Efficient Hosting of Robust IoT Applications on Edge Computing Platform

1st Cosmin Avasalcai
Distributed Systems Group
TU Vienna
Vienna, Austria

2nd Bahram Zarrin
DTU Compute
DTU
Copenhagen, Denmark

3rd Paul Pop
DTU Compute
DTU
Copenhagen, Denmark

4th Schahram Dustdar
Distributed Systems Group
TU Vienna
Vienna, Austria

***Abstract*—Demanding IoT application requirements such as high dependability and low latency, cannot be satisfied by centralized cloud computing when deploying these applications. Edge computing is emerging as an alternative to deploy demanding IoT applications closer to the edge of the network. However, with edge computing, available resources are distributed among different resource-constrained devices, which cannot host large monolithic applications. We propose a new hierarchical IoT application model, suitable for the distributed nature of edge computing. Thus, a task in the application is modeled using multiple configurations of smaller tasks, each with their own functionality level and resource requirements. For deployment we use a decentralized resource technical framework that finds a satisfiable task mapping on edge devices. Its functionality is inspired by an auction house, having the objectives of (i) deploying an application such that its requirements are met and (ii) empowers edge devices to be in control of their available resources. For the latter, we propose a new decision policy to help edge devices take better decisions regarding the use of local available resource. Our solution enables efficient device resource utilization when deploying IoT applications at the edge.**

***Index Terms*—Edge Computing, Internet of Things, Decentralization, IoT application model, Resource Management**

I. Introduction

The rapid adoption of Internet of Things (IoT) devices has pushed the development of new IoT applications with more stringent requirements like low latency and increased privacy. Requirements for which the current cloud-centric state of the art may prove inefficient, due to possible network congestion and increased latency [1]. Edge computing has emerged as a promising paradigm to extend cloud capabilities closer to the end-user, enabling the migration and hosting of parts or the entire application at the edge of the network. However, since the edge computing architecture is composed of distributed resource-constrained devices, the monolithic application model is impractical; a single edge device may not accommodate an entire application. As a solution, new application models and novel resource management techniques must be developed to efficiently deploy an application to an edge computing architecture.

To enable the deployment of an IoT application to such a distributed architecture, the monolithic model must be divided into multiple smaller tasks. Researchers proposed that an IoT application can be modeled as a Directed Acyclic Graph (DAG), where vertices represent an atomic task, while the edges represent the dependencies between them [2]. An atomic task represents a function that cannot be divided into smaller tasks. With the DAG model, an application can take advantage of the available resources found at the edge of the network.

Combining the DAG application model with resource management techniques [3] enables the successful deployment of an IoT application at the edge of the network. In previous work [4], we proposed a decentralized resource auctioning technique to deploy different IoT applications on resource-constrained devices. With this approach, we manage to fully deploy an application on an edge architecture, i.e., without the help of the cloud, under the conditions that we find at least one edge device which has the required available resources for every task; a limitation that is given by the inability to further decompose an atomic task. As a result, the outcome of the deployment strategy is highly dependent on the available resources of each node and the resource requirements of each task, forcing the mapping of tasks to the cloud while leaving available resources unused at the edge.

To address the aforementioned problems, we propose a robust application model and a novel resource management technique to lower the impact of available resources on deployment success. We define an IoT application a *robust application* if its functionality can be adapted based on the edge architecture's available resources. Such an application can still be functional, with a different functionality level, if it faces a resource shortage on the edge devices. To achieve deployment flexibility, we extend the DAG application model such that it can include composite tasks, and contains information that enables the deployment strategy to choose the best functionality level of the application considering the resource availability constraints of the edge devices. Such a modeling approach is inspired by the aspect-oriented flow-based programming [5]. In this case, the developer can define multiple aspects for each composite task to ensure different functionality levels. Combining the proposed application model with novel resource management is imperative to achieve efficient deployments. As such, we extend previous work with our robust application model, updating the deployment strategy and developing a new decision policy module to enable participants to efficiently utilize their available resources.

In this paper, we propose a decentralized resource auctioning technical framework to deploy IoT applications on an edge

978-1-7281-7305-4/20/$31.00 ©2020 IEEE
DOI 10.1109/ICFEC50348.2020.00008

computing architecture. Our objective is to find a satisfiable task mapping that meets the application requirements, i.e., the end-to-end delay (e2e delay) latency, and efficiently utilizing all computational available resources found on resource-constrained devices. Our contributions are:

- An improved IoT application model that allows for better resource utilization and enables configurable application functionality based on the available resources found at each host edge node.
- An extended decentralized resource auctioning that considers the proposed robust application model, to seamlessly deploy IoT applications at runtime, assuming no design-time knowledge of network topology or devices' available resources. Furthermore, it enables the developer to push task updates in the form of new aspects, without any downtime required.
- A new decision policy that empowers a device to take local decisions regarding its available resources. Our strategy is capable of providing both feasible and optimized solutions, having the following objectives: (i) maximizing task coverage, (ii) maximizing the available device resource utilization, and (iii) maximizing the application functionality.

The remainder of the paper is structured as follows. In Section II we present the overview of our proposed solution and introduce a motivational example. Section III defines the application and architecture considered in this paper. In Section IV we describe the implementation details of our proposed technique. Section V presents the methodology and results of our evaluation regarding both deployment and bid strategy. Section VI summarizes the related work on resource allocation techniques, and finally Section VII concludes the paper and provides an outlook on future work.

II. Technical Framework Overview

Our technical framework provides decentralized resource management at runtime, focusing on seamlessly deploying latency-sensitive IoT applications on the cloud-fog-edge continuum with the objective of efficiently utilizing the available resources found at the edge. Our main objective is to aid the developer to focus only on the application development process without requiring any knowledge of the current network topology and devices' available resources. Its functionality is inspired by an auction house, from where we borrow some elements like application advertisements and bids. However, there are some differences as well, like (i) a bid, sent by a participant, contains multiple offers that may be partially or fully fulfilled and (ii) there is no incentive mechanism since we assume devices trust each other; their incentive being the ability to share resources to execute certain applications. Thus, there are two main roles that an edge node can have during application deployment, i.e., coordinator and collaborator.

An edge node becomes a coordinator when an application arrives to be deployed. The coordinator's objective is to ensure that the deployment meets the application resource requirements. Its functionality is divided into three different phases:

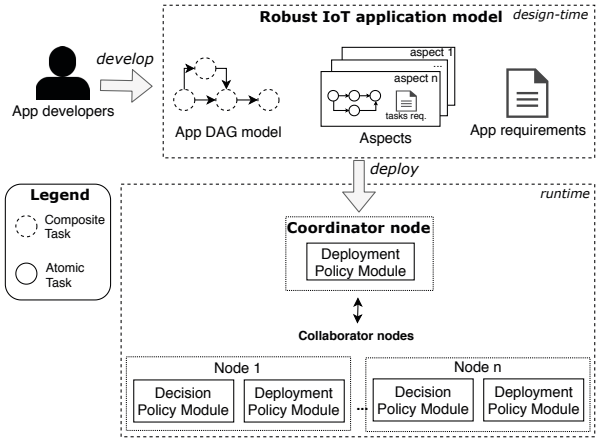

Fig. 1. Decentralized Resource Auctioning: Overview.

(i) find collaborators, (ii) find allocation, and (iii) application update. The first phase finds collaborators by advertising the application to its neighbors (i.e., all nodes that share a direct communication link with the coordinator). During this phase, the coordinator becomes a collaborator as well creating a bid for the received application. The second phase finds a satisfiable task allocation based on the received bids, while the third updates the composite tasks with new aspects.

An edge node becomes a collaborator once a node receives an advertisement message from the application coordinator. A collaborator has total control over its available resources, being able to create a specific bid based on its current internal status; a bid is composed of offers, where an offer contains a set of tasks. Each offer sent in a bid has the objective of maximizing the available resources and task coverage. The former objective aims at maximizing the allocated available resource for the advertised application; since we empower nodes to take local decisions, the strategies to allocate certain resources to share is defined by its administrative entity and can be different from the total available resources of a node. The latter objective, represents the ability of a bid to cover all advertised tasks, i.e., there is at least one offer for each task. Both equally important in creating improved bids that aid the coordinator in finding a satisfiable deployment. In the case of receiving multiple concurrent advertisements, a first-in first-out synchronization strategy is used; a bid is created for the first message received and the resources used are locked until a decision is made regarding that bid.

After the coordinator distributes the tasks to the collaborators, a collaborator role is changed to coordinator for the received tasks. In this case, the node can use the deployment policy module deploying the composite task instead of the entire application. By becoming a coordinator, a task can improve its functionality based on the available resources found locally or in the neighborhood of the host node. This is an important feature that ensures the best application functionality possible at the current time considering the host available resources. Furthermore, it shows that there is no need to obtain the optimal task allocation during application deployment

since (i) the application can achieve maximum functionality when more resources are available at the host of each task and (ii) the application is deployed in a dynamic network where uncertainty is introduced by device heterogeneity and mobility.

An overview of our technical framework is shown in Figure 1. The procedure starts at design time when an application developer defines his/her robust application model. To properly define the application, the developer must provide the data-flow between composite tasks as a DAG and the application requirements. During this process, the developer uses a model-based system engineering tool to model the composite tasks and their aspects as well as the communication flow. For each composite task, a set of aspects can be defined to specify different functionality level of the task; an aspect is defined by a DAG and the resource requirements of every task part of that aspect. Finally, the developer chooses an edge node to deploy it on; a node that becomes the coordinator of the deployed application. At runtime, the coordinator receives the application and starts the deployment procedure.

Motivational Example. Serving as our running example, we consider a public safety application, deployed in a smart city scenario, with the purpose of monitoring public areas. Although the main objective of the application is, for example, finding wanted criminals and discovering stolen cars, it is also desirable to aid the authorities with finding missing persons, if enough resources are available in the platform. For this purpose, the application is composed of distinct composite tasks, including *people analysis* and *environment analysis* tasks.

Privacy and low e2e delay latency define our public safety application, requirements that may introduce significant challenges when deploying to the cloud. First, to ensure privacy requirements and low latency, data must be processed near its origin, making a cloud deployment less desirable. Furthermore, the application should ensure correct functionality even when a stable connection to the cloud is missing. Consequently, a full deployment close to the origin of data is desirable; where the IoT application is distributed among multiple edge devices. In our scenario, an edge device may be a static device such as a CCTV camera as well as devices with mobility that may leave the area where the application is deployed at any time like a dash camera found in a car or images saved in a smart phone.

III. Problem Formulation

The underlying premise of edge computing is a distribution of resource-constrained devices at the edge of the network [6]. Thus, a monolithic application cannot be successfully deployed on a single edge device, due to the lack of available resources. As a result, current research [7] has focused on modeling such applications as a DAG. An approach that still faces some challenges because its success is dependent on the available resources of an edge node; a task cannot be further divided to be able to find a satisfiable mapping. In our conception, an application model is composed of interconnected composite tasks, where each task has defined a set of different configurations. Each configuration has a set of resource requirements that must be fulfilled upon deployment. In this section, we describe our application model, the considered objectives, and our assumptions.

A. Application and System Models

We consider that the edge computing architecture is defined by uncertainty introduced by mobile and heterogeneous devices, each having limited available resources, $R_E = \{r_1, r_2, r_3, ...\}$. Let E_N represents the total number of such devices deployed in the same neighborhood, $E_N = \{E_1, E_2, E_3, ...\}$. We do not have a predefined network topology, devices knowing only the nodes that share a direct communication link; all devices from the same neighborhood form a peer-to-peer network. Besides the available resources, a node has sensors to collect data from its surroundings and actuators to perform different actions. In Figure 2, an overview of a neighborhood is shown, where each device can be a coordinator or collaborator.

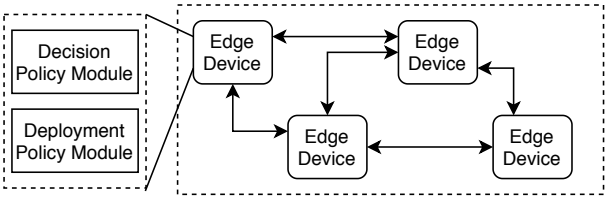

Fig. 2. Edge Computing Platform Architecture: Overview.

IoT applications may be deployed in such edge computing architectures; we assume an application model consists of multiple composite tasks and can be modeled as a hierarchical graph. A composite task has a set of configurations, covering a wide range of functions, all performing at least the critical functionality. Furthermore, we assume an application is described by a communication flow starting with a sensor task and ending with a sink (i.e., the actuator task), defining the path for which we want to compute the e2e delay. The e2e delay represents the total time required by an application to complete its execution; as such is provided by the developer as an application requirement to ensure proper functionality. Such an application division overcome the limitations of an edge device to execute the whole application locally.

Our application model is inspired by the aspect-oriented flow-based modeling paradigm [5], where cross-cutting concerns of an application, e.g., security, persistence, synchronization, fault detection, can be modularized as an aspect. In this paper, we see the functionality level of an IoT application as a cross-cutting concern which is impacted by the resource availability of edge devices; therefore it can be modeled as an aspect to adopt the application's behavior and functionality according to the available resources. The application developer specifies these aspects for different levels of functionalities, e.g., from a very critical operating mode to the fully functional operating mode, along with their priorities. Our resource management technique takes these aspects into account and tries to host the application according to available resources and the given functionality aspects of each composite task.

To be concrete, we assume that an application model consists of a set of composite tasks $T = \{t_1, t_2, t_3, ...\}$, where each task has defined a set of aspects (i.e., a configuration), e.g., $t_1 = \{a_1, a_2, a_3, ...\}$. The model uses a DAG, $G_{app} = (V, E)$, to model the data flow between composite tasks, where vertices represent a composite task and edges show the dependencies between them. The application model for our motivational example is presented in Figure 3.

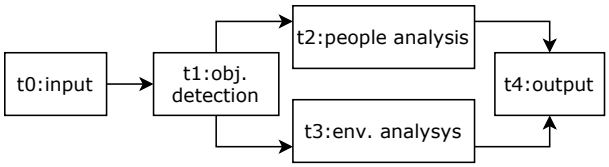

Fig. 3. Smart Building Application model.

A task t_i performs a specific application feature that is integral to the overall application performance. To this end, we assume each composite task has defined a set of aspects from which it will inherit their resource requirements like memory (i.e., RAM), storage (i.e., HDD), and computational power (i.e., CPU); resource requirements and functionality are dependent on the chosen aspect. Let us consider that the *people analysis* task, from our motivational example, has a set of 2 predefined aspects, $t_2 = \{a_1, a_2\}$. Aspect a_1 represents the desired functionality, having the highest resource requirements, while a_2 represents the minimum functionality level with the lowest resource requirements (see Figure 4 and 5).

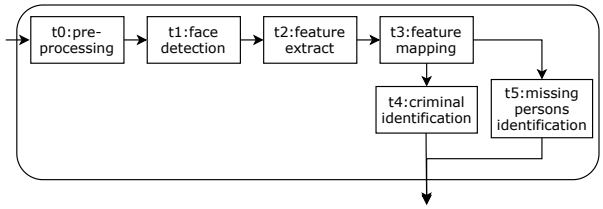

Fig. 4. Aspect 1 (a_1): Face recognition for missing persons.

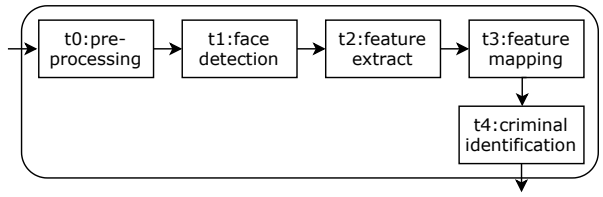

Fig. 5. Aspect 2 (a_2): Face Recognition for wanted persons. [8].

An aspect a_i defines the configuration and functionality of a composite task. Each aspect is developed at design-time and specifies at least the minimum functionality required by the application. Moreover, an aspect can enable a range of different functionalities that a composite task should perform. As a result, we have established that aspects will have different priorities based on their functionality and resource requirements. For example, the base configuration for a task is considered a_1 which always has the largest resource requirements and provides the maximum functionality. As such, this will have the highest priority. At the end of this spectrum, the last aspect, i.e., a_2 in the aforementioned example, has the lowest priority; having the smallest resource requirements and minimum functionality. In conclusion, an aspect is defined by a set of resources, $R_a = \{r_1, r_2, r_3, ...\}$, and a DAG modeling the workflow between its subtasks. With such a modular approach, a task can have different aspects depending on the available resources of the participant node; helping to efficiently use the available resources of edge device.

The e2e delay is critical to the overall deployment strategy, hence we define the two important components required for its computation: (i) communication latency and (ii) worst-case execution time (WCET) of a task. The former is dependent on the task allocation to nodes since the latency between t_i and t_j, i.e., l_{t_i,t_j}, is equal to the latency of their host nodes, i.e., l_{E_i,E_j}. For example, consider that t_1 is mapped on E_1 and t_2 on E_2, resulting that $l_{t_1,t_2} = l_{E_1,E_2}$. The latter is dependent on the host internal status and must be computed locally when a bid is sent. In a dynamic network composed of heterogeneous devices, finding the WCET of a task at design-time is not possible since we do not know the network topology and the current status of each node. However, with a decentralized solution, where nodes are empowered to make their own decisions, these challenges are overcome. In conclusion, we can find the WCET of a task when the bid is created, by computing its value based on the current CPU load. Since our focus is to efficiently use the node's available resources, we assume that the latency and WCET are already provided by a latency monitoring module as well as a WCET module.

B. Technical Framework Objectives

Our objectives differ depending on the role the node will play, i.e., a coordinator or a collaborator.

For a coordinator node, the main objective is to satisfy the *e2e delay* requirement of the deployed application. The *e2e delay* is composed of the communication latency between tasks and an overhead latency from the task allocation. The overhead latency is composed of the WCET of that task on its node plus the e2e delay of that composite task if the node becomes its coordinator. For example, consider the *authentication* task $t_2 = \{a_1, a_2\}$, where after the deployment strategy is mapped on node E_1 having as its configuration a_2; which has the lowest resource requirements and minimal functionality. In this case, to improve the performance of the application the task can be upgrade to a higher functionality level if E_1 becomes its coordinator and finds the required resources in its neighborhood. As a result, the overhead latency for t_1 can be equal to the WCET of the task if further deployment has failed, or WCET + e2e delay of the deployment result.

For the collaborator node, the objective is to provide a bid for the advertised application such that maximizes (i) the utilization of available resources, (ii) the task coverage, and (iii) the application functionality by minimizing the penalty of a bid. The penalty is dependent on the priority of an aspect, e.g., the highest priority aspect has the smallest penalty, while the lowest priority aspect has the highest penalty. All are based

on the local status of the collaborator (i.e., available resources and CPU load) and may differ at different points in time.

C. Assumptions

It is important to have a good understanding of our assumptions since it offers a completeness for our technique. As such, we present below our assumptions; their development is out of the scope of this paper.

Latency. A core component of our deployment objective as such is imperative to measure it at runtime when an application is deployed. For this reasons, we assume that there is a third module, i.e., a latency monitoring module, capable of providing the latency.

WCET. The second part of computing the e2e delay latency, hence it must be known at deployment time. As a result, we assume that each individual collaborator is capable of computing the WCET of its bid, based on the current internal status, i.e., the CPU load.

Incentives. A requirement mechanism in a distributed system where a collaboration between devices, owned by different administrative entities, is required to achieve a certain goal. In our scenario, we consider that these devices trust each other and are part of a community (i.e., the neighborhood). Devices from the same community share resources freely since it is beneficial for all parties involved; it offers the possibility exchange computational resources between devices, inside the community, to deploy applications.

Application development. Done at design-time using a modeling tool that aids the developer to model the application's tasks and communication flow. Once the application is modeled and verified, the model is extracted in a JSON file and sent to be deployed by the chosen edge device.

IV. Application deployment framework

From our resource management technique overview presented in Section II, we identify two major components, i.e., *the deployment policy module* and *the decision policy module*. The former implements an extension of a decentralized resource allocation technique [4] which aims at deploying an application entirely on edge devices. The latter empowers each participant node to take local decisions concerning their available resources.

A. Deployment policy module

The deployment policy module retains its functionality proposed in previous work [4], i.e., to find a task allocation on edge devices that satisfies the application requirements. However, we extend it with more functionality to support our robust application module and update the application functionality at runtime. Therefore, the functionality of our coordinator consists of three distinct parts:

- *Find collaborators.* The procedure starts when a request to deploy an IoT application arrives at a node. Under these conditions, the edge node becomes a coordinator for that application's lifespan. The coordinator prepares an advertisement message containing: (i) a description of the application model and (ii) the maximum duration allowed to receive a bid.
- *Find allocation.* Once all bids are collected from the participant nodes, the procedure of finding a satisfiable task allocation begins. The coordinator selects a mapping based only on the information received without knowing extra details on each participant. In the end, the solution always satisfies the application requirements, but not a bid; a bid can be fully or partially satisfied.
- *Application update.* Once the winner nodes are decided, the deployment policy module still keeps track of these devices. As a result, a developer can send, at a later stage, new aspects for each composite task to expand the application functionality. When a new aspect is received, the coordinator sends it to the location of its target composite task. Consequently, we can perform on the fly updates on the application without any downtime.

Due to the problem complexity of resource allocation techniques, a typical approach used in related work is based on heuristic optimization algorithms to find a mapping between the deployed application and an edge computing architecture [9]. However, this approach is not suitable for us, since we aim at allowing resource-constrained devices the be coordinators and perform deployments in a decentralized manner. As such, we move towards a less computational demanding approach, i.e., using a generalization of the Boolean satisfiability (SAT) problem called satisfiability modulo theories (SMT) [10]. SMT represents a technique that verifies if a first-order logic formula, composed of predicates symbols, is satisfiable. Considering this, we have translated our task allocation objectives into a SMT formula, using the following encodings: *task facts*, *domain facts*, *latency facts*, *bid constraints*, *aspect facts*, *aspect constraints*, and *constraint formulation*. The first four encodings represent the base we use from [4], while the rest are our contribution to this module.

First, we briefly introduce the first four encodings. *Task facts* ensures that the solver maps a task on a single participant node. To be considered as a valid choice, a node must send a bid for that particular task. Since there is no consensus between participant nodes, a task may receive multiple bids from distinct nodes. Therefore, we must guarantee that only one node is chosen for each task. Next, *latency facts* and *domain facts* helps to determine the latency between two dependent tasks; the latency is dependent on the current host nodes of each task. Since we don't know the solution found by the solver when the model is built, we must ensure that all possible combinations of tasks mapped on nodes are considered. Finally, *bid constraints* ensures that only one offer is chosen from a bid. As a result, we know that a solution does not exceed the available resources of a collaborator.

The *aspect facts* provides the possible aspects a task can have; information important for the e2e delay computation, since each aspect has associated an execution time. Hence, knowing the aspects of each task provides a knowledge base used by the solver to enforce the last two encodings, i.e., *aspect constraints* and *constraint formulation*. The encoding

is shown in Equation 1, where *latency* returns the overhead latency of aspect *a* from task *t*.

$$\mathsf{aspectFacts} = \mathsf{Or}(\mathsf{t_a} = \mathsf{latency(a)})) \quad \forall\, \mathsf{t} \in \mathsf{T}. \tag{1}$$

The *aspect constraints* aid the solver in getting the correct WCET for a task, based on its mapping. Similar to the *latency facts*, this constraint helps the solver to create a relation between the mapping of a task and its WCET. As we know, the WCET is closely related to the current state of the node. Hence, we need to know where the task is mapped to consider its WCET in the *constraint formulation*. Furthermore, since a bid contains multiple offers, we must ensure that we get the correct WCET for a task according to the rules presented in *bid constraints*. For example, node $\mathsf{E_1}$ sent a bid $B = \{\mathsf{O_1}, \mathsf{O_2}\}$, where $O_1 = \{\mathsf{t_1}, \mathsf{t_2}\}$ and $O_2 = \{\mathsf{t_1}\}$. Each task has associated an aspect and its WCET. As a consequence, if $\mathsf{t_1}$ is mapped on $\mathsf{E_1}$, then we have two possible WCET associated with it (one for each offer). If $\mathsf{t_1}$ has the same aspect in both offers, then the WCET of the task will be the value of the aspect. However, if the two offers have different aspects for $\mathsf{t_1}$, then the solver will choose the aspect from the task that satisfies the *bid constraints* rules.

Finally, the *constraint formulation* ensures that the solution satisfies our objectives. Such encoding accounts for both the communication latency and WCET of a task; the result must be smaller or equal to the application requirements. A rule that helps the solver to verify the satisfiability of a solution found using the aforementioned encodings. The formula is shown in Equation 2, where *d* represents the total number of dependencies between two tasks.

$$\mathsf{e2eDelay} = \sum_{\mathsf{i}=1}^{\mathsf{d}} \mathsf{l_i} + \sum_{\mathsf{t}}^{\mathsf{T}} \mathsf{t_a} \leq \mathsf{app\ requirements} \tag{2}$$

B. Decision policy module

The *decision policy module* has the purpose of aiding the collaborators in creating a bid for the advertised application. The procedure starts once a node receives the advertised message from the coordinator. Based on its current status, the node can become a collaborator if it has the required resources to host at least one task. To aid in its decision, we have developed a strategy to create a bid for each message received. A bid contains multiple offers and has the following three objectives: (i) maximize the utilization of available resources for each offer, (ii) maximize the coverage of tasks per bid, and (iii) maximize application functionality. Finally, once the bid is created, the collaborator reserves the required resources for the bid until a response is received from the coordinator to avoid any conflicts. Furthermore, for each task sent in the bid, it computes and send the tasks' WCET. In the end, the freshly created bid is sent to the coordinator.

To obtain a bid that satisfies all objectives aforementioned above, we have developed a new strategy capable of finding the best bid in the given amount of time. The strategy yields an optimal bid only if the solution can be found in the available computational time received from the coordinator. If the time has expired, the strategy produces a feasible solution that guarantees the constraints are met. As a result, we have decided to use Constraint Programming [11] to find our bid. With constraint programming, we can describe a model using *decision variables*, *constraints*, and *global objectives*.

Based on the advertisement message, we can create our *decision variables*. A decision variable aims at defining for each task a domain representing all the possible aspects the task can have. Let's assume that $\mathsf{t_1}$ has a set of two different aspects, i.e., $\mathsf{t_2} = \{\mathsf{a_1}, \mathsf{a_2}\}$. In these conditions, decision variable $\mathsf{d_{t_2}}$ can have assigned in the model only one aspect from the domain. Besides the task variables, to be able to maximize the available resources of a node in an offer, we must define three new decision variables to keep track of the resource requirements of a chosen aspect. Meaning that the domain of a resource decision variable, i.e., RAM, CPU, and HDD, has the same length as the domain of the task. For example, the domain for the RAM resource variable for $\mathsf{t_2}$ is $\{\mathsf{a1_{RAM}}, \mathsf{a2_{RAM}}\}$. Finally, we define a penalty variable to aid the solver in finding solutions containing higher priority aspects. For the highest priority aspect we have attached a penalty of 0, while for the rest aspects there will be a penalty increment of 2. In the end, if the task is not placed in an offer, it will have the largest penalty of 20.

Now that we have added all decision variables and their related domains to the constraint programming model, we can start defining *constraints* that will guide the solver in finding a feasible solution. We have created two major constraints, i.e., (i) *offer constraints* and (ii) *max coverage constraints*.

The aim of our *offer constraint* is to ensure that an offer does not exceed the available resources of a collaborator. Two important parts are found in this constraint, i.e., *set resource decision variables* and *check available resources*. The former aims at creating a logical relation between an aspect of a task and its resource requirements (see Equation 3). The latter ensures that the sum of all resources required for the current offer does not exceed the available resources of that node (see Equation 4, where n_t is total number of tasks from an offer).

$$\begin{aligned}\mathsf{setResource} : (\mathsf{t} = \mathsf{a}) \Rightarrow (\mathsf{t_{CPU}} = \mathsf{a_{CPU}} \\ \wedge\, \mathsf{t_{RAM}} = \mathsf{a_{RAM}} \wedge\, \mathsf{t_{HDD}} = \mathsf{a_{HDD}}) \\ \forall\, \mathsf{t} \in \mathsf{T}\, \mathsf{and}\, \forall\, \mathsf{a} \in \mathsf{t_a}.\end{aligned} \tag{3}$$

$$\begin{aligned}\mathsf{maxCPU} &= \sum_{\mathsf{i}=1}^{\mathsf{n_t}} \mathsf{t_{CPU}} \leq \mathsf{availabe_{CPU}} \\ \mathsf{maxRAM} &= \sum_{\mathsf{i}=1}^{\mathsf{n_t}} \mathsf{t_{RAM}} \leq \mathsf{availabe_{RAM}} \\ \mathsf{maxHDD} &= \sum_{\mathsf{i}=1}^{\mathsf{n_t}} \mathsf{t_{HDD}} \leq \mathsf{availabe_{HDD}}\end{aligned} \tag{4}$$

The aim of our *max coverage constraint* is to maximize the task coverage in a bid. To achieve this, we verify that across all offers in a bid there is at least one offer for a task (i.e., a *strict bid*). A *strict bid* is a bid that has *100 %* task coverage; all

other bids with a lower task coverage are considered infeasible. We can make the constraint soft if we change the > 0 to ≥ 0 and obtain a *permissive bid*. In this case, a *permissive bid* considers feasible all solutions independent if a task does not receive an offer from a collaborator. The constraint is shown in Equation 5, where n_o is the total number of offers sent in a bid and *priority* returns the priority of the aspect assigned to task *t*:

$$\mathsf{maxCoverage} = \sum_{\mathsf{i}=1}^{\mathsf{n_o}} \mathsf{priority(t)} > 0 \quad \forall \mathsf{t} \in \mathsf{T}. \tag{5}$$

Finally, to enable our decision policy to yield the best solution under the given time, we have defined a *global objective*. The purpose of the objective is to minimize the penalty at the bid level. As a result, we will obtain a bid that prioritizes the higher priority aspects for each task, while maximizing the utilization of all available resources.

V. Evaluation

A major goal of this paper is to efficiently utilize the available computational resources found on resource-constrained devices when deploying a latency-sensitive application on an edge architecture. To this end, we evaluate our proposed application model in terms of obtained successful mappings at the edge of the network. We first evaluate the decision strategy module to understand the effects of various applications and device local available resources on (i) the total number of offers sent in a bid and (ii) computational time required to find a feasible solution. Finally, with this knowledge, we proceed in deploying three different applications, (i) *montage*, a real-world DAG workflow [12], (ii) a cognitive application [13], and (iii) a mockup application defined by us.

A. Decision policy module: Experiment and Results

We separately evaluate the decision policy module and its bid strategy to demonstrate the collaborator's ability to provide feasible bids. The purpose of this evaluation is to observe how the following impact task coverage: (i) edge node available resources, (ii) application size (i.e., the total number of composite tasks), (iii) available computational time (i.e., the maximum time allowed to find a bid), and (iv) number of offers sent in a bid. Furthermore, we choose to evaluate two different types of bids (strict and permissive) by making the *max coverage constraint* a hard constraint and a soft constraint. With the former, each bid achieves *100%* task coverage; all bids with a lower percentage are considered infeasible. With the latter, bids with lower task coverage are accepted.

The maximum number of offers inside a bid plays an important role in the overall performance of the deployment technique. Depending on the type of bid, the number of tasks may increase or decrease the performance. For example, a strict bid requires a minimum number of offers to find a feasible bid; the minimum number of offers is dependent on the device available resources and application size. On the other hand, a permissive bid requires more computational time and available resources to yield better solutions.

The experimental setup for this policy consists of sending as an advertising message the *montage* application to a collaborator node and record the obtained bid; we chose this application due to its large size, making it harder to find a feasible bid. For each bid, we have a set of different metrics that we change such as available computational time (i.e., 1, 5, 15, 30, 45, and 60 seconds), number of offers (i.e., 2, 4, 6, and 8), available resources (12, 36, and 60 units for each considered resource), and type (permissive and strict). In our evaluation for this module, for each set of available resources we obtain the task coverage for every considered bid size at different computational times. For example, for a node with 12 units of available resources we obtain a total of 48 bids, i.e., for each bid size we change the available computational time resulting in 6 total bids per size; this approach is performed for both types of bids (see Table I).

Our results are shown in Table I and Table III where the task coverage is shown as the intersection between the number of offers and the execution time. For example, in Table I, a permissive bid with two offers having the execution time of 1 manages to offer a task coverage of 25 %.

Bid Type	# offers	execution time (seconds)					
		1	5	15	30	45	60
Permissive	2	25	25	25	30	29	29
	4	46	36	46	50	50	50
	6	55	58	50	50	42	42
	8	63	67	63	63	58	58
Strict	2–4	0	0	0	0	0	0
	6–8	100	100	100	100	100	100

TABLE I
TASK COVERAGE PERCENTAGE OF PERMISSIVE AND STRICT BIDS WITH 12 UNITS AVAILABLE RESOURCES

In Table I, we can observe that with the current available resources, it is impossible to find strict bids when the number of offers is 2 and 4. A result of the condition that a strict bid is considered feasible only when its task coverage is *100 %*. In comparison, since the conditions are more relaxed, a permissive bid can offer a task coverage in the rage between *0 %* and *100 %*; the only chance to not get a feasible bid is when the collaborator does not have available resources to host at least one task from the application.

The effects of increasing the collaborator's available resources on the task coverage of each bid type is shown in Table III and II. We can observe that it is possible to obtain a feasible strict bid for all sizes, while an improvement in the overall task coverage of a permissive bid is seen. We can observe that with more execution time and available resources a permissive bid can achieve *100 %* task coverage.

To evaluate of decision module, we have implemented the bid strategy using *CP-SAT solver* provided by Google OR-Tools [14]. We perform our measurements by deploying the collaborator on a machine with a single-core Intel i5 2.3GHz processor.

Bid Type	# offers	execution time (seconds)					
		1	5	15	30	45	60
Permissive	2	58	63	50	50	58	58
	4	63	58	71	71	71	66
	6	66	66	83	79	79	79
	8	86	88	100	100	100	83
Strict	2–4–6–8	100	100	100	100	100	100

TABLE II
TASK COVERAGE PERCENTAGE OF PERMISSIVE AND STRICT BIDS WITH 36 UNITS AVAILABLE RESOURCES

Bid Type	# offers	execution time (seconds)					
		1	5	15	30	45	60
Permissive	2	71	71	75	71	71	71
	4	75	88	92	92	92	92
	6	92	92	100	92	92	96
	8	83	83	83	83	96	100
Strict	2–4–6–8	100	100	100	100	100	100

TABLE III
TASK COVERAGE PERCENTAGE OF PERMISSIVE AND STRICT BIDS WITH 60 UNITS AVAILABLE RESOURCES

B. Deployment policy module: Experiments and Results

We consider as a performance metric the deployment strategy ability to find a satisfiable mapping using only edge available resources. Furthermore, we perform a comparison between our robust application model and the DAG model looking at their impact on the deployment strategy. Finally, we compare the deployment efficiency of both the strict and permissive bids; it is important to understand the impact of strict bids on the deployment results. For our experiment setup, we simulate the deployment of the three IoT applications aforementioned on an edge computing platform architecture, evaluating both the applicability and performance.

1) Applicability: Cognitive Application Deployment: Based on the results presented in Section V-A, for our applicability evaluation, since the application consists of 8 tasks, we set the configuration as follows: (i) the available resources (i.e., RAM, CPU, HDD) on each collaborator node is randomly set between *12 and 24 units*, (ii) a bid contains two offers, (iii) the computational time of a bid is *1 sec*, (iv) we use strict bids, and (v) the e2e delay is set to *50*. Our target edge computing architecture is composed of two resource-constrained devices, that can be both coordinators and collaborators for the deployed applications.

We define the *cognitive application* using our proposed robust model, assigning to each composite task a set of four different aspects. Each aspect has a set of resource requirements chosen in the range of 1 to 9 units; the lowest priority aspect has resource requirements closer to 1, while the highest priority aspect has resources close to 9. For the evaluation purposes, the WCET was chosen randomly between 1 and 10 for each aspect. The cognitive application model is shown in Figure 6.

The coordinator manages to find a satisfiable mapping in *40 ms* (i.e., the time required to find a mapping after the bids were received) at which the time required to compute the bids is added (i.e., *1 sec*); a total of *e2e delay = 49* was obtained, from which 12 comes from communication latency and 37 from WCET. The application mapping is shown in Figure 6,

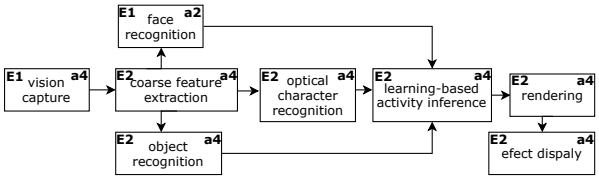

Fig. 6. Cognitive Application [13]

where on each composite task on the left upper corner we see its edge node, while on the right upper corner we have the chosen aspect. For example, t_1 is mapped on E_1 with the aspect a_4. Collaborator E_1 has the following available resource, R_E = {RAM = 19, CPU = 21, HDD = 18}, while E_2 has R_E = {RAM = 21, CPU = 22, HDD = 15}. The task's required resources based on their chosen aspect and related WCET are presented in Table IV.

Composite Tasks	Resource Requirements			WCET
	RAM	CPU	HDD	
t_1	3	1	1	1
t_2	3	2	4	7
t_3	4	5	5	1
t_4	3	1	2	7
t_5	3	1	3	3
t_6	3	1	1	9
t_7	3	1	3	6
t_8	3	1	1	3

TABLE IV
COGNITIVE APPLICATION RESOURCE REQUIREMENTS AND WCET.

The coordinator found a satisfiable solution for our cognitive application with its application functionality close to the minimum. The reason for this behavior is introduced by the strict bids, which ask for 100% task coverage in a bid. Additionally, the number of offers is set to 2, which further limits the possibility of increasing the application functionality. However, the application functionality can be maximized later locally at the collaborator level.

2) Performance evaluation: Considering the results presented in Section V-A, we concluded that the following configuration is suitable for our performance evaluation: (i) a collaborator has resources between 26 and 50 units, (ii) a bit has a total of *6 offers*, (iii) the computational time to receive a bid is set to *1 sec*, and (iv) the e2e delay is set to a high value to avoid any unsatisfiable deployments due to incorrect assignment.

To perform a better evaluation, we deploy two different size applications, i.e., the *montage graph* with a total number of 24 tasks and the *mockup application* having 16 tasks. By deploying different size applications we can evaluate the effect of total available resources and application size on the deployment efficiency. Similar to the cognitive application model, we assign to each composite task a set of four distinct aspects. However, for our edge architecture, we consider five different sizes (i.e., 2, 4, 6, 8 and 10).

Our results are shown in Table V where we compare three different deployments based on the total number of successful application mappings on edge devices. In the first deployment, the collaborators send only strict bids with 100%

task coverage, while in the second we change to permissive bids. For these two, we use our robust IoT application model. Finally, the third deployment used the DAG application model without the aspects.

Application	Type	Number of nodes				
		2	4	6	8	10
Montage	strict bid	0	81	100	100	92
	permissive bid	0	0	8	92	97
	DAG deployment [4]	0	0	0	0	0
Mockup	strict bid	44	100	100	100	98
	permissive bid	0	6	100	100	99
	DAG deployment [4]	0	0	0	0	0

TABLE V
SUCCESSFUL MAPPING ON EDGE.

We can see that deploying the *mockup* application without changing the edge architecture, the deployment strategy performs better compared to a larger application (i.e., *montage*); both permissive and strict bid deployments are capable of mapping all tests to the edge of the network when the architecture size increases. Moreover, the deployment using bids with 100% task coverage yields much better results compared to the other two deployments. We can observe that by deploying an application using our robust model we can achieve far better results compared with the normal DAG application model. In the case of normal deployment using the DAG model, the deployment cannot find one successful mapping at the edge.

The implementation of our coordinator is based on *Z3* SMT solver [15] and is deployed on the same device as described in Section V-A; all collaborators used for the performance evaluation are simulated on the same device.

C. Discussion

In Section V-A we have demonstrated that our decision policy module enables a collaborator to provide feasible strict and permissive bids. A strict bid is dependent on the number of offers sent in a bid as well as the device available resources. Such a behavior is seen in Tables I and III, where with an incremental increase in both metrics we achieve 100% task coverage independent of the number of offers. As a result, we can consider the configuration, with 36 units of available resources, an optimal node available resources for deploying the montage application. In contrast, a permissive bid is more flexible finding good bids for all configurations. We can observe from our results that the task coverage increases with the available resources of a node. Furthermore, compared to a strict bid where the objective is to maximize coverage, a permissive bid aims at providing better application functionality (i.e., chooses the highest priority aspect for each task). As a result, we see a fluctuation in task coverage given by the available computational time.

In Section V-B2 we show the coordinator's ability to successfully find an application mapping to resource-constrained devices. In the first part, with the aid of our experiments, we have proved that a satisfiable deployment can be found for a realistic IoT application; even when there are not enough available resources for the entire application functionality. In the second part, we focus on demonstrating that our proposed robust application model in combination with an efficient decision policy module, we can achieve better results compared with a normal DAG application.

From Table V we can observe that even for a larger application size we can provide full edge deployments for more than *80%* of total tests if there are the available resources required by the minimum application functionality. Considering this, we do not find a satisfiable deployment on the architecture composed of two collaborators, because it lacks the application's minimum required resources. In contrast, by adding more collaborators, we increase considerably the chances of finding good solutions. Comparing the different deployments (i.e., permissive bid, strict bid, and DAG deployment), we notice that deployments using strict bids outperform the other permissive bids on smaller architecture sizes, while the normal deployment is incapable of finding one satisfiable solution. In conclusion, using bids that offer full task coverage is desired to fully utilize the available resource found on edge nodes.

Comparable results are obtained when deploying our *mockup* application model. In this case, the deployment using strict bids outperforms the other two. We can see that by lowering the size of the application, we lowered the required available resources enabling the coordinator to find a mapping for the smallest architecture as well.

As a final note, we acknowledge that the current evaluations were performed on a laptop and not on a resource-constrained deceive (e.g., a Raspberry Pi). Even though the computation times for creating a bid may increase, the evaluated functionality is still valid. One limitation of our provided deployment technical framework is the inability to automatically decide at runtime, based on the application size and the available resources, what is the optimal number of offers. In this case, we had to perform multiple evaluations for all deployed applications, having the number of offers hardcoded into the decision policy module. As a solution, we intend in future work to develop such a method that decides at runtime the optimal bid size considering the deployed application.

VI. RELATED WORK

Recent resource management techniques have been proposed in the scientific literature to migrate an IoT application from cloud closer to the origin of data or to offload high computational tasks from resource-constrained devices.

A deployment technique to map a multi-component application model to a fog infrastructure is presented in [9]. Its objective is to find an allocation of components to fog devices such that the application's requirements are met. For a similar problem formulation, authors in [8] proposed a constraint programming model, extendable in terms of deployment constraints and objectives, which can obtain a competitive result in relation to heuristics and meta-heuristic algorithms.

An edge orchestration technique is presented in [16] focusing on providing task deployment based on resource proximity. The orchestrator is deployed on a router and dynamically find a task allocation based on the application requirements forwarding the tasks to either cloud or edge devices.

A heuristic task offloading aiming to migrate computational tasks from mobile devices to a heterogeneous architecture composed of both cloud and edge devices is presented in [2]. The offloading decision is based on multiple objectives such as application's runtime, mobile device energy, and user cost. Another multi-objective task offloading technique is presented in [17] aiming at finding a balance between users satisfaction and providers' profit.

Multiple resource auctioning techniques were proposed in the literature to distribute applications between devices. In [18], an auction-based technique is proposed that enable users to bid for the available computational resources of an edge server. Once a bid arrives, the server computes the price for the requested resources taking into account its location, i.e., cloud or edge. Similarly, in [19] an auction-based solution is presented to map the requests of bidders (i.e., mobile devices) to the available resources of an edge server.

Compared to the aforementioned technical papers, our proposed technical framework differs as follows: (i) we perform decentralized application deployment without requiring any knowledge at design time of network topology or device internal status, (ii) introduce a new robust IoT application model to achieve adaptive application functionality based on available resources of each collaborator, and (iii) we empower each edge node to make local decisions regarding the current available resources.

VII. Conclusion and Future Work

In this paper, we substantially extend our previous work [4] with a new decision policy module and provide a novel robust IoT application model. The former consists of developing a decision policy module that empowers collaborator nodes to make better decisions regarding their available resources; a policy that aims at providing feasible and optimized bids. The latter enables the developer to define different application's functionality levels. An approach that (i) enables a resource management technique to efficiently use the available resources found on edge devices, (ii) allows expanding the application functionality at runtime, and (iii) empowers each collaborator to adapt the task's functionality based on the available resources found locally or in the neighborhood. Based on our evaluation, we have proven that significant improvements are seen when deploying an IoT application defined using our proposed model. Moreover, we demonstrate that finding strict bids with 100% task coverage yields better results compared to permissive bids.

Regarding future work, we plan to investigate the possibility of finding the required number of offers based on the device available resources, aspect requirements, and application size. Additionally, we plan to investigate the relationship between application size and available computational time to create and send a bid. Furthermore, we intend to develop a task ranking at the device level to help a collaborator choose what task functionality should be upgraded first in the case when multiple tasks are mapped on the same node and the available resources have increased.

Acknowledgment

This research has received funding from the EU's H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785 FORA–Fog Computing for Robotics and Industrial Automation.

References

[1] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of computing, caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[2] V. D. Maio and I. Brandic, "First hop mobile offloading of dag computations," in *18th IEEE/ACM Intl. Symposium on Cluster, Cloud and Grid Computing*, May 2018, pp. 83–92.

[3] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey," 2018.

[4] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Decentralized resource auctioning for latency-sensitive edge computing," in *IEEE International Conference on Edge Computing (EDGE)*, 2019.

[5] B. Zarrin and H. Baumeister, "Towards separation of concerns in flow-based programming," in *Companion Proceedings of the 14th International Conference on Modularity*, ser. MODULARITY Companion 2015. New York, NY, USA: ACM, 2015, pp. 58–63. [Online]. Available: http://doi.acm.org/10.1145/2735386.2736752

[6] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.

[7] R. Mahmud, K. Ramamohanarao, and R. Buyya, "Latency-aware application module management for fog computing environments," *ACM Trans. Internet Technol.*, vol. 19, no. 1, pp. 9:1–9:21, Nov. 2018. [Online]. Available: http://doi.acm.org/10.1145/3186592

[8] F. AIT SALAHT, F. Desprez, A. Lebre, C. Prud'Homme, and M. Abderrahim, "Service Placement in Fog Computing Using Constraint Programming," in *SCC 2019 - IEEE International Conference on Services Computing*. Milan, Italy: IEEE, Jul. 2019, pp. 19–27. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02108806

[9] A. Brogi, S. Forti, and A. Ibrahim, "How to best deploy your fog applications, probably," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, May 2017, pp. 105–114.

[10] C. Barrett and C. Tinelli, *Satisfiability Modulo Theories*. Cham: Springer International Publishing, 2018, pp. 305–343.

[11] F. Rossi, P. Van Beek, and T. Walsh, *Handbook of constraint programming*. Elsevier, 2006.

[12] L. F. Bittencourt, R. Sakellariou, and E. R. M. Madeira, "Dag scheduling using a lookahead variant of the heterogeneous earliest finish time algorithm," in *18th Euromicro Conf. on Parallel, Distributed and Network-based Processing*, Feb 2010, pp. 27–34.

[13] F. AIT SALAHT, F. Desprez, and A. Lebre, "An overview of service placement problem in Fog and Edge Computing," Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, LYON, France, Research Report RR-9295, Oct. 2019. [Online]. Available: https://hal.inria.fr/hal-02313711

[14] "Google or-tools," https://developers.google.com/optimization, accessed: 2019-11-21.

[15] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *Intl. conf. on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[16] I. Petri, O. Rana, A. R. Zamani, and Y. Rezgui, "Edge-cloud orchestration: Strategies for service placement and enactment," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*, June 2019, pp. 67–75.

[17] V. De Maio and I. Brandic, "Multi-objective mobile edge provisioning in small cell clouds," in *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ser. ICPE '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 127–138. [Online]. Available: https://doi.org/10.1145/3297663.3310301

[18] T. Bahreini, H. Badri, and D. Grosu, "An envy-free auction mechanism for resource allocation in edge computing systems," in *IEEE/ACM Symposium on Edge Computing*, Oct 2018, pp. 313–322.

[19] W. Sun, J. Liu, Y. Yue, and H. Zhang, "Double auction-based resource allocation for mobile edge computing in industrial internet of things," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4692–4701, Oct 2018.