2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 978-1-6654-4066-0/20/$31.00 ©2021 IEEE | DOI: 10.1109/IPDPS49936.2021.00083

Towards Practical Cloud Offloading for Low-cost Ground Vehicle Workloads

Yuan Xu[1,2,3], Tianwei Zhang[4], Jimin Han[1,2,3], Sa Wang[1,2,3], Yungang Bao[1,2,3]
[1]State Key Laboratory of Computer Architecture, Institute of Computing Technology
[2]University of Chinese Academy of Sciences
[3]Peng Cheng Laboratory
[4]Nanyang Technological University

***Abstract*—Low-cost Ground Vehicles (LGVs) have been widely adopted to conduct various tasks in our daily life. However, the limited on-board battery capacity and computation resources prevent LGVs from taking more complex and intelligent workloads. A promising approach is to offload the computation from local LGVs to remote servers. However, current cloud-robotic research and platforms are still at a very early stage. Compared to other systems and devices, optimizing LGV workload offloading faces more challenges, such as the uncertainty of environments and the mobility feature of devices.**

In this paper, we explore the opportunities of optimizing cloud offloading of LGV workloads from the perspectives of performance, energy efficiency and network robustness. We first build an analytical model to reveal the computation role and impact of each function in LGV workloads. Then we propose several optimization strategies (fine-grained migration, cloud acceleration, real-time monitoring and adjustment) to accelerate workload computation, reduce on-board energy consumption, and increase the network robustness. We implement an end-to-end cloud-robotic framework with such strategies to achieve dynamic and adaptive offloading. Evaluations on physical LGVs show that our strategies can significantly reduce the total energy consumption by $2.12\times$ and mission completion time by $2.53\times$, and maintain strong robust ness under poor network quality.

***Index Terms*—performance analysis, adaptive offloading, cloud acceleration, network robustness**

I. Introduction

Low-cost Ground Vehicles (LGVs) have gained ever-increasing attention of the public. They exhibit a strong potential to assist or even replace human beings in particular scenarios to complete certain tedious or dangerous missions, e.g., delivering packages [1], housework [2], searching and rescuing [3]. Moreover, benefiting from the advance of artificial intelligence techniques, LGVs are expected to have deeper influence on every aspect of our daily life in the near future.

However, the limited battery capacity and computation capability on LGVs are proved to be key bottlenecks in the development of LGVs, hindering them from being more intelligent and multi-functional. Besides, these two factors mutually restrict each other, making it hard for developers to consider both high performance and low energy consumption. For instance, a Turtlebot3 [4] is equipped with a $19.98Wh$ lithium polymer battery. Most of the energy is consumed by motors, sensors and the microcontroller, leaving the embedded computer only $3.35Wh$ for 1 hour (Table I) to run workloads. Thus the vehicle has to integrate low-power embedded computers that execute the complex intelligent algorithms at a relatively slower speed. This reduces the velocity to ensure the execution of the tasks and significantly delays the entire mission completion time.

To overcome these challenges, cloud offloading is proposed, which leverages general-purpose cloud servers to run part or full workloads for robotic devices. Development of robotic workloads can also benefit from existing cloud services, e.g., big data analytics, collective learning, human computation [5]–[7]. Specifically, (1) cloud-robotic platforms (e.g. AWS RoboMaker [8], Rapyuta [9], Davinci [10]) were introduced to help programmers deploy robotic computations across cloud and robotic devices. (2) cloud-robotic services were designed based on those platforms to accelerate specific algorithms [11]–[17] or enable data sharing among a group of robots [18]–[20] in the cloud.

However, those cloud-robotic platforms and services are still not fully developed or optimized. They mainly attempt to improve the cloud-robotic functionalities [11]–[20], or ease the development of robotic workloads [8], [9]. Optimization of performance, energy consumption and network robustness is not systematically considered. This can underutilize the advantages of cloud servers, and without optimization, cloud offloading can sometimes have worse effects than local computing under certain conditions (e.g., poor network quality).

In this paper, we present a new and systematic study about the opportunities and challenges of optimizing the cloud offloading of LGV workloads, from the perspectives of performance, energy efficiency and network robustness. Optimization of mobile cloud systems have been widely studied [21]–[27]. However, it is unsuitable to apply those solutions to cloud-robotic systems, due to the significant differences between mobile and LGV workloads and devices. First, there are more factors to control in an LGV in order to efficiently complete the workload (CPU frequency, velocity, sensory frequency, etc.). This increases the difficulty of identifying offloading strategies for LGV workloads. Second, the majority of power consumption in a mobile device is GSM module and display [28], while the power is mainly consumed by the motor in an LGV. This also indicates the inapplicability of mobile cloud offloading methods to LGV workloads. Third, unlike

978-1-6654-4066-0/21/$31.00 ©2021 IEEE
DOI 10.1109/IPDPS49936.2021.00083

portable mobile devices, which passively interact with the environment through user's behaviors, the LGV autonomously controls the velocity and path based on the modeled surrounding environment. This indicates the computational process in an LGV workload must not be interrupted under poor network quality; otherwise the LGV will be suspended and the mission will fail. As a result, an optimization strategy specifically for LGV offloading is necessary.

The key to our solution is the introduction of an analytical model, which can reveal the main factors that determine the energy consumption and mission completion time of LGV workloads. Based on this model, we identify three opportunities of offloading LGV workloads. (1) *Fine-grained migration*: different from existing platforms [8], [9] which offload the entire workload, our solution selects the optimal nodes[1] for migration. This policy can achieve more efficient computation. (2) *Cloud acceleration:* we propose some approaches to optimize workload execution on the cloud servers. We utilize architecture characteristics (e.g., parallelization) of the server hardware to further accelerate the execution, and reduce the energy consumption and completion time of the entire task. (3) *Real-time adjustment:* current platforms only support static offloading policy. Instead, we design a novel strategy that uses the wireless signal direction (mobility feature) and network bandwidth (environment feature) to dynamically predict the network quality and adjust the offloading policy. This can achieve strong robustness under poor network condition.

We build an end-to-end cloud-robotic platform with above three optimization strategies and experiment with two standard LGV workloads: *Navigation* and *Exploration* on a Turtlebot3, an edge gateway in our lab and cloud servers in a remote datacenter. Evaluation results indicate that our offloading strategies can significantly reduce the total energy consumption by $1.61\times$ and $2.12\times$, and mission completion time of the two workloads by $2.53\times$ and $1.6\times$, respectively. The reduction of mission completion time mainly benefits from the parallelization optimization on the remote servers, as the processing time of velocity dependent path is remarkably reduced by $23.92\times$.

In summary, our contributions are as follows:

- An analytical model to reveal the computation role of energy and performance optimization in LGV workloads, and the chances and challenges of cloud offloading (Section III).
- A fine-grained migration strategy to select the optimal nodes for offloading, to achieve low energy consumption or short completion time, based on developers' demands (Section IV).
- A cloud acceleration method to identify bottleneck functions and accelerate them on cloud servers (Section V).
- A real-time monitoring and adjustment mechanism to provide robustness based on signal direction and bandwidth under poor network condition (Section VI).

[1]A node denotes one robotic functional process. A robotic workload consists of many concurrently-running nodes.

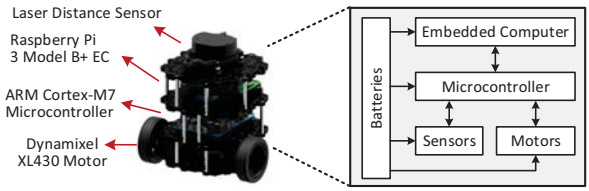

Fig. 1. Architecture of a common LGV – Turtlebot3.

- An end-to-end system to achieve dynamic and adaptive offloading and optimization (Section VII).

II. Background

We present the background of Low-cost Ground Vehicles (LGVs), including the characteristics of architecture (§ II-A), and standard workloads (II-B).

A. Architecture

An LGV (e.g. iRobot [2], turtlebot2 [29], Turtlebot3 [4]) is a mobile robot operating in a group without human interactions or controls. It autonomously observes the environment, makes decisions, and moves to certain destinations. An LGV is usually light (<5kg) and operates within a small area due to the limited battery capacity. This is different from autonomous vehicles or humanoid robots. Figure 1 shows five major components in a common LGV:

Batteries: Batteries provide the energy to support the operations of other four components. The capacity of batteries is a key factor, determining the types of workloads and mission duration an LGV can execute.

Sensors: Sensors are used to perceive the state associated with the LGV and its surrounding environment. According to the localization method, the LGV can fall into one of the two categories: vision-based LGV with RGBD cameras (e.g. turtlebot2) and laser-based LGV with laser distance sensor (e.g. Turtlebot3).

Motors: Motors convert electrical energy into mechanical energy to drive an LGV to move forward. When the LGV moves with an acceleration of a, a traction force of $m(a+g\mu)$ is needed for providing dynamics, where m, μ and g denote the LGV's mass, ground friction constant and gravity constant.

Microcontroller: The microcontroller is a bridge, connecting the embedded computer with sensors and motors. It is responsible for forwarding commands (e.g. polling data and moving forward) to sensors and motors from the computer.

Embedded Computer: The embedded computer is the essential component of an LGV for automation and communication. It is responsible for executing the workload, described below.

B. Standard Workload Characteristics

An LGV can be used in many scenarios, e.g., package delivery, house-cleaning, target-searching. All these workloads can be abstracted as: *navigating the vehicle to a given destination based on the environmental information captured from various sensors*. So those workloads always follow a standard pipeline, as shown in Figure 2.

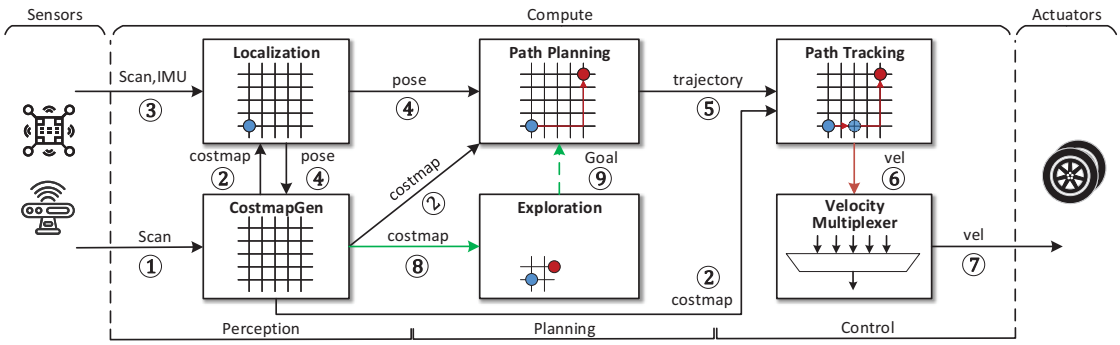

Fig. 2. The standard pipeline for common LGV workloads. The workflow with a map is depicted in black lines. The workflow without a map contains all operations in black lines and some new operations depicted in green lines.

This computation pipeline is composed of three major processing stages: PERCEPTION, PLANNING and CONTROL [30]. Each node in Figure 2 represents a type of functional computation. The solid arrows denote that the connected two nodes communicate in the subscriber/publisher mode, and the dashed arrows represent a client/server paradigm.

PERCEPTION: This stage perceives data from the sensors, processes such data to extract estimated states of the environment and the LGV. This stage usually consists of two computation nodes: *Localization* is responsible for determining the LGV's position; *Costmap Generation* is for modeling the LGV's surroundings with costmaps to maintain the navigation information of the LGV.

PLANNING: This stage is responsible for determining the long-range actions of the LGV based on high-level goals specified by users. It also consists of two nodes: *Path Planning* (PP) identifies the shortest path from start position to each destination in a known map; *Exploration* node searches for all user-defined accessible regions in absence of costmaps.

CONTROL: This stage processes the execution action and forwards these motion commands to the actuators in the control subsystem. The *Path Tracking* node produces velocity commands to follow the planned path given the costmap. The *Velocity Multiplexer* node selects the velocity from multiple commands based on user-defined priorities.

Although these functional nodes can be implemented by different libraries, according to whether the map is known, all LGV workloads can be classified into two typical categories:

Navigation with a map. *CostmapGen* uses existing map data to create a costmap of its surroundings (①②) and *Localization* estimates the LGV's position from the sensor data (③④). Based on the position and the costmap, *Path Planning* generates an efficient collision-free path to the destination (⑤). *Path Tracking* follows this path and outputs the best action from simulating multiple trajectories with different velocities to guarantee feasibility and robustness (⑥), such as obstacle avoidance and oscillation. Considering robot's kinematics and dynamics, some other velocity commands (e.g. safe controller, joystick) are all forwarded to *Velocity Multiplexer* with different priorities, and the final velocity command is sent to the actuators with the highest priority (⑦).

Exploration without a map. To navigate the LGV in an unknown area, *localization* executes Simultaneous Localization and Mapping (SLAM) algorithm to infer the LGV's position in absence of a map. Then, *Exploration* selects a position in the frontier of known map as a destination and sends the goal to *Path Planning* (⑧⑨). By repeating this process of costmap update and exploration, the map of the environment will be expanded by publishing the boundary between the "known" and "unknown" regions, until the entire area has been mapped.

III. GOAL ANALYSIS AND OPTIMIZATION

Cloud robotics [5], [6] provide a new opportunity to optimize computation energy and mission completion time. Instead of deploying all computations in resource-constrained LGVs, some workloads can be offloaded to the remote servers. Then the total energy is saved as the migrated nodes do not need to consume the LGV's energy. The mission time is also shortened as the cloud servers can process the computation at a faster speed. However, cloud offloading can also bring extra costs: more energy is consumed by the data transmission, and network latency is introduced to increase the mission time. How to dynamically figure out the sweet point is critical.

A. Computation Modeling and Analysis

We assume that an LGV runs $N + M$ computational nodes (i.e. processes) to execute one workload. Specifically, N nodes are executed in the embedded computer of the LGV and M nodes are offloaded to the cloud servers.

Energy Consumption. The total energy consumption of executing this workload E_{total} is the sum of the energy consumed by the LGV E^R_{total} and the energy consumption during data transmission E_{trans} (Equation 1a).

We first analyze the data transmission. E_{trans} is mainly consumed by the wireless controller to transmit and receive data. Similar to existing works [31]–[33], we ignore the receiving energy consumption as the size of the received

TABLE I
MAXIMUM POWER CONSUMPTION OF EACH COMPONENT (WATT).

LGV	Sensor	Motor	Micro-controller	Embedded Computer
Turtlebot2	2.5 (8%)	9 (30%)	4.6 (14%)	15 (48%)
Turtlebot3	1 (6.5%)	6.7 (44%)	1 (6.5%)	6.5 (43%)
Pioneer 3DX	0.82 (3%)	10.6 (34%)	4.6 (15%)	15 (48%)

data is much smaller in LGV workloads (e.g. 48B velocity commands). Thus, E_{trans} can be approximately expressed by Equation 1b. Here, P_{trans} denotes the transmission power of wireless controller; T_{trans} denotes the total transmission time, which is determined by the size of transmission data D_{trans} and the uplink data rate for computation offloading R_{uplink}.

The onboard energy E^R_{total} is consumed by the sensors, motors, microcontrollers and embedded computers. Table I shows the power consumption of each hardware component in three commodity LGVs. We can observe that the majority of power consumption is dedicated to motors and the embedded computer. So we approximate E^R_{total} as the sum of the computation energy consumption E_{ec} and the motor energy consumption E_m (Equation 1a).

$$E_{total} = E^R_{total} + E_{trans} \sim E_{ec} + E_m + E_{trans} \quad (1a)$$

$$E_{trans} \sim P_{trans} T_{trans} = P_{trans} D_{trans} / R_{uplink} \quad (1b)$$

$$E_{ec} = \int_0^T \sum_{n=1}^{N} P^n_c(t) dt = \int_0^T \sum_{n=1}^{N} (k L_{n,t} {f_t}^2) dt \quad (1c)$$

$$E_m = \int_0^T P_m(t) dt = \int_0^T (P_l + m(a + g\mu)v) dt \quad (1d)$$

The energy consumption E_{ec} can be modeled by Equation 1c [31][2]. Here k is the effective switched capacitance, determined by the chip architecture; $L_{n,t}$ denotes the computation resources (i.e., number of cycles) this workload requires from node n at time t; f_t denotes the CPU frequency of the LGV at time t. The energy consumption E_m can be expressed by Equation 1d [34]. Here P_l denotes the transforming loss. m, v and a are the LGV's mass, velocity and acceleration. The ground friction constant is μ and the gravity constant is g.

Mission Completion Time. This consists of two parts: standby time T_s and moving time T_m (Equation 2a).

$$T = T_s + T_m \quad (2a)$$

$$T_s \sim t_p = t^R_p + t^C_p + t_c \quad (2b)$$

$$T_m \sim 1/v_{max} = 1/[a_{max}(\sqrt{t_p^2 + 2d/a_{max}} - t_p)] \quad (2c)$$

Standby time T_s measures the time an LGV suspends during a task. It exists as the computation capacity cannot meet the workload's requirements. When the on-board computation capacity is lower, the LGV needs a longer processing time t_p, and the LGV has to stay standby for a longer time. Thus, the processing time t_p can be denoted by the sum of three parts, i.e. the processing time of nodes in the robot t^R_p, the processing time of nodes in the cloud server t^C_p, and the network latency t_c (Equation 2b).

Moving time T_m measures the time an LGV moves along the path. It is highly dependent on the maximum velocity of an LGV: the faster the speed is, the less time it will spend on moving (Equation 2c). The maximum velocity is determined by the computation capacity. A higher computation capacity help shorten the processing time t_p and increase the maximum velocity. The maximum velocity is also affected by the maximum acceleration limit a_{max}, and required stopping distance d due to the obstacle avoidance constraint [35]. These are determined by the LGV's mechanical characteristics, and are not considered in this paper.

[2]Note that this equation is different from the traditional DVFS scaling rule: $P \sim V^2 f$ where V is the supply voltage and f is the clock frequency. In our model, we assume the voltage is constant

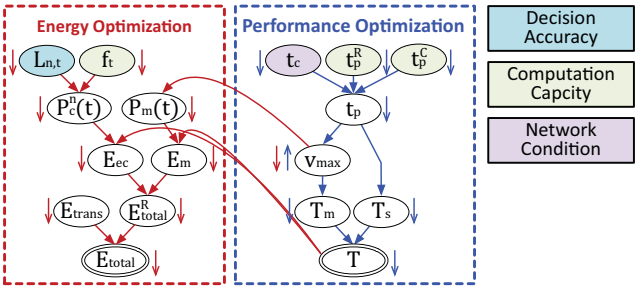

Fig. 3. Key factors and relationships with total energy consumption and completion time. ↑ means this factor is positive correlated to energy or completion time, while ↓ means the factor is negative correlated.

Analysis. Figure 3 summarizes the key factors and their relationships with the final optimization goals. We use arrows to show whether they are positive or negative correlated to the final goals. We also display that some factors are related to the task decision accuracy (cyan), computation capacity (green), or network condition (purple).

From this figure, we observe that optimization is non-trivial: (1) The two goals are coupling with each other. Optimizing one goal can also alter the other one. (2) There can be conflicts when tuning the parameters: reduction of E_m requires both reduction of T and $P_m(t)$, but the former is proportional to v_{max} and the latter is in an inverse ratio to v_{max}. (3) There are few factors we can alter for optimization. LGVs are commonly equipped with low-end embedded processors with low frequency, so t^R_p and f_t are commonly non-adjustable. Reducing $L_{n,t}$ inside the algorithm [36] can decrease the decision accuracy and become ineffective in complex environments.

B. Possible Optimization Approaches

Although optimization is challenging, we can still identify several possible approaches in the cloud offloading system.

Fine-grained migration. Cloud offloading can achieve the two goals simultaneously through decreasing $P^n_c(t)$ and t^R_p. However, it is not advisable to migrate all the computation nodes to the cloud due to two reasons. First, certain lightweight nodes have negligible impacts on the performance and energy consumption of the LGV devices. So there is no need to take the effort to upload those nodes to the cloud. Second, poor network quality can cause high packet loss rate, and even interrupt the LGV workloads entirely. So offloading all nodes would bring too much uncertainties in the complex environment. Then a fine-grained migration policy is necessary. We need to address this question: *which computation nodes should be migrated to the remote servers?* (§ IV)

Cloud acceleration. While it is difficult to optimize the computation on the LGV, there still exists opportunities to accelerate the node execution in the cloud servers. This can effectively reduce t^C_p for the mission completion time and energy consumption. The question is: *how to utilize the*

TABLE II
CYCLE BREAKDOWN OF EACH WORK NODE (GIGACYCLES). THE CONFIGURATION IN OUR EXPERIMENTS CAN BE FOUND IN § VIII-A.

Category	Perception			Planning		Control		Energy Critical Nodes
	Localization		*CostmapGen*	*Path Planning*	*Exploration*	*Path Tracking*	*Velocity Multiplexer*	
	Laser	SLAM						
With a Map	0.028 (1%)		0.857 (37%)	0.055 (2%)		1.385 (60%)	-	*CostmapGen* *Path Tracking*
Without a Map		3.327 (62%)	0.685 (12%)	0.052 (1%)	0.011 (1%)	1.207 (23%)	-	*CostmapGen* *Path Tracking*, SLAM

hardware architectures and resources from the cloud side to further improve the efficiency? (§ V)

Real-time adjustment. The network latency t_c plays an important role in determining the effectiveness of cloud offloading. Under certain network conditions and environments, the benefit from computation acceleration in the cloud may be not enough to offset the extra cost by network latency. Even worse, the LGV may move to somewhere far away from WAP and disconnect with the cloud. Then it is not advisable to perform cloud offloading. Although we cannot control the network conditions actively, we should dynamically monitor the network quality and adjust the offloading strategies adaptively. Then the question we need to consider is: *how to measure real-time network quality and achieve robustness under poor network conditions?* (§ VI)

IV. FINE-GRAINED MIGRATION

Our first optimization is a fine-grained migration policy. Current platforms [8], [9] delegate the offloading decision to programmers. From above analysis, the optimization of these performance and energy is complicated, thus those non-expert robotics programmers actually encounter more challenges. In this section, we present an approach to automatically select the optimal computation nodes for cloud offloading.

A. Bottleneck Identification

We introduce two concepts to help us better understand the computation characteristics of LGV workloads, and identify the performance and energy bottlenecks.

Energy-Critical Node (ECN): This denotes the node that consumes a major portion of the total energy in the workload. This is the energy bottleneck of an LGV workload.

According to Equation 1c, the energy consumption per unit time at each node is proportional to the workload CPU cycles. Table II shows the CPU cycle breakdown of each node in two types of LGV workloads (measured at 1.6GHz, 4 low-power cores). From this table we can observe that the majority of execution cycles is dedicated to *CostmapGen*, *Path Tracking* and *Localization* (SLAM). So we can conclude that those three nodes are ECNs in the LGV workloads.

Velocity-Dependent Path (VDP): This denotes the longest velocity-dependent execution flow path. The total processing time of all nodes along the VDP determines the LGV's maximum velocity (Equation 2c), and thus the mission completion time. It is the performance bottleneck of an LGV workload.

As Figure 2 shows, once receiving the laser data, *CostmapGen* detects and marks obstacles in the costmap. Then *Path Tracking* generates a collision-free path, sends the adjusted velocity to *Velocity Multiplexer* and further forwards to the motors. So the execution path along *CostmapGen*, *Path Tracking* and *Velocity Multiplexer* is the VDP in the LGV workloads.

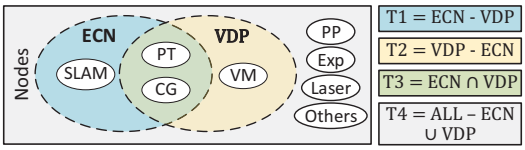

Fig. 4. Nodes classification of LGV apps.

B. Node Selection and Offloading Strategy

Figure 4 summarizes the relationship between ECN and VDP of the LGV workloads. Different sets of nodes will be selected for different goals.

Reducing energy consumption: As discussed in § III-A, total computation power E_{ec} is mainly determined by $P_c^n(t)$ and T. So it is necessary to migrate all ECNs (T1+T3) to the cloud for energy reduction. The rest lightweight nodes (T2+T4) can be executed on the LGV.

Shortening completion time: Mission completion time depends on the overall processing time of all nodes along the VDP[3]. So it is necessary to migrate all ECNs inside VDP (T3) to the cloud for performance acceleration. The rest nodes (T2) can be kept on the local LGV, as their processing time will not be improved even when they are hosted on the cloud server.

However, network latency has to be considered when optimizing the completion time: if extra network latency due to node migration is larger than the shortened completion time, then it is not necessary to conduct offloading. Specifically, we denote T_l^v as the overall VDP node processing time when all nodes are local, and T_c as the sum of VDP node processing time and real-time network latency when T3 nodes are offloaded to cloud. If the network quality is poor such that $T_c > T_l^v$, then we should just conduct the computation of all VDP nodes locally.

Algorithm 1 describes our offloading strategy. We provide two optional optimizing goal for programmers: (1) Reducing energy consumption (EC): all the T1+ T3 nodes which have heavy computation overhead will be offloaded to the cloud. (2) Shortening mission completion time (MCT): we first submit all ECNs to the cloud, and compare the local VDP time T_l^v with cloud VDP time T_c. If $T_c > T_l^v$ due to a high-cost

[3]We do not consider the impact of network bandwidth because the size of the transferred data is very small such that the network bandwidth does not fluctuate significantly according to our experiments. The high-frequency loss of data transfers can affect the mission completion time when the LGV moves far away from the wireless access point, which will be discussed in § VI.

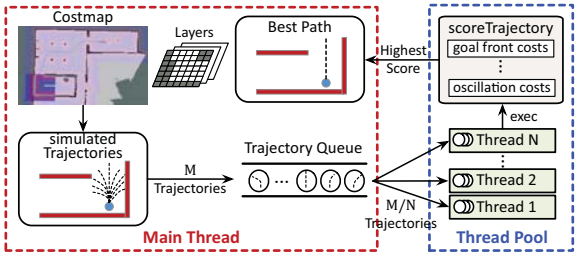

Fig. 5. Parallel path tracking algorithm with N threads acceleration. It crops the sensor data and static map to update the costmap and produces velocity.

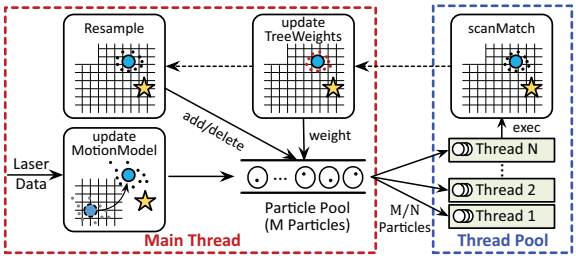

Fig. 6. Parallel gmapping algorithm with N threads acceleration. Each thread execute the scanMatch function to process M/N particles. The dotted circles and gray solid circles denote the most recent pose and particles respectively, while blue circles and black solid circles denote current pose and particles respectively. The yellow stars represent the landmarks.

network latency, we will migrate all T3 nodes back to the local LGV. The maximum velocity is set based on the overall VDP processing time (Equation 2c).

Algorithm 1 Offloading Strategy

INPUT:
- N^t ▷ A set of nodes to be scheduled at time t
- T_l^v ▷ Local VDP time at maximum velocity v
- T_c ▷ Cloud VDP time (including network latency)
- G ▷ Optimization Goal (EC/MCT)

BEGIN:
```
submit all nodes ∈ ECN to the remote server
while N^t != φ do
    let n_i be the nodes ∈ T3
    if T_c > T_l^v and G == MCT then
        migrate n_i to LGV
    end if
end while
set new maximum velocity velocityOA(T_c) according to Equation 2c
```

V. Cloud Acceleration

Our second optimization is to accelerate the node execution in the cloud. This can reduce the on-board energy consumption, and end-to-end mission completion time. Although existing cloud-robotic services provide execution acceleration [11]–[17], they only focus on the algorithm level. In this section, we propose some approaches to optimize different work nodes utilizing the architectural features of remote servers and discuss the optimal selection of different architectural configurations in the optimization.

CostmapGen and Path Tracking. As discussed in § III-A, The acceleration in computational-intensive nodes along VDP is crucial to reduce mission completion time. To navigate along a planned path in an obstacle-filled environment, the LGV needs a costmap to represent the knowledge of geometric world and a path tracker to compute the optimal velocity. A costmap uses the static map, laser data and LGV's footprint to generate multiple layers to store and update the information about the obstacles, including a static map layer, obstacle map layer and inflation layer. Based on the costmap and the global path generated from *Path Planning*, *Path Tracking* simulates multiple possible trajectories based on the current velocity. For each possible velocity, it performs forward simulation to generate N trajectories. To find the best path, it scores each trajectory using a cost function that incorporates many characteristics, including proximity to the goal, to the global path, to obstacle and oscillation. After discarding all illegal trajectories (e.g. colliding with obstacles), the trajectory with the highest score will be selected and the corresponding velocity is sent to *Velocity Multiplexer*. Thus, we can conclude that the high computation burden for *Path Tracking* is derived from two factors: the sequentially performed duplicated scoring work and the number of trajectories.

Based on this observation, we propose an efficient path tracking algorithm exploiting the manycore characteristic of the cloud servers. Figure 5 describes how our algorithm can parallelize the execution of scoreTrajectory process. Specifically, we set up a thread pool with N threads. Once the main thread of path tracking generates M trajectories, we partition these trajectories into N parts and assign each one to a thread in the pool for the scoring operation. After all threads complete their works, we choose the trajectory with the highest score as the best path.

SLAM. Although accelerating SLAM on remote servers makes no benefits to either energy consumption or mission time, the reduction of SLAM processing time can decrease the rate of mission failure caused by wrong decision after receiving an obsolete data (i.e. pose) from the SLAM node. The main idea of the SLAM algorithm is to maintain a set of particles M to estimate the real pose before mapping. Each particle represents a possible pose of a robot and is encoded by a potential trajectory and the related occupancy grid map. Thus, the number of particles M determines the decision accuracy of the gmapping algorithm. Using the timestamp method, we can verify that 98% of the computation time of SLAM is spent on the scanMatch function, which involves many mathematical operations in processing M particles repeatedly. Thus, we conclude that the high computation burden on the SLAM node is derived from two factors: the sequentially performed duplicated matching work and the number of particles.

We designed an efficient parallel gmapping algorithm, which exploits the manycore server features to accelerate the SLAM processing time. Figure 6 briefly shows the mechanism of our algorithm for parallelizing the execution of the scanMatch process. Specifically, we set up a thread pool with N threads and each thread is responsible for operations of M/N particles. Therefore, once receiving the laser data, each thread in the thread pool will subscribe to one copy of

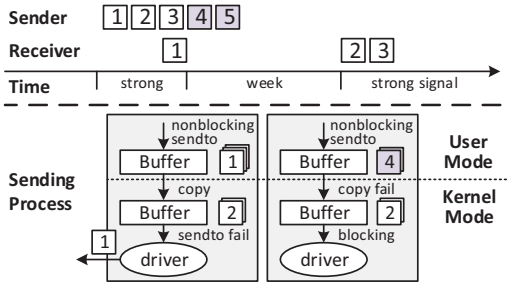

Fig. 7. An example of traditional UDP communication pattern under unstable wireless network and the detailed implementation of sending process. The discarded packets are depicted with purple squares.

the data and execute the scanMatch function to update the latest state of the partial particles. After all particles in the particle pool are processed, the main thread continues to call updateTreeWeights and resample functions sequentially.

VI. Real-time Adjustment

The third optimization we introduce is to dynamically monitor the workload execution and environment, and alter the offloading policy when possible. This is particularly important for achieving strong robustness, but rarely considered in existing platforms. During the task the LGVs are moving to different places, and it is possible to locate in a place with rather bad network quality. So we need an adaptive solution to ensure that the LGV workload can be executed smoothly for the entire process.

Challenges. To achieve network robustness, we first need to accurately evaluate and measure the real-time network quality. We focus on the communication between nodes along VDP because both the offloading decision and the maximum velocity depend on the VDP makespan. To ensure *real-time* communication and data *freshness*, these nodes commonly use an efficient UDP communication pattern with a one-length queue. Previous researches [37] used tail latency (99th-, 99.99th-) or worst-case latency as metrics to evaluate the network quality. Specifically, they collected end-to-end communication timestamps and predicted the network quality from history data. These metrics can work well in TCP pattern which hides packet loss and disorder in the communication timestamps, but fail to give a fair evaluation for UDP pattern.

Figure 7 illustrates a case that traditional tail latency or worst-case latency cannot be applied to cloud-robotic networks. An LGV transmits five packets to the remote cloud sequentially within a period of time. First, the LGV issues the system call *sendto* to copy the data of packet 1 from the user buffer to the kernel buffer. Then, it calls the wireless driver function *sendto* to transmit the data to the cloud. After that, the driver detects that the quality of the signal is weak while sending packets 2 and 3. It blocks the kernel buffer and holds on the data until it detects that the wireless signal becomes strong. Due to the nonblocking configuration of the socket, packets 4 and 5 fail to be copied into the full kernel buffer and are discarded. Thus, from the cloud's side, both tail latency and worst-case latency will reveal a good network condition when receiving packet 1 and the current policy will be maintained. In this case, the LGV will stop at the time of weak signal forever or spend much time to restart mission without state migration.

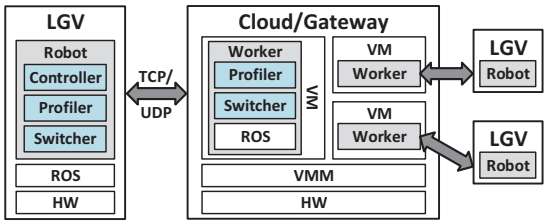

Fig. 8. System overview of cloud-robotic offloading.

A. Offload Network Quality Control

We propose an offload network quality control strategy for accurate real-time network quality prediction, and node switching. As shown in Algorithm 2, we use packet bandwidth (i.e. receiving packet rate) and signal direction (i.e. LGV moving direction relative to WAP) as metrics instead of latency. The signal direction is built on the position of the WAP marked in the LGV's internal model of the environment. Due to the stable sending rate, the packet bandwidth can predict the future network quality by reflecting the packet loss in a period of time (e.g. 1s). The signal direction can help us make switching decision by monitoring the variation of directions between the LGV and the WAP. In the above example, if sending rate is 5Hz and the LGV moves away from WAP, we will achieve a 1Hz bandwidth at weak signal and a negative signal direction. Then, the LGV will invoke offloaded computation nodes locally and migrate related states back from the cloud.

VII. System Design

We design and implement an end-to-end cloud-robotic system to realize our optimization strategies. Figure 8 shows the architectural overview of our prototype. The system includes two main entities: a local LGV, and the remote edge gateway or cloud server. The local LGV runs Robot Operating System (ROS) [38], which is the most popular open-source robotic middleware for programming abstraction. A ROBOT system module runs on top of ROS, consisting of three threads: Controller, Profiler and Switcher. The remote server hosts a Virtual Machine Manager (VMM), with multiple Virtual

Algorithm 2 Offload Network Quality Control

INPUT:
N^t ▷ A set of nodes executed in remote server at time t
r^t ▷ Packet bandwidth at time t
d^t ▷ Signal direction between LGV and WAP at time t
BEGIN: ▷ We focus on these two cases caused by mobility.
if $r^t < threshold$ and $d^t < 0$ **then**
invoke N^t on LGV locally
else if $r^t > threshold$ and $d^t > 0$ **then**
invoke N^t on remote server
end if

TABLE III
COMPUTING OFFLOADING PLATFORM SPECIFICATIONS.

	Turtlebot 3	Edge Gateway	Cloud Server
Model	Raspberry Pi 3 B+	Intel i7-7700K	Intel Xeon Gold 6149
Frequency	1.4 GHz	4.2 GHz	3.1 GHz
Cores	4	4	24
Memory	1 GB	16 GB	768 GB
Sensor	LDS-01	-	-
Feature	Low Freq	High Freq	Manycore

Machines (VMs). Each VM runs a WORKER system module, consisting of the same Profiler and Switcher threads as the LGV. Each VM is connected to an LGV via TCP/UDP. The functionalities of different threads are explained below:

Switcher. This is the main thread that maintains data communication between different worker nodes deployed in the local LGV and the remote server. Specifically, it attaches temporal information to each ROS message and then sends the message to the receiver with a serialized data structures. We implement the switcher with a C++ library called `evpp` [39] that provides a multi-threaded nonblocking RPC model using asynchronous I/O mechanisms. We also use `protobuf` [40] to serialize ROS message for efficient data transmission.

Profiler. This module collects relevant data for making offloading decisions in Algorithms 1 and 2. (1) Processing time: it records the timestamp of each node along VDP and publishes them to the corresponding topic. (2) Network latency: it measures the uplink time and downlink time by monitoring the packet round trip time (RTT). When the remote switcher receives messages from the local switcher, it attaches the subscribed processing time of the cloud worker nodes and returns to the local switcher. Thus, the VDP makespan can be calculated as the sum of received cloud processing time, subscribed local processing time and RTT. (3) Bandwidth: it counts the number of local received messages during a fixed period. (4) Signal direction: it estimates the variation of distance between the local LGV and WAP as a bool value.

Controller. This specifies the configuration parameters of functional worker nodes for computation offloading and robustness at runtime. Specifically, it exposes interfaces of decision accuracy and maximum velocity adjustment through ROS APIs, and uses profiling data to make corresponding actions based on our strategies.

VIII. EVALUATION

A. Experiment Setup

We select a representative LGV, Turtlebot3, and two types of remote platforms with different hardware configurations. Table III details the specifications of hardware resources. Specifically, the Turtlebot3 is a low-end LGV equipped with a Raspberry Pi CPU@1.3GHz, 1 GB memory and a 360 Laser Distance Sensor (LDS). It is connected to either a high-frequency edge gateway (Intel i7-7700K CPU@4.2GHz with 16GB of RAM) in our lab or a virtual machine (Intel Xeon Gold 6149 CPU@3.1GHz with 758GB of RAM) from a public cloud provider with a passive 5GHz band wireless network.

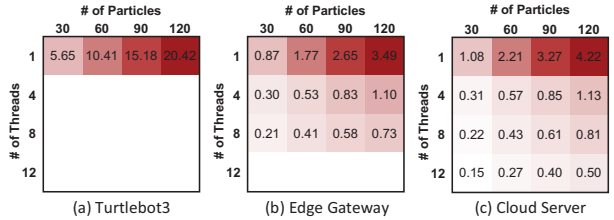

Fig. 9. Processing time (s) of energy critical node (SLAM) under different numbers of threads and particles.

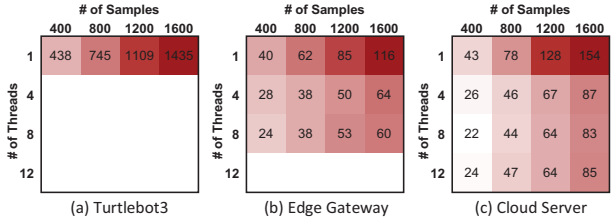

Fig. 10. Processing time (ms) of velocity dependent path (CG + PT + VM) under different numbers of threads and particles.

For the computation nodes, we choose Adaptive Monte Carlo localization (AMCL) [41] as the laser-based localization algorithm for the known map scenario, and GMapping [42] as the SLAM algorithm to build a 2D occupancy grid map of LGV's surroundings for the unknown map scenario. We adopt the CostMap 2D package [43], a multi-layer tracking and updating algorithm to mark and clear obstacles intelligently. We use ROS global planner [44] paired with the A* [45] and Dijkstra's [46] algorithms for path planning. We use the frontier-based algorithm [47] for autonomous exploration, an effective approach for LGVs to extend their maps by moving to new frontiers (i.e. regions on the boundary between observable space and unexplored space) and updating unknown regions. We use ROS local planner [48] integrated with the Trajectory Rollout and DynamicWindow [49] algorithm to drive LGVs in the plane. We choose Yujin Robot's open-source control system [50] for velocity multiplexer.

B. Cloud Acceleration Improvement

We use Intel Research Lab [51] as a representative dataset of LGV workloads to examine the effects of our cloud acceleration strategy. We measure the processing time with different numbers of threads (parallelization), and particles (computation complexity). Figures 9 and 10 show the processing time of ECN and VDP respectively. For each figure, we report the processing time when the nodes are (a) local, (b) migrated to the edge gateway, or (c) to the cloud server.

ECN: As the numbers of threads and particles scale up, we observe that the processing time is reduced by up to $27.97\times$ and $40.84\times$ when the nodes are offloaded to the gateway and cloud server, respectively (Figure 9). As described in § V, increasing the number of particles can improve the decision accuracy of the mapping algorithm at a higher computation cost. So, a larger number of threads can achieve a higher processing time reduction when the number of particles is large. Besides, we observe that the cloud server with the manycore feature has a better acceleration improvement for ECN offloading.

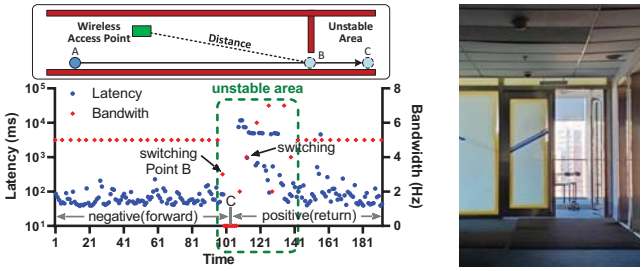

Fig. 11. Network latency and bandwidth of UDP transmission in a wireless network. We monitor the real-time communication latency (blue rhombus), bandwidth (red dot) of velocity messages and signal direction.

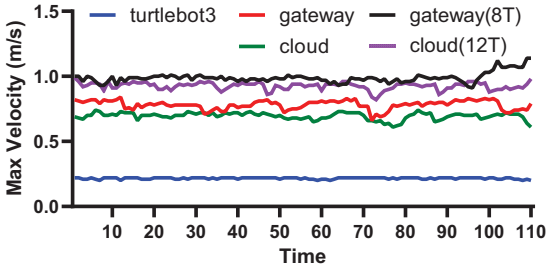

Fig. 12. The maximum velocity of the LGV in a navigation workload with different platforms and strategies

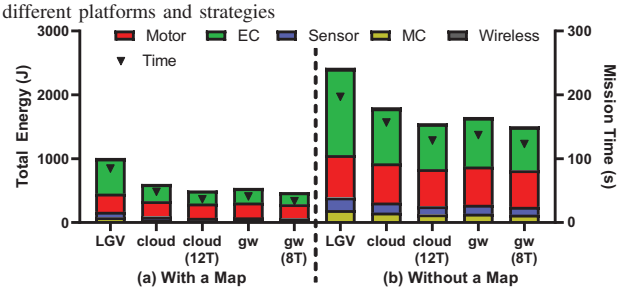

Fig. 13. The total energy consumption and mission completion time.

VDP: The processing time of VDP affects the maximum velocity selection. As described in § V, the number of samples determines the number of simulated trajectories. Thus, high processing time in low-frequency embedded board of LGV limits not only the maximum velocity, but also the decision accuracy. We achieve a processing time reduction of up to $23.92\times$ and $17.29\times$ for optimization across the gateway and cloud server (Figure 10). It is interesting to observe that parallelization has no impact on the processing time when the number of threads is larger than 4, as the computation overhead in each thread is very small. Besides, we can observe that the edge gateway with the high-frequency feature has a better acceleration improvement for VDP offloading.

C. Network Robustness

We evaluate the effectiveness of our real-time adjustment strategy under poor network quality in physical world. Figure 11 shows the network latency and bandwidth when the LGV moves from point A to C and then returns to A. Note that the designated point C is in an unstable area far away from the WAP. From the figure, we can observe that bandwidth can accurately reflect the packet loss rate due to the fixed sending rate (5Hz) of the *Path Tracking* node deployed in the cloud server. We set the threshold as 4 in Algorithm 2.

When the LGV moves to the unstable area (green dashed box), the latency increases while the bandwidth decreases. The critical point that divides the negative and positive signal direction is the designated point (C) of the LGV's entire path. Note that before the LGV enters into the unstable area under weak signal quality (Point B), the communication latency cannot reflect the network quality correctly due to the "best-effort delivery" of UDP as discussed in § VI. Based on our strategy, we can predict the network quality through the reduction of bandwidth and make local migration decisions with negative signal direction. When the LGV returns, it makes cloud migrating decisions based on the increased bandwidth and positive signal direction. This decision also depends on our offloading strategy in Algorithm 1.

D. End-to-End performance

To estimate the end-to-end performance improvement of our offloading strategy, we investigate the maximum velocity increase, total energy saving and mission time reduction under different hardware platforms. Specifically, we first instruct our LGV to explore in our lab and make a 2D grid occupied map. Then we perform the navigation evaluation with a designated position on the known map.

Maximum velocity increase: Figure 12 presents the maximum velocity breakdown during a period of time based on our offloading strategy in Algorithm 1. Specifically, we analyze five offloading cases in our experiments: no offloading (blue line), offloading to the gateway without optimization (red line), offloading to the gateway with 8 threads parallelization (black line), offloading to the cloud without optimization (green line) and offloading to the cloud with 12 threads parallelization (purple line).

It is clear to see that without any computation offloading the LGV moves in a slow speed due to limited on-boarding resources. With computation offloading and parallel optimizations, the maximum velocity of the LGV can be remarkably increased by 4-5 times. Note that the maximum velocity of each offloading strategy fluctuates more widely than no offloading. This is caused by the unstable network latency, including both the wireless latency between the LGV and WAP, and the wired latency between our lab and the cloud.

Energy saving and mission time reduction: As Figure 13 shows, we model energy consumption as the sum of five hardware components of the LGV: motor, sensor, microcontroller, embedded computer and wireless controller. We measure the total energy consumption through connecting a power meter on the battery, and estimate the energy consumption of each component using the power models from [34], [52]. The bars with different colors in this figure denote the total energy consumption of each component respectively. In addition, we also mark the mission completion time of each workload with different computation deployment strategies.

From the figure, we can observe our offloading strategy can significantly reduce the total energy consumption by factors

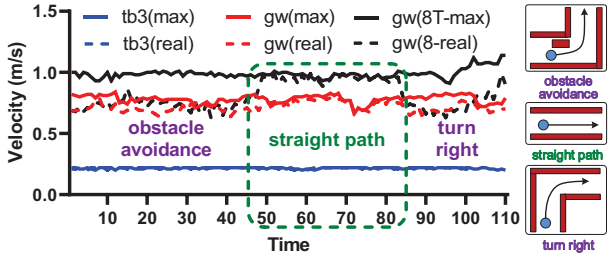

Fig. 14. The relationship between maximum velocity and real velocity.

of 1.61×, 2.12× and mission completion time by factors of 2.53×, 1.6× respectively. Although computation deployment on remote servers can significantly reduce the energy consumption of the embedded computer (green bar) and mission completion time (triangle), there is almost no performance improvement on motor energy (red bar). This is because motor energy is proportional to the LGV's velocity. Although we can reduce the mission completion time by increasing its velocity, the benefit is balanced by the increase of real-time motor energy. Beisdes, the energy consumption in wireless controller is small due to the small size of D_{trans} in Equation 1b, whose maximum size is 2.94KB (laser scan).

Moreover, the total energy reduction of the workload without a map is higher than that of the workload with a map, because the computational-intensive SLAM node runs out the resources. As such, the embedded computer consumes a larger portion of the overall energy. Besides, due to a larger number of curves and uncertainties in the path of the workload without a map, the LGV drives at a slower velocity for safety. Thus, its mission completion time reduction is lower than that of the workload with a map.

E. Adaptivity Analysis

While we maximize the energy benefit by dynamically adjusting the maximum velocity, there is a gap between the maximum velocity and the real velocity due to the mobility of the LGV. Different from stationary servers, the LGV interacts with the dynamic and unstructured environment when executing workloads. Considering the obstacles in a map, the planned path can be straight, arc, or U-shaped line. Figure 14 shows the relationship between the maximum velocity (solid line) and the real velocity (dotted line) when the LGV moves in a complex real world with many obstacles. The traveling path includes three phases: avoiding obstacles (Time 0-45), heading straight (Time 45-85) and turning right (Time 85-110).

Obviously, only when the LGV follows a straight path (green square), the real velocity can reach the maximum velocity. When the LGV faces obstacles or needs to make a turn, the real velocity will decrease. *The higher maximum velocity is set, the bigger gap between the real and maximum velocity will be expanded*. Driving at relatively low speeds (blue line) can bridge the gap caused by different phases. Thus, we can adopt the optimal offloading policy which has a minimum gap based on different phases of environment. For instance, if there are more obstacles in the environment, we can reduce the parallelization since the LGV cannot reach the maximum velocity. This can also save the financial cost and resource usage on the cloud servers or edge gateway.

IX. Discussion and Future Work

Theoretical Guarantee . In this paper, we empirically validate our proposed solution with real-world devices and environments. How to theoretically and formally verify the effectiveness of the optimization is a challenging and open problem, as the strategy depends on not only the electronic features of embedded computers, but also the domain knowledge of modeling both kinematics and dynamics of robotics. This process can introduce high complexity and stochasticity. To the best of our knowledge, most relevant solutions only adopt the empirical studies [36], [37], [53]–[55], and there are very few works providing theoretical guarantee. This will be an important and promising direction of our future work.

Other robotic devices . In this paper, we focus on the offloading optimization of LGVs. Our strategies can be applied to other autonomous devices as well, with certain changes. For instance, autonomous vehicles and drones have stronger safety requirements since they move in a more dynamic environment with much high speeds than LGVs. These safety requirements set a strict constraint on processing time to guarantee these vehicles react to real-time conditions promptly. So in the fine-grained migration, we need to keep those safety-critical nodes (e.g., obstacle avoidance) in the vehicle other than being offloaded to the cloud. Performance and energy optimization in the cloud can be further optimized following our strategies.

We conduct analysis and experiment on laser-based LGVs, which are sensitive to computation and energy due to the low-end embedded computer and limited battery. Our strategies can adapt to vision-based LGVs as well, since they share the standard pipeline and computation model with laser-based LGVs. The only difference is that the localization failure effect needs to be considered: the vision-based LGV estimates its pose by tracking a set of points/features through successive camera frames. A slower speed is needed to prevent the localization failure due to the high rate of environment changes.

Alternative parallelization methods. We identify ECNs and parallelize their computational processes through multiple threads. Despite this method is algorithm specific, it is generally effective in most of LGV workloads, as the particle-based localization and mapping are dominant in the robotic domain [56]. Other parallelization methods for vision-based SLAM have been widely studied in recent work [57], [58], we plan to integrate these methods to our system to verify the end-to-end performance benefit in the future.

X. Related Work

Fine-grained Migration. Various cloud-based systems were built to facilitate the development of robotic workloads [8]–[10]. They are used to help programmers deploy computations across cloud and robots. However, those non-expert robotics programmers need to manually make the migration strategies with their expertise, which is not automatic or efficient.

Cloud Acceleration. On top of such cloud-robotic platforms, cloud services were introduced to accelerate different robotic functions, e.g., grasp planning [16]–[18], object recognition [12], [13], SLAM [11], [59]–[62]. However, these works focus reducing the execution time of specific algorithms. They did not consider the benefits of systematic performance and energy efficiency of the entire mission from cloud acceleration.

Real-time Adjustment. To achieve robustness under poor network condition, previous works focus on access point selection, which automatically chooses an available network among multiple communication links based on the bandwidth assessment [63]–[67]. However, this method cannot work when there are no multiple optional communication links.

Robotic Workloads Analysis. Previous works designed methods and benchmarks to analyze the energy and performance of drones [36] and autonomous driving [37], [53], but they did not investigate optimizations on the cloud-robotic scenario. Rahman *et al*. [68] proposed a genetic algorithm for energy and mission completion time optimization of oil factory maintenance application. Their method is based on a 3-layer decision: task offloading, path planning, and access point selection. However, this method requires to model the factory environment, and its application is limited to the scenario where the environment is static and known. This work also did not consider cloud acceleration or velocity adjustment for better optimization. In contrast, our proposed method can be applied to general UGV tasks and scenarios, even when the environment is unknown and dynamic. Pandey *et al*. [69] presented a novel resource provisioning algorithm for Autonomous Underwater Vehicles (AUVs) applications to benefit from the cloud resources. The objective of their algorithm is to minimize either the execution time or budget for cloud usage. This migration method did not consider energy consumption of each component in the robots and velocity control, or the cloud acceleration and real-time adjustment. Hence this solution is less optimized than our proposed work.

XI. Conclusion

This paper explores the opportunities of performance and energy optimization on LGV workloads with cloud offloading. To understand the computation role in total energy consumption and mission completion time, we build an analytical model to disclose the relationship among its constituted factors and identify the optimization bottlenecks in each workload. We propose three optimization strategies to select the optimal nodes for offloading, accelerate the execution in the cloud, and dynamically monitor and adjust the policies at runtime. We implement an end-to-end prototype on the Turtlebot3 vehicle. Evaluations indicate that this system can achieve significant performance and energy efficiency. We expect this work can encourage researchers from different communities to pay more attention to the cloud-robotic area.

XII. Acknowledgements

We thank the anonymous shepherd and reviewers for their valuable comments. This work was supported in part by Key-Area Research and Development Program of Guangdong Province (NO.2020B010164003), the National Natural Science Foundation of China (Grant No. 62090020, 61702480, 61672499), Youth Innovation Promotion Association of Chinese Academy of Sciences (2013073, 2020105), the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDC05030200), Singapore MoE AcRF Tier 1 RG108/19 (S) and NTU-Desay Research Program 2018-0980.

References

[1] "Amazon robootics," https://www.amazonrobotics.com/, 2019.
[2] "irobot vacuums," https://www.irobot.com/, 2019.
[3] "Disaster robotics research projec," http://www.tradr-project.eu/, 2019.
[4] "Turtlebot 3," http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/, 2016.
[5] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Trans. Automation Science and Engineering (T-ASE)*, vol. 12, no. 2, pp. 398–409, 2015.
[6] J. Wan, S. Tang, H. Yan, D. Li, S. Wang, and A. V. Vasilakos, "Cloud robotics: Current status and open issues," *IEEE Access*, vol. 4, pp. 2797–2807, 2016.
[7] I. Stoica, D. Song, R. A. Popa, D. A. Patterson, M. W. Mahoney, R. H. Katz, A. D. Joseph, M. I. Jordan, J. M. Hellerstein, J. E. Gonzalez, K. Goldberg, A. Ghodsi, D. Culler, and P. Abbeel, "A berkeley view of systems challenges for ai," in *CoRR abs/1712.05855*, 2017.
[8] "Amazon aws robomaker," https://aws.amazon.com/robomaker/, 2019.
[9] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *International Conference on Robotics and Automation (ICRA)*, 2013.
[10] B. L. Rajesh Arumugam, Vikas Reddy Enti, X. Wu, K. Baskaran, F. K. Foong, A. S. Kumar, D. M. Kang, and W. K. Goh, "Davinci: A cloud computing framework for service robots," in *Symposium on Cloud Computing (SoCC)*, 2013.
[11] L. Riazuelo, J. Civera, and J. M. M. Montiel, "C2tam: A cloud framework for cooperative tracking and mapping," in *Robotics and Autonomous Systems (RSS)*, 2014.
[12] W. J. Beksi, J. Spruth, and N. Papanikolopoulos, "Core: A cloud-based object recognition engine for robotics," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2015.
[13] B. Kehoe, A. Matsukawa, S. Candido, J. Kuffner, and K. Goldberg, "Cloud-based robot grasping with the google object recognition engine," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2013.
[14] L. Riazuelo, M. Tenorth, D. D. Marco, M. Salas, D. Gálvez-López, L. Mösenlechner, L. Kunze, M. Beetz, J. D. Tardós, L. Montano, and J. M. M. Montiel, "Roboearth semantic mapping: A cloud enabled knowledge-based approach," *IEEE Trans. Automation Science and Engineering (T-ASE)*, vol. 12, no. 2, pp. 432–443, 2015.
[15] B. Kehoe, D. Berenson, and K. Goldberg, "A cloud robot system using the dexterity network and berkeley robotics and automation as a service (brass)," in *IEEE Conference on Automation Science and Engineering (CASE)*, 2012.
[16] B. Kehoe, D. Warrier, S. Patil, and K. Goldberg, "Cloud-based grasp analysis and planning for toleranced parts using parallelized monte carlo sampling," *IEEE Trans. Automation Science and Engineering (T-ASE)*, vol. 12, no. 2, pp. 455–470, 2015.
[17] B. Kehoe, D. Berenson, and K. Goldberg, "Toward cloud-based grasping with uncertainty in shape: Estimating lower bounds on achieving force closure with zero-slip push grasps," in *International Conference on Robotics and Automation (ICRA)*, 2012.
[18] N. Tian, M. Matl, J. Mahler, Y. X. Zhou, S. Staszak, C. Correa, S. Zheng, Q. Li, R. Zhang, and K. Goldberg, "Estimating part tolerance bounds based on adaptive cloud-based grasp planning with slip," in *International Conference on Robotics and Automation (ICRA)*, 2017.
[19] L. Vasiliu, I. Trochidis, C. Bussler, and A. Koumpis, "Robobrain: A software architecture mapping the human brain," in *Humanoids*, 2014.
[20] M. Beetz, D. Beßler, A. Haidu, M. Pomarlan, A. K. Bozcuoğlu, and G. Bartels, "Know rob 2.0 - a 2nd generation knowledge processing framework for cognition-enabled robotic agents," in *International Conference on Robotics and Automation (ICRA)*, 2018.

[21] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, "Clonecloud: elastic execution between mobile device and cloud," in *European Conference on Computer Systems (EuroSys)*, 2011.

[22] E. Cuervo, A. Balasubramanian, D. ki Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, "Maui: making smartphones last longer with code offload," in *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2018.

[23] Y. Wen, W. Zhang, and H. Luo, "Energy-optimal mobile application execution: Taming resource-poor mobile devices with cloud clones," in *IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[24] S. Kosta, A. Aucinas, P. Hui, R. Mortier, and X. Zhang, "Thinkair: Dynamic resource allocation and parallel execution in the cloud for mobile code offloading," in *IEEE Conference on Computer Communications (INFOCOM)*, 2012.

[25] M.-R. Ra, A. Sheth, L. B. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: enabling interactive perception applications on mobile devices," in *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2011.

[26] Y.-W. Kwon and E. Tilevich, "Energy-efficient and fault-tolerant distributed mobile execution," in *IEEE Conference on Computer Communications (ICDCS)*, 2012.

[27] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, and B. L. Yupeng Qu, "etime: Energy-efficient transmission between cloud and mobile devices," in *IEEE Conference on Computer Communications (INFOCOM)*, 2013.

[28] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *USENIX Annual Technical Conference (ATC)*, 2010.

[29] "Turtlebot 2," https://www.turtlebot.com/turtlebot2/, 2012.

[30] Siciliano, Bruno, and O. Khatib, *Springer handbook of robotics*. Secaucus, NJ, USA: Sprinter-Verlag New York, Inc.: Springer, 2016.

[31] S. Guo, B. Xiao, Y. Yang, and Y. Yang, "Energy-efficient dynamic offloading and resource scheduling in mobile cloud computing," in *IEEE International Conference on Computer Communications (INFOCOMM)*, 2016.

[32] X. Chen, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Parallel and Distributed Systems (TPDS)*, vol. 26, no. 4, pp. 974–983, 2015.

[33] D. Huang, P. Wang, and D. Niyato, "Decentralized computation offloading game for mobile cloud computing," *IEEE Trans. Wireless Communications (TWC)*, vol. 11, no. 6, pp. 1991–1995, 2012.

[34] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, "A case study of mobile robot's energy consumption and conservation techniques," in *International Conference on Advanced Robotics (ICAR)*, 2005.

[35] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *International Conference on Robotics and Automation (ICRA)*, 2016.

[36] B. Boroujerdian, H. Genc, S. Krishnan, W. Cui, A. Faust, and V. J. Reddi, "Mavbench: Micro aerial vehicle benchmarking," in *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018.

[37] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars, "The architectural implications of autonomous driving: Constraints and acceleration," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[38] "Open source robot operating system," http://www.ros.org/, 2019.

[39] "evpp: A modern c++ network library for developing high performance network services in tcp/udp/http protocols," https://github.com/Qihoo360/evpp, 2018.

[40] "Google protocol buffers," https://developers.google.com/protocol-buffers/, 2019.

[41] D. Fox, "Kld-sampling: Adaptive particle filters," in *Neural Information Processing Systems (NIPS)*, 2001.

[42] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling," in *International Conference on Robotics and Automation (ICRA)*, 2005.

[43] "Ros costmap package," http://wiki.ros.org/costmap_2d, 2018.

[44] "Ros global planner package," http://wiki.ros.org/global_planner, 2018.

[45] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[46] D. E. W., "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[47] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Computational Intelligence in Robotics and Automation (CIRA)*, 1997.

[48] "Ros local planner package," http://wiki.ros.org/base_local_planner, 2018.

[49] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[50] "Yujin open control system (yocs)," https://github.com/yujinrobot/yujin_ocs, 2019.

[51] "Intel research lab 2d slam dataset," http://www.ipb.uni-bonn.de/datasets/, 2014.

[52] Y. Mei, Y.-H. Lu, Y. C. Hu, and C. G. Lee, "Deployment of mobile robots with energy and timing constraints," *IEEE Trans. Robotics*, vol. 22, no. 3, pp. 507–522, 2006.

[53] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," in *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[54] J. Sacks, D. Mahajan, R. C. Lawson, and H. Esmaeilzadeh, "Robox: An end-to-end solution to accelerate autonomous control in robotics," in *Annual International Symposium on Computer Architecture (ISCA)*, 2018.

[55] S. Murray, W. Floyd-Jones, Y. Qi, G. Konidaris, and D. J. Sorin, "The microarchitecture of a real-time robot motion planning accelerator," in *Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[56] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. The MIT Press, 2005.

[57] B. Clipp, J. Lim, J.-M. Frahm, and M. Pollefeys, "Parallel, real-time visual slam," in *IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[58] M. D. Croce, T. Pire, and F. Bergero, "Ds-ptam: Distributed stereo parallel tracking and mapping slam system," *Journal of Intelligent and Robotic Systems*, vol. 95, no. 2, pp. 365–377, 2019.

[59] G. Klein and D. W. Murray, "Parallel tracking and mapping for small ar workspaces," in *International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.

[60] T. Pire, T. Fischer, G. I. Castro, P. de Cristóforis, J. Civera, and J. Jacobo-Berlles, "S-ptam: Stereo parallel tracking and mapping," in *Robotics and Autonomous Systems (RSS)*, 2017.

[61] B. D. Gouveia, D. Portugal, D. C. Silva, and L. Marquesu, "Computation sharing in distributed robotic systems: A case study on slam," *IEEE Trans. Automation Science and Engineering (T-ASE)*, vol. 12, no. 2, pp. 410–422, 2015.

[62] S. Dey and A. Mukherjee, "Robotic slam: a review from fog computing and mobile edge computing perspective," in *MobiQuitous (Adjunct Proceedings)*, 2016.

[63] Y. Dai, D. Xu, and Y. Zhan, "Towards optimal access point selection with available bandwidth estimation," in *iThings/GreenCom/CPSCom/SmartData*, 2017.

[64] H. Wu, Q. Wang, and K. Wolter, "Methods of cloud-path selection for offloading in mobile cloud computing systems," in *International Conference on Cloud Computing (CloudCom)*, 2012.

[65] J. Li, L. Huang, Y. Zhou, S. qiang He, and Z. Ming, "An approximation algorithm for ap association under user migration cost constraint," in *IEEE Conference on Computer Communications (INFOCOM)*, 2016.

[66] A. Croitoru, D. Niculescu, and C. Raiciu, "Towards wifi mobility without fast handover," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2015.

[67] A. J. Nicholson, Y. Chawathe, M. Y. Chen, B. D. Noble, and D. Wetherall, "Improved access point selection," in *International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2006.

[68] A. Rahman, J. Jin, A. L. Cricenti, A. Rahman, and A. Kulkarni, "Communication-aware cloud robotic task offloading with on-demand mobility for smart factory maintenance," *IEEE Trans. Ind. Informatics*, vol. 15, no. 5, pp. 2500–2511, 2019.

[69] P. Pandey, D. Pompili, and J. Yi, "Dynamic collaboration between networked robots and clouds in resource-constrained environments," *IEEE Trans. Automation Science and Engineering (T-ASE)*, vol. 12, no. 2, pp. 471–480, 2015.