2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

A Comprehensive Modeling and Scheduling Approach for Allocating Distributed Multi-Robot Software to the Edge/Cloud

Yongzhou Zhang*,1,3, Florian Mirus*,2, Frederik Pasch*,2, Kay-Ulrich Scholl[2], Christian Wurll[1], Björn Hein[1,3]

***Abstract*— Offloading software modules to the edge/cloud can enhance a robot's capabilities by leveraging massive computing power. However, determining which software module should be offloaded and scheduled to which robot/edge/cloud node is a challenging task, particularly for robot fleets with diverse tasks. In this paper, we tackle the software scheduling problem and introduce a taxonomy to categorize software modules and classify their applicability and requirements for offloading. Additionally, by using prior measurements, we model the compute cluster and formalize software scheduling as a multi-objective optimization problem which we tackle with a genetic algorithm. To evaluate our approach with a challenging setup, we build a mobile manipulation task using open-source frameworks and libraries in the Robot Operating System (ROS2) community in simulation as well as a mildly simplified real-world variant. Our evaluation shows significant improvements compared to the built-in scheduler of Kubernetes (K8s) regarding robotic specific metrics such as the rate of missed cycle time in both simulated and real-world experiments.**

I. INTRODUCTION

Recent advances in robotics and AI research lead to the development of robots that are able to perform diverse tasks autonomously in various environments. However, with growing task diversity, the complexity and size of the corresponding robotic software stack increases as well. For instance, for a mobile manipulator to accomplish a pick-up and delivery task in an industrial environment, a plethora of software modules solving sub-tasks such as scene perception, arm motion control or navigation is required [1]. All those modules require massive computational resource that cannot provided by a single onboard computer due to limited space and battery capacity. Therefore, offloading to the edge or cloud could enhance the robot's capabilities for many tasks [2]. However, two main questions have to be addressed: A) which parts should be offloaded from the robot and B) to which edge/cloud nodes. Different resource limitations of compute nodes in a potentially heterogeneous cluster (i.e., consisting of robots, edge and potentially cloud nodes of different specs) lead to variations in the software modules' performance in addition to uncertainties regarding bandwidth and latency in the communication network. Therefore, a systematic modeling and scheduling approach is necessary to make assignment decisions.

Compared to scheduling in cloud native computing [3], robotic software has specific requirements, such as real-time constraints, safety considerations, and the module activation periods that depend on the task allocation. Therefore, we first introduce a taxonomy for robotic software that takes into account the activation pattern, safety relevance, and the real-time requirements and categorizes all software modules regarding their suitability for offloading to the edge or cloud. Based on this taxonomy and prior measurements to create a model of available compute hardware in the cluster, we mathematically formulate robotic software scheduling as a multi-objective optimization problem. In this paper, we use the Nondominated Sorting Genetic Algorithm II (NSGA-II) to optimize with respect to four objective functions which define the optimization goals such as reducing onboard resource requirements, load balancing, and guarantees regarding cycle times. A high-level overview is illustrated in Fig. 1. Note that we focus on a-priori scheduling in this paper, i.e., assigning software modules to compute nodes before actual deployment of the robots. This work can serve as a basis for more complex orchestration tasks such as coordinating multiple services or scaling and/or moving workloads dynamically during deployment, i.e., while the application is running, which are not considered here.

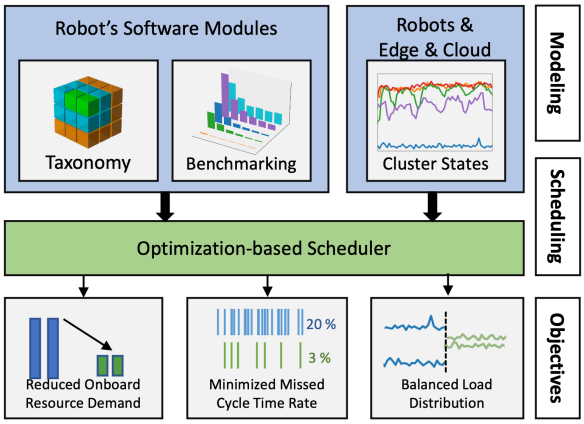


Fig. 1. High-level overview of the modeling and scheduling approach to achieve multiple objectives in robotic software scheduling.

To demonstrate the usefulness of our proposed approach, we build a mobile manipulation application using ROS2 (distribution: humble) and Gazebo [4] on top of commonly used open-source robotic software stacks, including Nav2 and MoveIt2. We containerize each software module with Docker to be fully compatible within Kubernetes in the hybrid robot/edge/cloud computing infrastructure. We conduct experiments in single-robot and multi-robot setups (scale-up is achieved through multiple parallel Gazebo instances),

*Equal contribution
[1]Karlsruhe University of Applied Sciences, 76133 Karlsruhe, Germany. yongzhou.zhang@h-ka.de
[2]Intel Labs, 76131 Karlsruhe, Germany. {florian.mirus, frederik.pasch, kay-ulrich.scholl}@intel.com
[3]Karlsruhe Institute of Technology, 76131 Karlsruhe, Germany

considering both fleet task allocation and the case of all software modules running in parallel. Using the rate of missed cycle times as metric, we compare our approach to a purely onboard deployment in the single-robot setup and the Kubernetes default scheduling scheme in the multi-robot setup. In addition to the simulation-based experiments, we test our approach with a slightly stripped-down variant of the application using three physical Turtlebots navigating an office environment with a heterogeneous cluster of compute nodes. Using our approach, we achieve significant improvements regarding software runtime quality in all experiments.

In summary, the main contributions of this paper are a taxonomy to categorize robotic software modules for scheduling, a mathematical formulation of robotic software scheduling as a multi-objective optimization problem built on top of the taxonomy and a pre-measured model of the compute cluster. Finally, we demonstrate the effectiveness of our approach in both simulated and real-world experiments with multiple robots and different compute clusters.

II. RELATED WORK

Offloading robotic software to the edge/cloud offers a powerful enhancement to various robotic tasks [2], [5]. A lot of innovative concepts and methods were proposed for tasks ranging from perception [6], grasping pose estimation [7], navigation [8] to motion planning [9], [10]. Many "as-a-service" models were proposed to provide robotic skills as a cloud service and improve accessibility, such as Dex-Net as a Service (DNaaS) [11] or Robot Inference and Learning as a Service (RILaaS) [12]. Those approaches show that the performance of one specific task can be significantly improved by using cloud computing, even in the presence of a high network latency.

Complementing these advancements from a system perspective, several architectural solutions were proposed to simplify the access to cloud computing for roboticists. For instance, FogROS2 automatically provisions Virtual Machines (VMs) in the cloud to deploy ROS2 nodes [13]. Rapyuta establishes web-socket-based communication to access the robot knowledge base [14]. KubeROS uses K8s to facilitate the deployment of ROS2-based applications across robot, edge, and cloud [15]. These concepts provided an easy way for offloading. However, the question of which module should be offloaded and to which nodes remains a challenge in the robotics domain.

In cloud native computing, scheduling is a key process that aims to efficiently place the software modules or containers into appropriate compute nodes to meet requirements such as high service availability and reliability, workload balancing, low cluster cost [3]. In the widely used orchestration platform Kubernetes (K8s), the built-in scheduler makes assignment decisions in two main steps: selecting nodes that meet the requirements and then ranking them based on configurations such as `LeastAllocated` and `BalancedResourceAllocation` [16], [17].

Based on the K8s built-in scheduler, a new Kubernetes Container Scheduling Strategy (KCSS) was proposed to improve the performance by considering hybrid criteria [18]. In [19], the authors proposed a genetic algorithm-based approach to optimize container allocation and elasticity management related with respect the users' demands. To minimize energy utilization, a Kubernetes-Based Energy and Interference Driven Scheduler (KEIDS) is proposed for industrial IoT in an edge-cloud ecosystem [20]. A recent study proposed to extend the K8s built-in scheduler with RT-container (realtime) capabilities through the RT plugins that provide additional data and introduced a platform called RT-Kube, which aims to reduce the missed real-time deadline rate for time-critical tasks and evaluated with a single mobile manipulator [21]. In comparison, our work focuses on scheduling software modules of an entire robot fleet and minimizing the overall cycle time that also considers the network communication latency.

In general, schedulers from cloud computing are appropriate for numerous use cases in cloud applications, but not optimized for robotic software. When deploying numerous software modules of an entire robot fleet, an uneven deployment can cause node overload because the scheduler lacks prior-knowledge about the software modules and their requirements from a robotic application perspective. In contrast, our modeling and scheduling approach takes both factors into account to create an optimized assignment.

Additionally, there are two relevant research areas: resource allocation in cloud robotics, which mainly focuses on task level offloading and scheduling to improve service performance and overall workflow [22]–[24], and multi-robot task allocation, which focuses on task distribution among multiple robots to achieve the overall goals of the system [25]–[27]. In this paper, we focus on software-level scheduling, which is independent of task-level allocation. Our approach aims to ensure that all software modules can be invoked at any time without risking performance deterioration. Finally, our scheduler is platform-agnostic and can be used with various orchestration platforms such as Docker Swarm or K8s.

III. OPTIMIZED SCHEDULING APPROACH

In this section, we describe our approach for optimized scheduling of robotics software modules to nodes in a hybrid compute cluster. In Sec. III-A, we propose a taxonomy to assess the suitability of modules in a given robotic software stack for offloading. In Sec. III-B, III-C and III-D, we give a mathematical formulation of the scheduling problem, a-priori assumptions and the formulation of optimization objectives respectively. Finally, Sec. III-E describes the implementation of our optimization approach.

A. Taxonomy

For robotic software modules, the requirements and constraints that need to be taken into account when assessing their suitability for offloading the edge or cloud are different compared to cloud native computing. Inspired by research in multi-robot task allocation [28], [29], we therefore propose a taxonomy (see Fig. 2) that provides a comprehensive

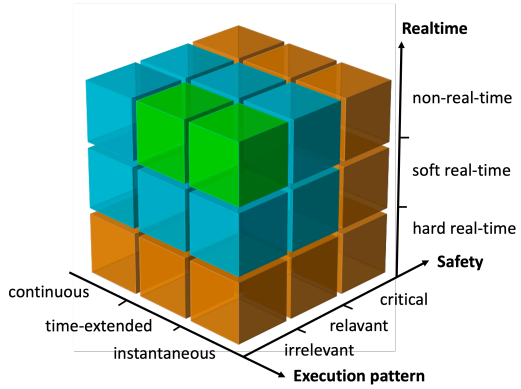


Fig. 2. **Visual representation of the robotics software taxonomy**: All modules (colored carrot orange) that either require hard real-time or are safety-critical must be deployed in the robots. Modules in green can be offloaded to reduce the onboard compute load. Modules in between (blue) need to be scheduled based on the requirements and robot cluster state.

way to categorize and abstract the entire software for a robotic application to the mathematical models designed for multi-objective optimization of the scheduling problem. The taxonomy considers three important metrics: activation patterns, security importance, and real-time requirements.

1) Execution pattern: We consider three levels of execution patterns: *Continuous tasks* run continuously while the robot is active and have a desired control loop time. *Time-extended periodic tasks* get activated at regular, predictable intervals and run for an extended period with a defined start and end point. *Instantaneous tasks (event-based)* are executed in response to specific events or conditions. They are event-driven, non-determined, and less predictable.

2) Real-time requirements: Again, we consider three levels of real-time requirements: For modules with *hard real-time* requirements, timing is critical, and missing a deadline can lead to catastrophic results or system failure. Such modules have to be deployed in the robot and the scheduler needs to reserve enough resources for them to work properly at all times. For modules with *soft real-time* requirements, timing is important but not critical. Deadlines may be missed occasionally without causing system failure, but might lead to performance degradation. Finally, modules with *Non-real-time* requirements have no strict timing constraints such that their timing is less critical and delays usually do not result in performance degradation.

3) Safety relevance: Safety-critical modules directly impact the safety of the robot or its environment. Failures of such modules can lead to severe consequences, hence there must be sufficient resources reserved for that module even in the worst-case scenario. *Safety-relevant* modules are important for ensuring safe robot operation, but failures may not have immediate catastrophic consequences. *Safety-irrelevant* modules are not critical for the safety of the robot or its environment, but they may influence functionality or efficiency of the overall application.

Fig. 2 visualizes the three metric axes of our taxonomy. In general, software modules with hard real-time and/or are safety-critical requirements (carrot orange in Fig. 2) must be deployed onboard and their resource consumption must be considered when scheduling other modules. In contrast, modules without safety requirements that do not necessarily need to run continuously (green in Fig. 2) should be prioritized for offloading. All modules marked in blue can be scheduled based on their requirements and the cluster and network status.

TABLE I
VARIABLES AND DEFINITIONS

Compute cluster	
N_R	Number of robots' on-board compute nodes, $N_R \in \mathbb{N}$
N_E	Number of edge/cloud compute nodes, $N_E \in \mathbb{N}$
N	Number of all compute nodes $N = N_R + N_E$
$\mathfrak{C}$	Compute cluster $\{n_1,\ldots,n_N\} = \mathfrak{C}_R \cup \mathfrak{C}_E$
$\mathfrak{C}_R$	On-board compute cluster $\{n_1,\ldots,n_{N_R}\}$
$\mathfrak{C}_E$	Edge and cloud compute cluster $\{n_{N_R+1},\ldots,n_{N_R+N_E}\}$
n_j	The j-th compute node in $\mathfrak{C}$
Software and Resources	
$\mathfrak{R}$	Set of resources, e.g., $\mathfrak{R} = \{CPU, RAM, GPU\}$
S	Number of software modules
$\mathbf{S}$	Set of software modules in the stack $\{m_1,\ldots,m_S\}$
m_i	The i-th software module in $\mathbf{S}$
$\mathscr{L}_r$	Set of available resource limits $\{L_0^r,\ldots,L_N^r\}$ of each node n_j
Scheduling/Deployment	
$\delta(m_i,n_j)$	Binary indicator, 1 if the module m_i is scheduled to node n_j
$U(m_i,n_j,r)$	Usage of resource type $r \in \mathfrak{R}$ by module m_i on node n_j
$\mathfrak{u}_j^r$	total usage of resource type $r \in R$ on compute node n_j
L_j^r	Limit of resource type $r \in R$ on compute node n_j
$t_{\text{exe}}(m_i,n_j)$	Execution time of m_i on n_j
$t_{\text{cycle}}(m_i,n_j)$	Cycle time of m_i on n_j
$\tau(m_i,n_j)$	Cycle time (expected vs actual) factor if $\delta(m_i,n_j) = 1$

B. Problem Formulation

Let $\mathbf{S} = \{m_1,\ldots,m_S\}$ be a set of containerized software modules to be scheduled in a given robotics software stack (entire fleet). Furthermore, let $\mathfrak{C} = \{n_1,\ldots,n_N\}$ be a set of N available compute nodes (referred to as compute cluster) consisting of a set of N_R robots, or more precisely, robot on-board compute nodes $\mathfrak{C}_R = \{n_1,\ldots,n_{N_R}\}$, and a set $\mathfrak{C}_E = \{n_{N_R+1},\ldots,n_{N_R+N_E}\}$ of N_E compute nodes from the edge or cloud, i.e., $\mathfrak{C} = \mathfrak{C}_R \cup \mathfrak{C}_E$ and $N = N_R + N_E$. We consider the scheduling problem in this paper to be the optimal distribution of containerized software modules to compute nodes. That is, a deployment function $\varphi : \mathbf{S} \to \mathfrak{C}$ that assigns software module m_i to a compute node $n_j = \varphi(m_i)$. The definitions of relevant symbols are listed in Table I.

As the quantification of safety relevance requires human expertise, and the critical safety requirements or hard real-time constraints cannot be guaranteed due to the variability of network latency, these modules must run on the robot's on-board computing platform. According to our taxonomy, we only consider software modules without hard real-time requirements that are not safety-critical (green and blue modules in Fig. 2) for scheduling to the edge/cloud. Additionally, cloud and edge nodes are considered the same in our model, in contrast to robot/on-board nodes. We consider the difference between cloud and edge in their resource scalability as well as the network latency, bandwidth, and reliability, which are considered by our model. For brevity, we will use the term edge in the remainder of this paper. As optimal deployment in a robotics context is different from

cloud applications, we formulate optimality criteria under the following robotics-specific assumptions.

C. Assumptions

1) Limited on-board resources: As we particularly target mobile robots with our approach, this assumption is key. On the one hand, most mobile robots are equipped with lean compute devices, mostly in a small form factor to meet the space constraints. On the other hand, deploying software modules to the on-board compute is directly related to the power consumption and therefore its battery life.

2) Balanced edge-/cloud-resource utilization: Available compute resources in the edge or cloud are much higher compared to on-board compute, but a balanced workload distribution is desired. An edge cluster typically consists of multiple server nodes with powerful hardware, which should be evenly used to avoid fully loading individual nodes while others are running in idle mode.

3) Communication network variability: There is a variable latency and available bandwidth between the (mobile) robot nodes and the compute nodes in the edge, particularly during task execution. That is, exchanging data between compute nodes inside the robot or the edge requires less time and offers higher bandwidth due to a wired network connection and is thus subject to a lower network latency compared to data sent via a wireless network connection from robot to edge/cloud nodes and vice versa.

D. Objective Formulation

In this section, we formulate scheduling objectives deduced from our aforementioned assumptions. The function $\delta(m_i,n_j)$ is a binary indicator if software modules m_i is deployed on compute node n_j, i.e., $\delta(m_i,n_j)=1$ if $n_j=\varphi(m_i)$ and $\delta(m_i,n_j)=0$ otherwise. With a set of resources $\mathfrak{R}$ (such as CPU, GPU, or RAM), the function $U(m_i,n_j,r)$ is defined as the usage of resource type r of software module m_i on compute n_j. Using these definitions, the following function F_1 formalizes the total resource consumption of the software modules deployed on the robot onboard compute nodes to be minimized according to assumption III-C.1:

$$F_1 = \sum_{r\in\mathfrak{R}} \sum_{j=1}^{N_R} \underbrace{\sum_{i=1}^{S} \delta(m_i,n_j)\cdot U(m_i,n_j,r)}_{=:\mathfrak{u}_j^r}, \tag{1}$$

with $\mathfrak{u}_j^r := \sum_{i=1}^{S} \delta(m_i,n_j)\cdot U(m_i,n_j,r)$ formalizing the total usage of resource type r on compute node n_j. However, we not only want the deployment solution to minimize the total on-board resource consumption but also to respect certain resource limitations. Therefore, we define a set of resource limitations $\mathscr{L}_r=\{L_0^r,\ldots,L_N^r\}$ for each compute node n_j per resource type $r\in\mathfrak{R}$. That is, the total resource consumption of all software modules deployed to one compute node n_j must be lower than the node resource limit: $\mathfrak{u}_j^r < L_j^r$. We formalize this with the following function:

$$F_2 = \mathfrak{u}_j^r - \sum_{r\in\mathfrak{R}} \sum_{j=1}^{N} L_j^r \tag{2}$$

In contrast to considering the resource limit as a hard constraint, we model it as an objective to maintain a certain resource reserve. In case a deployment solution φ violates the resource limitations, i.e., if $\mathfrak{u}_j^r > L_j^r$ for any compute node n_j and any resource type $r\in\mathfrak{R}$, the solution φ is considered invalid.

To realize assumption III-C.2 to evenly distribute the software modules on the edge nodes, we formalize and quantify the "evenness" of the resource usage in the edge cluster nodes. Therefore, we follow the notation in [19] and use the standard deviation σ of the resource utilization rate among edge/cloud nodes:

$$F_3 = \sum_{r\in\mathfrak{R}} \sigma\left(\left[\mathfrak{u}_j^r/L_j^r\right]_{j=N_R+1}^{N}\right) \tag{3}$$

All of the objectives considered so far are related to the amount of resource utilization across (parts of) the compute cluster caused by the scheduling solution φ. To consider assumption III-C.3, let $t_{\text{expected}}(m_i,n_j)$ and $t_{\text{cycle}}(m_i,n_j)$ be the expected and actual mean cycle time of software module m_i running on n_j for a given scheduling solution φ. We consider the cycle time of the module m_i when scheduled to n_j the sum of the execution time $t_{\text{exe}}(m_i,n_j)$, the incoming latency $t_{\text{in}}(m_i,n_j)$, and the outgoing latency $t_{\text{out}}(m_i,n_j)$:

$$t_{\text{cycle}}(m_i,n_j) = t_{\text{in}}(m_i,n_j) + t_{\text{exe}}(m_i,n_j) + t_{\text{out}}(m_i,n_j) \tag{4}$$

The communication latency implicitly depends on the amount of data software module m_i sends and receives as well as on the deployment function φ, i.e., the latency varies depending on where a software module is deployed and if the data it sends and/or receives needs to be transmitted over wireless communication. If the entire software stack is in a healthy state, the actual average cycle time of all software modules should be less than the expected, i.e., all the cycle time factors

$$\tau(m_i,n_j) := \frac{t_{\text{cycle}}(m_i,n_j)}{t_{\text{expected}}(m_i,n_j)} \tag{5}$$

should less than 1. We set $\tau_{>1}(m_i,n_j)=\tau(m_i,n_j)$ if $\tau(m_i,n_j)>1$ and otherwise $\tau_{>1}(m_i,n_j)=0$ to denote missed cycle time factors. Hence, we aim to minimize the function

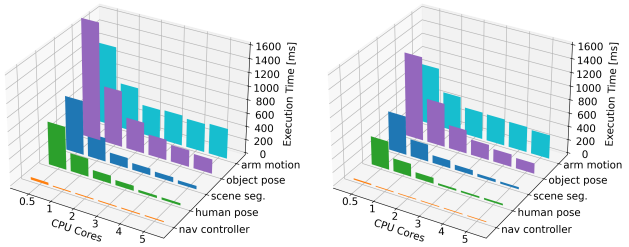
$$F_4 = \sum_{i=1}^{S} \sum_{j=1}^{N} \delta(m_i,n_j)\cdot\tau_{>1}(m_i,n_j), \tag{6}$$

which formalizes the amount of missed cycle times of all software modules across the compute cluster.

The choice of objective functions presented in this section is not and can not be comprehensive. There is a plethora of possible alternative objective functions. For example, by assigning an economic cost value to each compute node, the cost of the entire system can be optimized.

E. Optimization Implementation

The scheduling optimization problem formulated above involves a large search space and the complexity grows with the cluster size and the number of software modules. As genetic algorithms have been widely used in cloud



(a) Azure Dsv3-Series (b) Azure Fsv2-Series

Fig. 3. **Performance test on different VM series**: This data is used as prior knowledge in the scheduler. We run the performance test automatically in a K8s cluster for a set of given software modules.

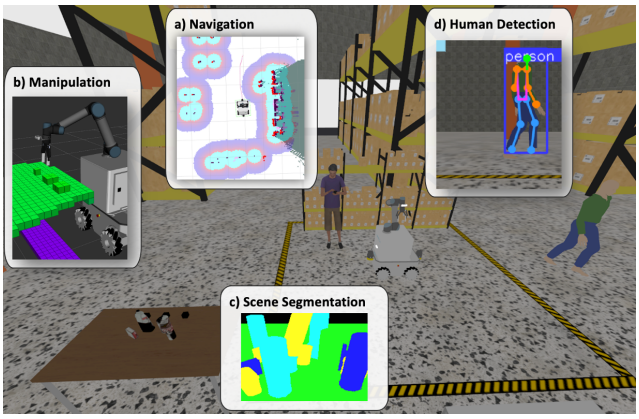


Fig. 4. **Mobile manipulation**: Multiple mobile manipulators work in a warehouse to pick up and deliver goods. It consists of following subtasks: a) autonomous navigation; b) manipulation to pickup object; c) scene segmentation to localize target object; d) human pose estimation.

computing and multi-robot task allocation research to tackle such optimization problems [19], [24], [30], we propose to use a genetic algorithm to find an optimal or near-optimal solution. Similar to [19], we employ the Nondominated Sorting Genetic Algorithm II [31] using the available implementation in the Python Genetic Algorithm (PyGAD) library [32] to find an optimal deployment function $\varphi : \mathbf{S} \rightarrow \mathfrak{C}$ for a given software stack $\mathbf{S}$ and compute cluster $\mathfrak{C}$.

In this paper, we focus on a-priori scheduling, i.e., finding an optimal scheduling solution before the actual deployment of the software stack and implement the four objective functions $F_1, \ldots, F_4$ described in Sec. III-B. Next to the explicit implementation of objectives as fitness functions and the implicit implementation of constraints penalizing an invalid solution with a negative reward as described in Sec. III-B, we also implement robotic specific constraints through assigning a specific gene spaces to each software module. For each software module m_i, we can define a set of compute nodes $C_i \subseteq \mathfrak{C}$, which NSGA-II is allowed to deploy m_i on. For instance, it does not make sense to deploy a software module m_k belonging to robot n_1 on any other robot's on-board compute node, hence we set the gene space to $C_k := \{n_1, n_{N_R+1}, \ldots, n_N\} = \mathfrak{C} \setminus \{n_2, \ldots, n_{N_R}\}$. Another option for modules that cannot be processed at all on the robots' on-board compute nodes due to their limitations could be to restrict the gene space to (a subset of) $\mathfrak{C}_E$, which reduces the search space. We run the optimization algorithm with random mutations with a mutation probability of 30 %, "single point" crossover, 100 solutions per population for a maximum 100 generations as these parameters delivered the best optimization results in our experiments (see [32] for details on parametrization of the PyGAD library).

Prior Data and Cluster Model to Calculate the Fitness: For our NSGA-II optimization algorithm to produce plausible deployment functions $\varphi : \mathbf{S} \rightarrow \mathfrak{C}$ as results, we need to create a model of the performance of our software stack S and the compute cluster $\mathfrak{C}$. Therefore, we use default K8s and deploy each software module $m_i \in \mathbf{S}$ on one cluster node of each hardware type and measure its execution time over a certain time window. To eliminate the influence of other software modules running in parallel on the same compute node n, we assign varying CPU-limits from the set $\{0.25, 0.5, 1, 2, \ldots, L_n^{CPU}\}$ during these measurements,

TABLE II

SOFTWARE MODULES IN EXAMPLE APPLICATION

Module	Max. Cycle Time	Avg. Execution Rate*	Framework / Algorithm
nav. motion planning	1 s	70%	NavFN
nav. localization	33 ms	70%	SLAM-toolbox
nav. controller	33 ms	70%	MPPI
nav. periphery	1 s	100%	Nav. coordinator
human pose estimation **	100 ms	100%	Yolov8n-pose
arm motion generation	600 ms	30%	MoveIt 2
scene segmentation **	150 ms	14%	Yolov8n-seg
object pose estimation	400 ms	8%	ICP

* Estimated average execution time based on fleet task allocation
** Both AI modules are accelerated with OpenVino on CPU

where L_n^{CPU} denotes the maximal number of allocatable CPU-cores on compute node n, i.e., its resource limit. The result is a look-up table containing the (mean) execution time $t_{\text{exe}}(m_i, n_j)$ and variance for each software module m_i in the software stack S depending on the assigned resource usage $U(m_i, n_j, CPU)$ on each node $n_j \in \mathfrak{C}$. This data serves as prior model for our NSGA-II optimization algorithm to calculate the individuals' fitness (see Fig. 3). To get an estimation of the modules' cycle time including the communication latency, we measured the time necessary to transmit messages of different size (e.g., camera image, LIDAR scan, odometry, and robot state messages) between different compute nodes depending on the type of communication (WiFi, LAN, Ethernet to cloud.).

IV. EXPERIMENTS

To demonstrate our scheduling approach, we conducted experiments with both, simulation and physical robots. As an example application, we implemented a mobile manipulation application in the Gazebo simulator [4] as shown in Fig. 4. We build our implementation upon open source frameworks and libraries that are widely used in robotics community.

The overall mobile manipulation task consists of multiple sub-tasks, namely autonomous navigation, collision-free motion planning and trajectory generation for the robot

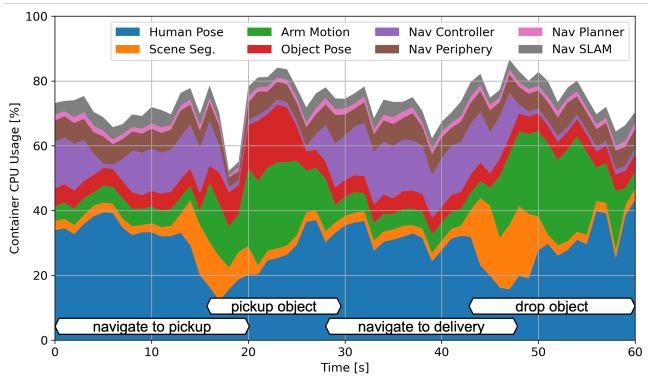
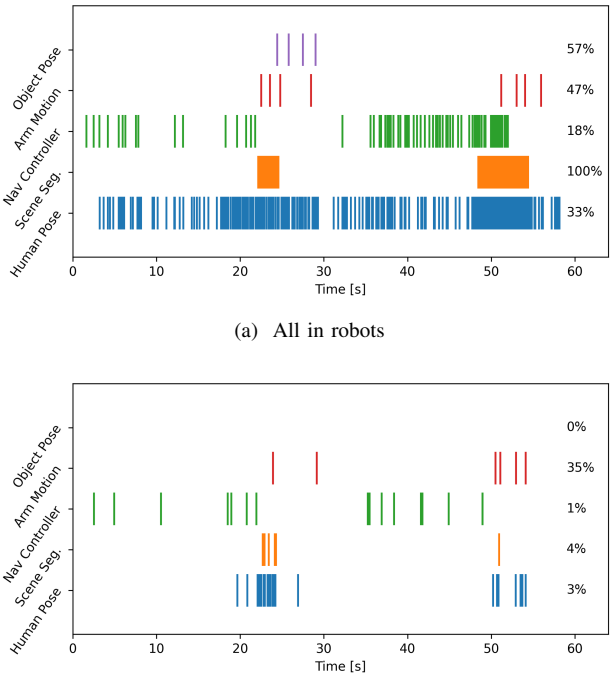


Fig. 5. **Container CPU usage in a task period** without offloading, tested on Dsv3-series with 6 available cores as onboard computer. During times, where activation of sub-tasks overlap, multiple modules running in parallel lead to performance degradation of all modules.



(a) All in robots

(b) With Offloading to Edge

Fig. 6. **Comparison of missed cycle time distribution**: By offloading multiple modules to the edge (one node with 16 core), the missed cycle time rate is dramatically reduced. The safety-relevant module nav2 controller gets more onboard resources. When offloading to an edge node with 32 cores, the missing rates of all modules are close to 0.

arm, scene segmentation, object pose estimation for grasping, human pose estimation, which are integrated into eight individual software containers (see Table II). For navigation, we encapsulate the Nav2 [33] software stack in four individual containers containing the NavFn global path planner, the Model Predictive Path Integral (MPPI) controller, Simultaneous Localization and Mapping (SLAM) for localization and periphery modules such as the lifecycle manager or behaviour tree navigator. The arm motion module utilizes the Transition-based Rapidly-exploring Random Trees (TRRT) planner [34] to generate a collision-free path and the time optimal trajectory generation algorithm [35] in MoveIt2 [36] to obtain the trajectory for the low-level arm controller. To estimate the object pose, we use the Iterative Closest Point (ICP) implementation in Open3D [37]. For perception, we employ two pre-trained YOLO v8 models [38] to segment the scene and estimate human pose. We chose the CPU usage of the individual containers as well as the rate at which each module misses its desired cycle time as metrics to evaluate the deployment results. If a software module exceeds the desired cycle time by any value, we count a cycle time miss, i.e., we do not distinguish between near-misses (e.g., some milliseconds) and severe misses (e.g., hundreds of milliseconds or more).

We obtain the prior data, i.e., the input data for our scheduling optimization approach, to model the resource usage, execution times and latency by automatically running tests on the available compute nodes in each cluster as described in Sec. III-E. Fig. 3 shows the execution time on two different virtual machine groups (Dvs3-series and Fvs2-series) from the Azure Cloud under different available CPU resources.

A. Single Robot: Offloading to Edge vs. Onboard Only

We consider a mobile manipulator application with a single robot in simulation as a baseline use-case. As a first experiment, we compare the deployment of all software modules on one compute node with 6 available CPU-cores (emulating an on-board node; note that 2 other cores are reserved for monitoring containers and other system processes) with one deployment, which offloads all software modules except Nav2 components to one edge node with 16 CPU-cores. Fig. 5 shows that the total CPU usage of the on-board deployment whereas Fig. 6 depicts the distribution of missed cycle times of the on-board and offloaded deployment over the run-time of the application. We observe that the full on-board deployment fully loads the onboard compute node to capacity (Fig. 5), which leads to multiple software modules missing their desired cycle times at much higher frequency (Fig. 6a) compared to the offloaded deployment (Fig. 6b).

B. Multi-Robot: GA-based vs. K8s Default Scheme

To evaluate our optimization-based scheduling approach in a more challenging use-case, we conduct a multi-robot experiment with 5 robots and 6 edge compute nodes. To simulate fleet behavior, we run multiple simulation instances in parallel in the cloud and measured the total CPU utilization of the nodes using node-exporter and Prometheus [39] for evaluation. Fig. 7 compares the CPU utilization of both deployments (default K8s and our approach) for all edge compute nodes over the total run-time of one task period. We already observe that our optimized deployment distributes the workloads more evenly across the edge nodes compared to the default deployment, which is expected according to objective function F_3 (Eq. 3), which aims to avoid overloading compute nodes. To account for variations and uncertainty in the measurements of a single run, we conduct

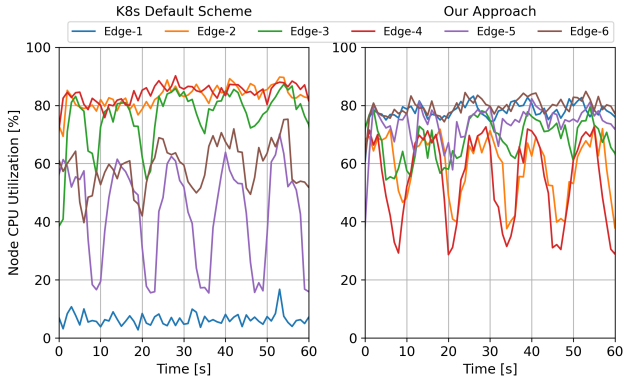


Fig. 7. **Edge node CPU utilization**: Compared to one deployment using K8s default scheduling scheme. The workload is more evenly distributed across all edge nodes using our approach.

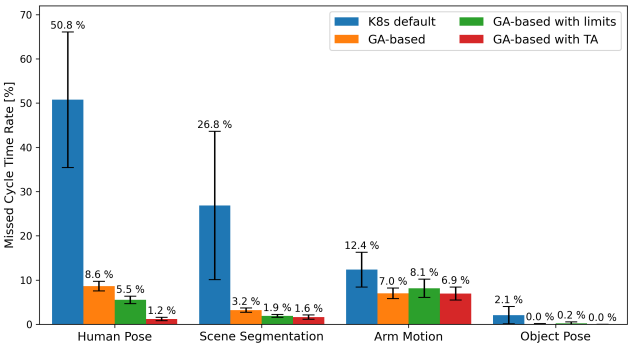
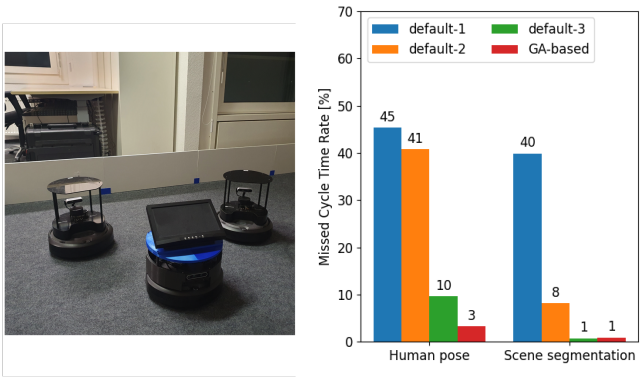


Fig. 8. **Comparison of missed cycle time rate of modules in the entire fleet**: K8s default scheme vs. GA-based vs. GA-based with container resource allocation limits vs. GA-based considering the fleet task allocation.

this experiment 10 times for each deployment strategy and evaluate the results for statistical significance.

Fig. 8 depicts the rate at which selected modules miss their desired cycle time for the different deployment approaches: *(1) K8s default* denotes the default K8s deployment scheme, whereas *(2) GA-based* and its variants denote our approach. *(3) GA-based with limits* additionally uses minimum container resource requests and maximum limits to avoid overallocation whereas *(4) GA-based with TA* considers the fleet task allocation, i.e., software modules are activated when needed for a specific task. The first three approaches consider task allocation to be unpredictable and activate all software modules in parallel. The resource requests and limits used in the third approach are not compatible with K8s default scheme, because the values vary for different compute node types in a heterogeneous cluster. Our approach consistently reduces the missed cycle time rate for both, scene segmentation and human pose estimation modules on average by at least by 63 % without compromising the arm motion planning and grasp pose modules. Furthermore, the optimized deployment is more consistent and shows less variance in repeated experiments. Finally, combining application-specific task allocation with our scheduling approach leads to the lowest missed cycle time rate for all software modules.



(a) Setup (b) Result

Fig. 9. **Experiment with 3 Turtlebots**: Robots navigate autonomously in the office and the front camera is used to detect the humans and perceive the scene. Comparison of 3 random deployments using the K8s default scheme (notated as default-1/2/3) and our approach as validation of experiments conducted in simulation.

C. Real Robots with heterogeneous edge cluster

For the real-world experiments, we used a stripped-down variant of the mobile manipulation task using three physical Turtlebots (see Fig. 9a) each equipped with a Raspberry Pi 4 and a smaller compute cluster consisting of three edge compute nodes, namely two equipped with 9th generation i9 CPU and one equipped with a 13th generation i9 CPU. This results in different resource usage limits in the edge part of the cluster to be considered by the optimization algorithm. The robots are connected to the edge cluster via a single access point (WiFi-6). The software stack is similar to the simulated task described above (excluding software modules for manipulation). Fig. 9b compares the rate of missed cycle times of the human pose estimation and scene segmentation software modules for deployments using our approach with the default K8s scheduler (over 3 repetitions). Similar to the experiments in simulation, our optimized deployment is more consistent, shows less variance and reduces the missed cycle time rate on average by at least by 80 %.

V. CONCLUSION

In this paper, we proposed a modeling and optimization approach employing the multi-objective NSGA-II for scheduling robotic workloads in a hybrid robot/edge/cloud computing cluster. As a basis, we firstly introduced a taxonomy which allows deductions on the question which software modules are suitable candidates for offloading to remote compute nodes. Additionally, we used prior measurements for application-specific parameters, here resource usage and cycle time for each software module, as a model of the cluster at hand as prior data for our optimization approach. Finally, we demonstrated the usefulness and effectiveness of our approach on a challenging mobile manipulation application use case with multiple robots in simulation and a slightly simplified variant with 3 physical robots. Our approach delivered promising results, significantly and consistently reducing the rate at which robotic software modules

missed their desired cycle times compared to the K8s default deployment scheme.

Although our approach showed promising results, there are several options for future work. In this paper, we have used a simplification related to the behavior of the communication network, which could be further improved in favor of a more accurate model. Furthermore, we aim to investigate additional optimization objectives. For example, one could consider the economic cost of each node and minimize the total cost to find its most cost-effective subset of the cluster. Finally, we will explore learning-based alternatives to the genetic algorithm NSGA-II used in this paper to find an optimal deployment.

ACKNOWLEDGMENT

Part of this research is being conducted as part of the KI5GRob project funded by the German Federal Ministry of Education and Research (BMBF) under project number 13FH579KX9.

REFERENCES

[1] A. Dömel, S. Kriegel, M. Kaßecker, M. Brucker, T. Bodenmüller, and M. Suppa, "Toward fully autonomous mobile manipulation for industrial environments," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, p. 1729881417718588, 2017.

[2] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.

[3] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[4] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[5] G. Hu, W. P. Tay, and Y. Wen, "Cloud robotics: architecture, challenges and applications," *IEEE network*, vol. 26, no. 3, pp. 21–28, 2012.

[6] W. J. Beksi, J. Spruth, and N. Papanikolopoulos, "Core: A cloud-based object recognition engine for robotics," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4512–4517.

[7] B. Kehoe, D. Warrier, S. Patil, and K. Goldberg, "Cloud-based grasp analysis and planning for toleranced parts using parallelized monte carlo sampling," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 455–470, 2014.

[8] F. Mirus, F. Pasch, and K.-U. Scholl, "Towards fault-tolerant deployment of mobile robot navigation in the edge: an experimental study," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2024, pp. 6791–6797.

[9] K. Bekris, R. Shome, A. Krontiris, and A. Dobson, "Cloud automation: Precomputing roadmaps for flexible manipulation," *IEEE Robotics & Automation Magazine*, vol. 22, no. 2, pp. 41–50, 2015.

[10] J. Ichnowski, W. Lee, V. Murta, S. Paradis, R. Alterovitz, J. E. Gonzalez, I. Stoica, and K. Goldberg, "Fog robotics algorithms for distributed motion planning using lambda serverless computing," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4232–4238.

[11] P. Li, B. DeRose, J. Mahler, J. A. Ojea, A. K. Tanwani, and K. Goldberg, "Dex-net as a service (dnaas): A cloud-based robust robot grasp planning system," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, 2018, pp. 1420–1427.

[12] A. K. Tanwani, R. Anand, J. E. Gonzalez, and K. Goldberg, "Rilaas: Robot inference and learning as a service," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4423–4430, 2020.

[13] J. Ichnowski, K. Chen, K. Dharmarajan, S. Adebola, M. Danielczuk, V. Mayoral-Vilches, N. Jha, H. Zhan, E. Llontop, D. Xu, C. Buscaron, J. Kubiatowicz, I. Stoica, J. Gonzalez, and K. Goldberg, "Fogros2: An adaptive platform for cloud and fog robotics using ros 2," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 5493–5500.

[14] D. Hunziker, M. Gajamohan, M. Waibel, and R. D'Andrea, "Rapyuta: The roboearth cloud engine," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 438–444.

[15] Y. Zhang, C. Wurll, and B. Hein, "Kuberos: A unified platform for automated and scalable deployment of ros2-based multi-robot applications," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2023, pp. 9097–9103.

[16] The Kubernetes Authors, "Kubernetes documentation," https://kubernetes.io/docs/, Accessed: 2024-03-12.

[17] M. Luksa, *Kubernetes in action*. Simon and Schuster, 2017.

[18] T. Menouer, "Kcss: Kubernetes container scheduling strategy," *The Journal of Supercomputing*, vol. 77, no. 5, pp. 4267–4293, 2021.

[19] C. Guerrero, I. Lera, and C. Juiz, "Genetic algorithm for multi-objective optimization of container allocation in cloud architecture," *Journal of Grid Computing*, vol. 16, pp. 113–135, 2018.

[20] K. Kaur, S. Garg, G. Kaddoum, S. H. Ahmed, and M. Atiquzzaman, "Keids: Kubernetes-based energy and interference driven scheduler for industrial iot in edge-cloud ecosystem," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4228–4237, 2019.

[21] F. Lumpp, F. Fummi, H. D. Patel, and N. Bombieri, "Enabling kubernetes orchestration of mixed-criticality software for autonomous mobile robots," *IEEE Transactions on Robotics*, 2023.

[22] M. Afrin, J. Jin, A. Rahman, A. Rahman, J. Wan, and E. Hossain, "Resource allocation and service provisioning in multi-agent cloud robotics: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 842–870, 2021.

[23] L. Wang, M. Liu, and M. Q.-H. Meng, "A hierarchical auction-based mechanism for real-time resource allocation in cloud robotic systems," *IEEE transactions on cybernetics*, vol. 47, no. 2, pp. 473–484, 2016.

[24] W. Li, C. Zhu, L. T. Yang, L. Shu, E. C.-H. Ngai, and Y. Ma, "Subtask scheduling for distributed robots in cloud manufacturing," *IEEE Systems Journal*, vol. 11, no. 2, pp. 941–950, 2015.

[25] C. Wei, Z. Ji, and B. Cai, "Particle swarm optimization for cooperative multi-robot task allocation: a multi-objective approach," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2530–2537, 2020.

[26] S. Park, Y. D. Zhong, and N. E. Leonard, "Multi-robot task allocation games in dynamically changing environments," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 8678–8684.

[27] S. Zhang and F. Pecora, "Online sequential task assignment with execution uncertainties for multiple robot manipulators," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 6993–7000, 2021.

[28] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International journal of robotics research*, vol. 23, no. 9, pp. 939–954, 2004.

[29] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[30] Z. Ding, S. Wang, and C. Jiang, "Kubernetes-oriented microservice placement with dynamic resource allocation," *IEEE Transactions on Cloud Computing*, 2022.

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[32] A. F. Gad, "PyGAD: An Intuitive Genetic Algorithm Python Library," *Computing Research Repository (CoRR)*, vol. abs/2106.06158, 2021. [Online]. Available: https://pygad.readthedocs.io/en/latest/

[33] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[34] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based rrt to deal with complex cost spaces," in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 4120–4125.

[35] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," *Robotics: Science and Systems VIII*, p. 209, 2013.

[36] Ioan A. Sucan and Sachin Chitta, "Moveit," https://moveit.ros.org, Accessed: 2024-03-12.

[37] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[38] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLO," Jan. 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[39] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.