAM on the vehicle *during the second loop*; CloudSLAM is shown processing the Rectangular trace using different strategies. We see that the difference in CPU use between CloudSLAM and ORB-SLAM is fairly small. CloudSLAM reduces CPU use on the vehicle by about 20%, compared to the original, by offloading the Remote Mapping module to the edge, so the vehicle does not have to perform loop detection or global bundle adjustments. However, with regards to map data, CloudSLAM reduces memory usage by 68% on average as shown in Figure 14(b).



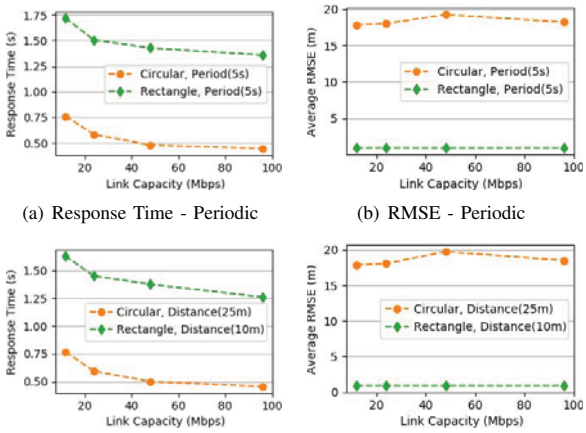
(a) CPU Usage (b) Memory Usage

Fig. 14. CloudSLAM vehicle resource usage for Rectangle trace

E. Impact of Network Bandwidth and Latency

The wireless link connecting the vehicle to the edge is an essential part of the operation of CloudSLAM. We investigated the impact of the available bandwidth and link latency between the vehicle and the edge on CloudSLAM's performance. In our experiments, the vehicle and edge node are connected by an Ethernet link that can sustain 940 Mbps as measured by the `iperf3` network measurement tool and the link has an average RTT of 0.8 ms. On top of this we use the Mahimahi tool [18] to emulate different link speeds, ranging from 12-96 Mbps. The lower rates represent average uplink LTE bandwidth today, while the faster rates represent possible future 5G bandwidth. We also utilize Mahimahi to emulate different link latencies, ranging from 10 to 400 ms round-trip.

1) Variable Wireless Bandwidth: As a starting point, we examine the performance of CloudSLAM using a simulated mobile 4G LTE link based on real-world trace data of a major carrier. This is representative of the variable wireless bandwidth one can expect using 4G. The traces we obtained are compatible with Mahimahi and were collected by Winstein et al. while driving around the Boston area in 2013 [19]. The throughput achieved in the trace for the uplink varies from 0-17.7 Mbps, with an average of 6.2 Mbps. For the Rectangle trace, the Periodic Strategy achieves an RMSE of 0.944 m at an average bandwidth of 0.732 Mbps and the Distance Strategy achieves an RMSE of 0.923 m at 0.979 Mbps. This indicates that our CloudSLAM design is robust to dynamic bandwidth availability and is consistent with the results of the more systematic bandwidth evaluation we present below.



(a) Response Time - Periodic (b) RMSE - Periodic

(c) Response Time - Distance (d) RMSE - Distance

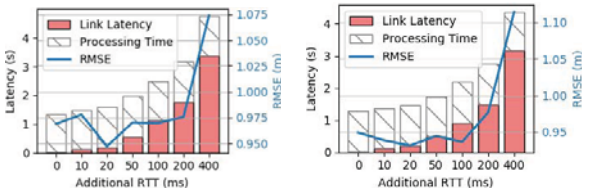
Fig. 15. Response time and RMSE versus available bandwidth.

2) Controlled Bandwidth Experiments: For a more systematic assessment, Figures 15(a) and 15(b) show CloudSLAM's response time and RMSE, respectively, for the Periodic Strategy under different available link bandwidths. These results are averaged over five runs. Here, response time refers to the amount of time it takes the vehicle to receive a response from the edge starting from when a keyframe begins transmission. As we expect, Figure 15(a) shows that the delay decreases as the bandwidth increases. We initially thought that the delay for both traces would be similar, however it turns out that the Rectangle trace experiences a higher number of map point matches on the edge, almost double that of the Circular trace. This causes the bundle adjustment performed during edge processing to take significantly longer for the Rectangle trace

than for the Circular trace, leading to an increase in the response time. For the RMSE shown in Figure 15(b), the results are interesting. It can be seen that even with the change in delay incurred from the bandwidth constraint, there is no noticeable impact on the RMSE by varying the bandwidth. The RMSE remains steady across the range of bandwidth values we tested. We conclude that the bandwidth does not play a significant role in the performance for the Periodic Strategy, as the computation performed in the edge dominates the response time.

Similarly, Figures 15(c) and 15(d) show CloudSLAM's response time and RMSE, respectively, for the Distance Strategy. Figure 15(c) shows that the delay is inversely correlated with the bandwidth as expected. In Figure 15(d), for both the Rectangle and Circular traces, the RMSE remains steady as we saw previously. As with the Periodic Strategy, there is no correlation between the RMSE and bandwidth for the range of values we tested, since the edge server computation is where most of the time is spent.

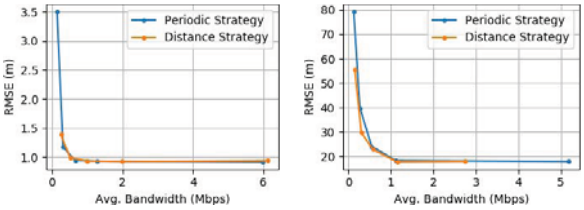
Based on the bandwidth results, we conclude that with wireless bandwidths above 10 Mbps, the bandwidth is not as important as the proper selection of which keyframes to transmit. Higher bandwidth does reduce the response time, but even at 10 Mbps, the time needed to transmit a keyframe is low compared to the time needed by the edge to process them. This insight about the impact of bandwidth on system performance provided guidance to our system design strategy of CloudSLAM and how this framework could apply to other applications of this class.



(a) RTT impact on Periodic Strategy (10s) (b) RTT impact on Distance Strategy (20m)

Fig. 16. Impact of RTT on CloudSLAM performance

3) Controlled Latency Experiments: We next examine how the performance of CloudSLAM is impacted by the roundtrip time (RTT) between the vehicle and the edge server. This allows us to evaluate the impact of using a centralized cloud as opposed to an edge cloud. The results are shown in Figure 16. We see that as the RTT increases, the performance of CloudSLAM degrades: the RMSE shown as the blue curve (right y-axis) increases rapidly. The reason is that the data transfer latency becomes the dominant portion of the response time (left y-axis). As a result, pose corrections for the vehicle are delayed, so they become less useful to the vehicle for correcting drift as the keyframe that was sent is now too far in the past. Note that the artificial delay added for this experiment gets amplified by the mechanisms involved in TCP's congestion control algorithm.

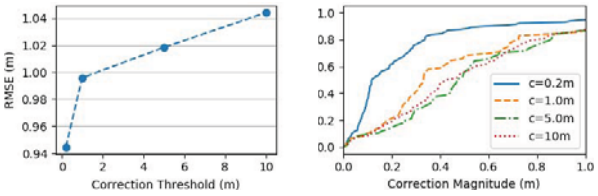


(a) Rectangular trace trade-off (b) Circular trace trade-off

Fig. 17. RMSE versus bandwidth for the two traces and strategies

F. Adaptive Strategy

Figure 17 depicts the trade-off in error and bandwidth that the Periodic and Distance strategies provide for our traces. These curves are achieved by varying the respective parameters of the strategies, e.g. time or distance. Our results indicate that there is not necessarily a single parameter that works well for all cases. Rather, it depends on the needs and constraints of the application (i.e., what RMSE is acceptable), as well as the environment in which the vehicle is operating (e.g., degree of difficulty, rectangular versus circular). To assist with dynamically finding the best parameter value, we now evaluate an adaptive strategy that is based on the algorithm described in Section V-B. The algorithm adapts its distance threshold based on the size of the pose adjustments, or corrections, returned by the edge.



(a) Impact of correction threshold on RMSE (b) CDF of pose correction sizes

Fig. 18. Performance of adaptive strategy on Rectangular trace for initial distance threshold of 40m

Figure 18(a) shows the RMSE versus the correction threshold for the Rectangular trace. The initial distance for controlling the keyframe rate was set at 40m. We observe that the RMSE decreases when the correction threshold that is used to trigger changes in the distance threshold decreases, showing that the algorithm allows us to indirectly control the overall error. One would expect that lower correction thresholds would also reduce the magnitude of the keyframe adjustment since it results in sending more keyframes. Figure 18(b) confirms this: it shows the CDF of the resulting correction magnitudes. It can be seen that, as the correction threshold decreases, the likelihood of large corrections decreases as well. Altogether, this demonstrates that it is possible for the strategy to adapt a parameter based on a user-specified tolerance.

We provide this Adaptive Strategy as a proof-of-concept indicating that it is possible to use rate-based strategies without having to manually tune their respective parameters for differ-

ent environments, which would be impractical. We recognize that more sophisticated adaptation mechanisms are warranted given the wide variety of driving scenarios one can encounter.

VIII. Related Work

A. Mobile and Vehicular Cloud Offloading

Cloud offloading in general is a well-studied topic and is widely used. For example, it is widely applied to speed up computation or reduce power consumption for various mobile devices and more recently for vehicles. However, most of the previous work involves offloading tasks with relatively modest input data, so it is possible to offload the entire application or to use RPC to offload application components [2], [1], [20], [21], [22], [23]. Some more recent work has looked at offloading applications that process video or other high bandwidth inputs that cannot be sent to the cloud. Offloading is still possible by partitioning the application, i.e., the mobile device processes the high bandwidth sensor input, reducing the volume of the data, while other tasks can be offloaded [3], [4], [5]. Our CloudSLAM design builds on this idea, specifically by executing the Tracking and Local Mapping tasks on the vehicle and sending only a subset of keyframes to the cloud. However, none of the earlier works on vehicular offloading that involve large internal state consider the compromise on state replication that we explore with Output-driven Consistency.

B. Consistency Models in Distributed Computing

Consistency has been a challenging issue in the history of distributed computing [16], [24], [25]. In the early phase of distributed computing systems, strong consistency criteria were used to make a distributed system appear to be a centralized system, providing the illusion of sequential access [26], [27]. Providing strong consistency imposes performance overhead and limits system availability, especially in Internet-based computing systems. Alonso et al. [28] studied caching strategies in information retrieval systems. The caching strategies enhance system performance by allowing certain deviations during system or communication failures. Yu et al. [28] explored the semantic space between strong consistency and optimistic consistency for replicated services and introduced the concept of inconsistency error bound. Our work has benefited from this existing knowledge in consistency models, but our work differs because our focus is on optimizing the *accuracy* of the output, allowing us to focus the consistency requirements of the shared state between the edge and vehicle using the output-driven consistency model.

C. SLAM

Camera-based Visual SLAM systems are receiving increasing interest in robotics and autonomous driving. Previous works related to offloading SLAM between a mobile device and the edge include Edge-SLAM [9] and edgeSLAM [10]. In Edge-SLAM [9], the edge server periodically copies a portion of its global map to the mobile device, while in the other edgeSLAM [10], the edge server sends differential pose and map point updates to the mobile device for each keyframe it processes. Unfortunately, these solutions are not ideal for the use case CloudSLAM considers. Both of the above systems were designed for indoor use, so the speed of mobility is low and they have access to well-provisioned WiFi networks, so bandwidth usage is not a primary concern. In contrast, CloudSLAM is designed for fast-moving vehicles that use a cellular network. This makes the offloading problem much more challenging. Indoor walking speeds are typically 1 m/sec [10], which is about 14 times slower than a car driving at 50 km/h, so the part of the map that is of interest to a user changes much faster. This will cause an increase in the network traffic needed to keep the data structures in a distributed setup consistent. In addition, while typical 4G download speeds are 5-12 Mbps, upload speeds are more critical for CloudSLAM and are much lower (2-5 Mbps), especially compared to WiFi.

As robots and autonomous vehicles are connected using wireless communication technology, a new trend is growing for simultaneous localization and mapping in a collaborative manner, leveraging the powerful computing resources of the cloud. Forster et al. [29] did a pilot study of joint mapping using micro aerial vehicles. Schmuck et al. [30] proposed a collaborative SLAM system for multiple UAVs based on ORB-SLAM2. In their system, each UAV runs a lightweight visual odometry system and sends keyframes and map points to a central server, which performs computationally intensive tasks such as map optimization and merging. Van Opdenbosch et al. [31] proposed an efficient coding framework to compress the visual features in a collaborative SLAM system. These pioneer works have laid the technical basis for a cooperative SLAM that uses cloud computing. Our objective is not to build a new SLAM system, but we use SLAM as an example of a stateful, latency-sensitive application that can benefit from using the edge in a dynamic fashion. How our results apply to cooperative SLAM is interesting future research.

IX. Conclusion

Many vehicular applications, especially those related to autonomous driving, are becoming increasingly complex and resource hungry. They currently run entirely in the vehicle and are limited by the equipped resources. In this work we present a new consistency model, Output-driven Consistency, that enables us to efficiently support partitioning of latency-sensitive applications that consume high-bandwidth data streams and maintain large complex state. Output-driven Consistency enables us to tradeoff accuracy and bandwidth, so that we can maintain application performance while minimizing network traffic. Output-driven Consistency accomplishes this by focusing on the consistency of application output rather than the consistency of the application's internal state. Using a Simultaneous Localization and Mapping (SLAM) application as an example, we develop CloudSLAM, a partitioned version of ORB-SLAM that incorporates Output-driven Consistency. Through an extensive evaluation, we demonstrate that CloudSLAM is effective at maintaining accuracy while minimizing network traffic.

REFERENCES

[1] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: Making smartphones last longer with code offload," in *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, (New York, NY, USA), p. 49–62, Association for Computing Machinery, 2010.

[2] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: Elastic execution between mobile device and cloud," in *Proceedings of the Sixth Conference on Computer Systems*, EuroSys '11, (New York, NY, USA), p. 301–314, Association for Computing Machinery, 2011.

[3] M.-R. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, (New York, NY, USA), p. 43–56, Association for Computing Machinery, 2011.

[4] A. Ashok, P. Steenkiste, and F. Bai, "Enabling vehicular applications using cloud services through adaptive computation offloading," in *Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services*, MCS '15, (New York, NY, USA), p. 1–7, Association for Computing Machinery, 2015.

[5] A. Ashok, P. Steenkiste, and F. Bai, "Adaptive cloud offloading for vehicular applications," in *Proceedings of IEEE Vehicular Networking Conference (VNC)*, (Piscataway, NJ), pp. 1–8, IEEE, December 2016.

[6] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The Case for VM-Based Cloudlets in Mobile Computing," *IEEE Pervasive Computing*, vol. 8, October-December 2009.

[7] M. Satyanarayanan, "The Emergence of Edge Computing," *IEEE Computer*, vol. 50, January 2017.

[8] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, SEC '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[9] A. J. B. Ali, Z. S. Hashemifar, and K. Dantu, "Edge-slam: Edge-assisted visual simultaneous localization and mapping," in *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, MobiSys '20, (New York, NY, USA), p. 325–337, Association for Computing Machinery, 2020.

[10] J. Xu, H. Cao, D. Li, K. Huang, C. Qian, L. Shangguan, and Z. Yang, "Edge assisted mobile semantic visual slam," in *IEEE Conference on Computer Communications, (INFOCOM '20)*, Infocom '20, IEEE, April 2020.

[11] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics - Special Issue on the 2007 DARPA Urban Challenge*, vol. 25, pp. 425–466, Aug. 2008.

[12] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE transactions on robotics*, vol. 31, no. 5, pp. 1147–1163, 2015.

[13] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[14] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, (Piscataway, NJ), pp. 3354–3361, IEEE, 2012.

[15] C. Pirchheim, D. Schmalstieg, and G. Reitmayr, "Handling pure camera rotation in keyframe-based SLAM," in *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, (Piscataway, NJ), pp. 229–238, IEEE, 2013.

[16] M. K. Aguilera and D. B. Terry, "The many faces of consistency.," *IEEE Data Eng. Bull.*, vol. 39, no. 1, pp. 3–13, 2016.

[17] Google, "Google AR & VR." https://arvr.google.com/ar/, June 2020.

[18] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for HTTP," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, (Santa Clara, CA), pp. 417–429, USENIX Association, July 2015.

[19] K. Winstein, A. Sivaraman, and H. Balakrishnan, "Stochastic forecasts achieve high throughput and low delay over cellular networks," in *Presented as part of the 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 459–471, 2013.

[20] X. Gu, K. Nahrstedt, A. Messer, I. Greenberg, and D. Milojicic, "Adaptive offloading for pervasive computing," *IEEE Pervasive Computing*, vol. 3, pp. 66–73, March 2004.

[21] D. S. an Adam Barker, "MAMoC: Multisite Adaptive Offloading Framework for Mobile Cloud Applications," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, (Hong Kong), pp. 17–24, IEEE, 2017.

[22] K. Kumar, J. Liu, Y.-H. Lu, and B. Bhargava, "A survey of computation offloading for mobile systems," *Mobile Networks and Applications*, vol. 18, p. 129–140, Feb. 2013.

[23] D. Sulaiman and A. Barker, "Task offloading engine for heterogeneous mobile clouds," in *MobiCASE'16: Proceedings of the 8th EAI International Conference on Mobile Computing, Applications and Services*, (New York, NY, USA), pp. 1–2, Association for Computing Machinery, 2016.

[24] P. Viotti and M. Vukolic, "Consistency in Non-Transactional Distributed Storage Systems," *ACM Computing Surveys*, vol. 49, pp. 359–384, Jul. 2016.

[25] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database system concepts*. Boston, MA: McGraw-Hill Higher Education, 5th ed., 2006.

[26] L. Lamport, "The implementation of reliable distributed multiprocess systems," *Computer Networks (1976)*, vol. 2, no. 2, pp. 95–114, 1978.

[27] Lamport, "How to make a multiprocessor computer that correctly executes multiprocess programs," *IEEE Transactions on Computers*, vol. C-28, pp. 690–691, Sep. 1979.

[28] R. Alonso, D. Barbara, and H. Garcia-Molina, "Data caching issues in an information retrieval system," *ACM Transactions on Database Systems*, vol. 15, pp. 359–384, Sep. 1990.

[29] C. Forster, S. Lynen, L. Kneip, and D. Scaramuzza, "Collaborative monocular slam with multiple micro aerial vehicles," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Piscataway, NJ), pp. 3962–3970, IEEE, 2013.

[30] P. Schmuck and M. Chli, "Multi-uav collaborative monocular slam," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, (Piscataway, NJ), pp. 3863–3870, IEEE, 2017.

[31] D. Van Opdenbosch and E. Steinbach, "Collaborative visual slam using compressed feature exchange," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 57–64, 2018.