Resource Management for Latency-Sensitive IoT Applications with Satisfiability

Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar

Abstract—Satisfying the software requirements of emerging service-based Internet of Things (IoT) applications has become challenging for cloud-centric architectures, as applications demand fast response times and availability of computational resources closer to end-users. Meeting application demands must occur at runtime, facing uncertainty and in a decentralized manner, something that must be reflected in system deployment. We propose a decentralized resource management technique and accompanying technical framework for the deployment of service-based IoT applications at the edge. Faithful to services engineering, applications we consider are composed of interdependent tasks, which in the IoT setting may be concretized as containerized microservices or serverless functions. A deployment for an arbitrary application is found at runtime through satisfiability; the mapping produced is compliant with tasks' individual resource requirements and latency constraints by construction. Our approach ensures seamless deployment at runtime, assuming no design-time knowledge of device resources or the current network topology. We evaluate the applicability and realizability of our technique over single-board computers as edge devices, particularly in the absence of cloud resources.

Index Terms—Resource Management, Edge Computing Services, Decentralization, Internet of Things

1 Introduction

Contemporary Internet of Things (IoT) systems consist of multiple heterogeneous computing nodes often running software tasks packaged as containerized services. Utilizing distributed computing resources within an IoT system is challenging, as applications often have stringent performance and deployment requirements.

Current cloud-centric designs fail to satisfy such requirements, due to inherent centralization limitations and the often high volumes of data required to be transferred to the cloud leading to congestion and bandwidth waste [1]. Those shortcomings can be tackled by taking advantage of distributed computational resources in the spirit of edge computing, where data processing occurs locally by functionality deployed on edge nodes, with advantages including data locality and fast response times [2].

IoT applications are often comprised of multiple interconnected tasks in turn characterised by special resource requirements which must be fulfilled upon deployment. This is in line with typical service-oriented applications, i.e., ones defined as a service composition, where interconnected services create a workflow to achieve a certain goal [3]. As such, we work within Service-Oriented Architectures (SOA), which in the IoT setting ensure interoperability among heterogeneous nodes making up the system, and abstract functionality as a set of well-defined services. SOA applied to IoT provides extensibility, scalability, modularity, and interoperability among heterogeneous software components; functionalities and capabilities are abstracted as a common set of tasks, where each represents a service.

Consider an application that is particularly data-intensive and requires low latency communication to function properly. To satisfy its requirements, a deployment strategy should as much as possible take advantage of available resources distributed at the edge of the network and avoid utilization of the cloud, as latency would be prohibitive and uplink bandwidth may be saturated. However, benefiting from distributed computational resources is not trivial and requires novel resource management techniques.

Resource management in this context [4] aims at enabling collaboration between edge nodes by sharing their available computational resources. IoT applications are deployed on possibly resource-constrained devices and in dynamic networks where high uncertainty is introduced by (i) node mobility, (ii) node heterogeneity, as an edge node can range from single-board computer to datacenter-grade, and (iii) lack of knowledge at design time of network topology and edge nodes' available resources. We propose a novel resource management technique focusing particularly on resource sharing and allocation for application deployment. Previously, deployment of applications at the edge of the network has been generally tackled from two perspectives: (i) task offloading from resource-constrained devices to improve objectives such as energy consumption [5] or (ii) relying on the cloud to perform task allocation [6]. Still, such approaches do not sufficiently take into account latency application requirements, do not consider node's preferences, and assume knowledge of participant nodes' internals. We have tackled resource management in previous work, providing (i) resource coordination for IoT systems [7], investigated (ii) research challenges inherent in decentralized resource management at the edge [8], and introduced (iii) resource auctioning as a resource management abstraction [9].

In this paper, we propose a decentralized resource allocation technical framework aiming to deploy applications at the edge of the network, guaranteeing adherence to (i) defined latency Service Level Agreements (SLAs) and (ii) resource preferences of participating nodes. We extend previous work [9] by focusing solely on task allocation performed by a resource-constrained single-board device, by

providing optimized procedures, and by framing our work within services engineering. Specifically, our contributions are:

- A decentralized task allocation technique for sharing IoT resources with nearby nodes based on application requirements;
- A scheme where participating nodes on the network may utilize multiple decision strategies, making their own choices regarding their contribution of their local resources, including data;
- We advocate decentralization, as the system can operate without an assumed connection to the cloud, if there are enough available resources at the edge of the network.

In our framework, an IoT application to be deployed consists of interdependent discrete tasks, which can be concretized as, e.g., containerized microservices or serverless functions. Nodes participating in the system individually select tasks they may host – perhaps based on some incentive scheme [10], [11]. The overall application has certain (strict) latency requirements, while other concerns may impose further constraints over where a task may be deployed as well. Our framework encodes the resource allocation problem within Satisfiability Modulo Theories (SMT [12]), where placement of tasks on edge nodes generates constraints in first-order logic while latency SLAs are encoded with integer linear arithmetic. Thus, we provide guarantees – if a mapping exists, it is always found at runtime by a solver situated in some edge node deploying the application, and is always correct, i.e., it satisfies latency SLAs, preferences of participating nodes, and other constraints.

Our framework provides seamless deployment for service-based IoT applications, independent of the target domain. We evaluate the applicability and performance of our technique, especially compared to the absence of cloud resources. Our obtained results demonstrate its efficiency for relevant problems, particularly on resource-constrained single-board edge devices. Our experiments show that our framework is capable of deploying IoT applications entirely at the edge, the SMT solver providing the mapping being deployed on a resource-constrained device as well.

The remainder of the paper is structured as follows. In Section 2, we present an overview of our solution and introduce a motivational example. Section 3 defines the IoT application and architecture considered in this paper. In Section 4, we describe implementation details of our proposed technique, while Section 5 presents the methodology and results of our evaluation regarding applicability and performance. Section 6 discusses related work on resource allocation techniques, and finally Section 7 concludes the paper and provides an outlook on future work.

2 Decentralized Resource Management

Novel types of distributed systems achieved through new paradigms such as the IoT are composed of heterogeneous nodes, computing infrastructures, and cloud services, recently known as the edge-fog-cloud continuum [13]. Generally, applications provide data-centric, device-centric, and service-centric functionalities where data, computation, or control is situated locally near nodes and not in the cloud. As such, resources needed for the application's operation must be dynamically allocated among different nodes and in a decentralized manner, with minimal central coordination.

Faithful to the services engineering viewpoint, we assume a general model where an IoT application is composed of interdependent *tasks* which need to be executed in some specific way to provide the application's functionality [3], [5]. Within the IoT context, tasks may be concretized as e.g., containerized microservices or serverless functions. Applications have a single point of entry (the initial task) and some sink (or final) task, signifying the result of the computation (i.e., the service composition). Within the IoT setting, those tasks may be deployed on different networked physical nodes. Network latency between edge nodes must be also taken into account, as it may affect the overall application execution time. More precisely, latency is understood as an adherence to certain defined Service Level Agreements (SLAs) and represents the time required for a message to traverse the application's communication flow (i.e., from the input data task to a sink task). As nodes may participate in multiple application deployments at the same time, they may thus contribute different resources of their own to each, perhaps based on some reward mechanism.

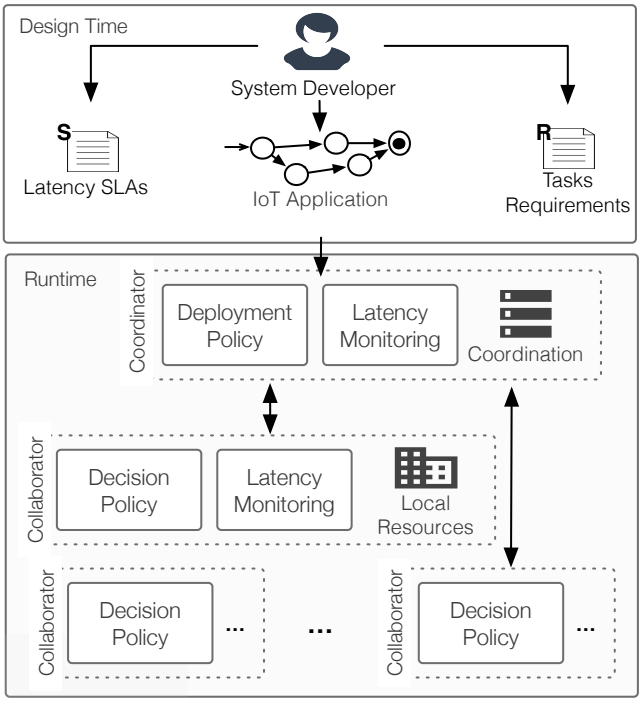


Fig. 1. Resource Management: From design time to runtime.

The functionality of the framework we advocate revolves around two key concepts; decentralization and node participation. To encourage edge nodes to participate and share resources with applications on the network, we assume that an incentive mechanism exists that offers rewards based on the involvement of a participant node. As such, the framework establishes collaboration between nodes to achieve the deployment goal. Two key components can be found in our solution: the *coordinator* node, which seeks to deploy some application and the *collaborator* nodes, which are the system participants. Any edge node can in principle take any of

those roles. Once a request for deployment of an application arrives, a list of participants is created. We consider some limited number of them, chosen based on some manifestation of proximity to the coordinator node. The coordinator serves as the local decision making authority, by advertising the IoT application to the nearby participants and eventually deciding a task distribution that satisfies latency, preferences of participants and other requirements. Collaborator nodes offer resources for parts of the application, at their own preference and based on their current state of available resources. Our framework provides seamless deployment of applications with latency requirements; as illustrated in Figure 1, the application designer defines the building blocks of an application (as tasks) as well as their dependencies in an application model, at design time – essentially the service composition. In practice, tasks are concretized as containerized microservices. When the system is operational, tasks are deployed to appropriate nodes so that requirements are satisfied, without requiring any knowledge of the current network topology.

Running example.

Consider a public safety application that aids law enforcement officers to identify wanted persons in a crowd by performing video analysis utilizing available resources found in their proximity; this is performed by processing video captured from the police officer's body camera, or on video and images stored in nearby edge nodes (i.e., other smartphones, tablets, dashboard cameras, etc.). Such video analysis is computation-intensive and may require specialized hardware running machine learning workloads, such as tensor processing units. The officer's smartphone (or dash cam) represents the application coordinator which is connected to his/her body camera. The application, illustrated in Figure 2, consists of multiple distinct services (abstracted as tasks) including (i) motion detection, (ii) object detection, (iii) object tracking, and (iv) result generation; arrows indicate the invocation of tasks within the application workflow.

Deploying such an application to a centralized location is not desirable due to its stringent requirements as well as inherent privacy concerns – video data should not be stored in a remote centralized location. First, we can observe that the decentralized nature of our technique fits well the application requirements since it handles video without sending it to a central facility for processing. Furthermore, the particular application is data-intensive, as vast amounts of generated data are analyzed. The centralization imposed by a cloud design has implications for both network congestion as well as latency, among others. Moreover, some tasks may require specialized device resources (e.g., the object detection task may require machine learning supporting hardware), making deployment on a single edge node which does not possess such capabilities infeasible. Considering this, deployment on nearby end-devices is advised – in this manner, computation and data management can be performed closer to the targeted area and distributed among participating nodes.

3 Problem Formulation

The objective of taking advantage of resources distributed among multiple interconnected nodes lies at the core of edge computing [14]. Application deployment implies utilization of these distributed resources. We consider an application modeled as a collection of tasks, each having a set of resource requirements that must be fulfilled under a set of application objectives. In this section, we outline our system model and the assumptions behind it, as well as the objectives that we consider.

3.1 Application and System Model

The edge computing setting we consider, is a distributed system consisting of multiple, possibly heterogeneous edge nodes. Nodes, with different architectures (ranging from mobile devices and single-board computers to powerful edge data centers), are capable of executing tasks and to communicate with others. Each edge node has a certain set of available resources $E_{res}=\{r_1, r_2, ...\}$ that can be shared within the network. Let $EN=\{E_1, E_2, ...\}$ be the group of nodes selected by the coordinator. Finally, we assume that nodes collaborate and share resources willingly. We point out the importance of suitable incentive mechanisms to provide rewards to nodes that share resources and behave cooperatively instead of competitively, and identify their development as future work.

To fully utilize nearby available computational resources, an application may be deployed on different edge nodes. We note that faithfully to SOA principles, a partition of application functionality into tasks is typical within distributed edge computing architectures, where the execution of an entire application may not fit on a single edge node [15]. For example, one could deploy the application described in the previous section, on a single node if there are enough available resources, but performance will be hindered. Moreover, considering that data of interest is distributed among multiple nodes, deploying the application on a single edge node is not feasible since data must be sent to a central point, increasing communication latency while failing to preserve tasks' requirements.

An application model is defined by the developer at design time and consists of a set of tasks $T=\{t_1, t_2, ...\}$, along with links representing communication flow. We assume that this flow starts with a source task which provides source data e.g., from an IoT sensor, and a sink, i.e., an actuator task, to take actions on the obtained results. More concretely, we assume that an application model is described by a direct acyclic graph (DAG), $G_{app} = (V, E)$, where vertices represent tasks and edges shows the links between them. Considering this, we can model our motivational example as shown in Figure 2. We abstain from application particulars such as how coordination occurs at the business logic level; our approach is concerned with finding a suitable deployment strategy across the edge network. Given the application model, our technique is agnostic about the inner workings of the deployed application.

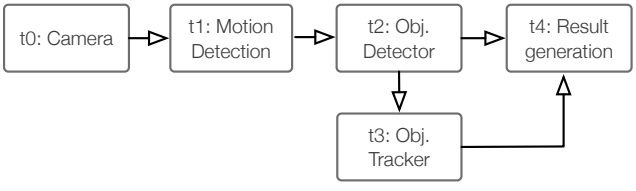


Fig. 2. Public Safety application model.

A task t_i represents a containerized microservice or serverless function, which implements a set of instructions that perform application business logic. Each task may require distinct computational or other resources. To this end, we assume that a task is defined by a set of resource requirements, $T_{req}=\{r_1, r_2, ...\}$, that may consist not only of generic memory, storage, and computational aspects but also could represent specific requirements like data, domain-specific hardware such as machine learning accelerators, sensors, or actuators. For example, a particular requirement, for the *motion detection microservice* of our example, besides computational resources, like *RAM, CPU and storage capacity*, is represented by the collected data from a specific area during a time frame.

A communication link between two edge nodes, E_1 and E_2, has an associated latency l_{E_1,E_2}; latency is inherited by tasks from their host node. For example, if t_1 is mapped on E_1 and t_2 on E_2, the communication latency of l_{t_1,t_2} is equal to l_{E_1,E_2}. Furthermore, the latency is computed as the sum of all communication delays between dependent tasks along the application's communication flow. Acknowledging the important role of latency, a latency monitoring function is imperative to the overall functionality. We consider this out of our scope as we work on a model level; we assume that latency is adequately measured and provided.

3.2 Objectives

Applications may be deployed across different connected nodes, making latency induced due to network and distribution a prime concern. A secondary concern highly pertinent to peer-to-peer systems, is edge node's resource preferences; participant nodes should be able to take decisions on how many (and how much of) resources to share and for what tasks, according to internal strategies defined by their administrative entity and possibly by other incentives. We treat those two concerns as key drivers, which must be satisfied upon deployment.

Our first objective targets one of the fundamental concerns of edge applications. We focus on a particular manifestation of latency, which is the e2e (end-to-end) delay of an application when operational. The e2e delay is defined by the duration of time required by an application to produce a result from received source data. For example, the e2e delay of our example application (Figure 2) captures the duration of time for t_4 to generate a result once t_1 collects data from its sensors. We assume that the desired e2e delay (as an SLA) for an application is defined at design time.

Our second objective is to respect resource preferences of participating nodes. Each node has authority on how its resources (including hardware or sensing capabilities) are shared with others – data that may reside locally are similarly treated. We achieve this behaviour by enabling edge nodes to take decisions locally, which guarantees the mapping of tasks where data or resources reside as dictated by participating nodes.

Note that a centralized solution where the coordinator resides in the cloud is generally and traditionally possible. However, we target decentralized edge-intensive systems, where (i) a connection to the cloud (for all participating nodes) cannot be assumed, and (ii) we seek to avoid the single point of failure that such an arrangement would introduce – in fact, any participating edge node (with or without a cloud connection) can serve the role of a coordinator.

4 Resource management Framework

An overview of our resource management technique is presented in Figure 3, showing the internal exchange between different modules as well as the communication between an application coordinator and a collaborator when an application is deployed. Two major components define our proposed technical framework– (i) *the deployment policy module*, implementing decentralized resource allocation functionality which aims to deploy an application without prior knowledge of edge nodes' available resources, and (ii) *the decision policy module*, where multiple decision strategies enable participant nodes to take local decisions regarding their current available resources.

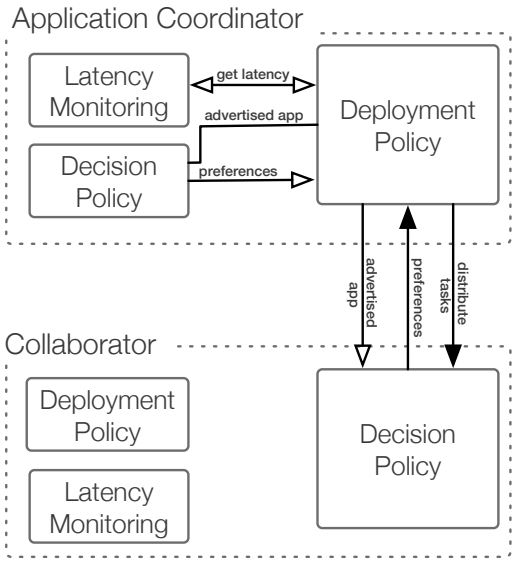


Fig. 3. Resource management: deployment & dataflow diagram.

4.1 Deployment policy module

We design the deployment policy module with the purpose of distributing tasks to a set of participants such that the overall application requirements are satisfied. Therefore, the functionality of this module captures the base capabilities of the coordinator node and consists of two different stages:

1) *Advertising stage*. Once an inquiry for application deployment is received, the application coordinator creates a message containing the tasks application model which is advertised to each participant node. The coordinator allocates a bounded time frame for receiving node's preferences; if during this period a node does not send its preferences, then the node is not considered for deployment.
2) *Deployment stage*. When all preferences arrive, the deployment stage starts. The coordinator finds a satisfiable allocation of tasks to participant nodes by considering the application requirements and participant preferences; a node preference can be partial or fully fulfilled.

We note that for complex resource allocation problems, research efforts in the field have proposed solutions based on metaheuristic optimization algorithms. As a result, a centralized solution where the coordinator is deployed in the cloud or on nodes with powerful computational resources is followed [16], [17]. Such solutions typically yield a near-optimal solution over a longer period of time. Our differences to these approaches are as follows. Firstly, our target domain is edge computing where (i) edge nodes may have limited computational resources, and (ii) latency and nodes' preferences are first-class concerns. Secondly, we aim for guarantees. Our approach provides a satisfiable solution that does not represent the near-optimal task allocation, but arguably more valuable in a dynamic network settings where the topology can change during execution rendering an optimized solution unnecessary. In metaheuristic approaches, there are no guarantees that the generated solution satisfies the objectives, especially if simulation time is limited. In contrast, we choose to work within Satisfiability Modulo Theories (SMT) [12], a generalization of the boolean satisfiability problem. By casting the problem within SMT, we provide guarantees that if a solution exists, it is found and it correctly satisfies the application requirements.

Notice that SMT fits our resource allocation problem particularly well; (i) the placement of tasks on nodes are essentially constraints over the space of deployment options, which can be encoded in first-order logic, and (ii) numerical latency SLAs can be encoded by integer linear arithmetic. Consequently, to solve our task allocation problem, we divide our encoding into four different parts, i.e., the *task facts*, the *domain facts*, *preferences constraints*, and *constraint formulation*. These capture different constraints over the desired solution, and are illustrated in the following.

Task Facts. Firstly, we encode the logical placement of a tasks. As a rule, a task t_i can be deployed on an edge node E_n only if it is part of the task preferences sent by that particular node. Furthermore, we ensure that in the placement solution exactly one task t_i is mapped on a node E_n. For example, recall the motivational example application of Section 2, composed of five tasks t_0, t_1, t_2, t_3, and t_4, on an edge architecture with two nodes, i.e., E_1 and E_2. Now, let us assume that the coordinator receives the following preferences: $P_1 = \{[t_3, t_4]\}$ from node E_1 and $P_2 = \{[t_1, t_2, t_3]\}$ is received from E_2. Based on the rules enforced by this encoding, each participant node can receive the tasks that are not common between P_1 and P_2. However, the common ones can be deployed only on one node, independent of how many nodes prefer to receive it. The general formula is shown in Formula 1, where n_T represents the total number of tasks. The semantics of *map()* is to provide a task allocation between t_i and one participant, where *participants* represents the set of nodes that preferred to share resources for that particular task.

$$\text{taskFacts}: \bigwedge_{i=1}^{n_T} (\exists! \, E : \text{map}(t_i = E)), \forall E \in \text{participants}. \quad (1)$$

Domain Facts. These capture the latency between two dependent tasks which are mapped on different nodes. The latency is found by giving an analogy between the task mapping derived from the *task facts* and their associated node latency. As a result, if a task t_1 is deployed on E_2 and t_2 is deployed on E_1, the communication latency between t_1 and t_2 is equal to the communication latency of the two edge nodes, i.e., E_1 and E_2. The general formula is shown in Formula 2, where l_{t_i,t_j} represents the latency between two tasks, while l_{E_i,E_j} represents the latency between two nodes, and n_{EN} represents the total number of participant nodes.

$$\begin{aligned} \text{domainFacts}: \bigwedge_{k=1}^{D} (t_i = E_i,\, t_j = E_j) \Rightarrow (l_{t_i,t_j} = l_{E_i,E_j}) \\ \text{for } D = \{i, j\} \text{ where } i \in [0,\, n_t] \text{ and } j \in [0,\, n_{EN}]. \end{aligned} \quad (2)$$

To ensure the correct functionality of our deployment technique and guarantee that the selected collaborator receives tasks that combined do not exceed its available resources, we must incorporate in our encoding multiple constraints for mapping tasks from the same P. Creating such an encoding is not trivial since each participant computes its preferences locally and the coordinator does not know any information about the node's available resources. As a result, the coordinator makes a decision based on the node preferences received. This functionality is defined in the *preferences constraints* part explained below.

Preferences Constraints. We encode task allocation constraints by defining a *preferences constraints* function. Its functionality entails creating a set of mapping rules to the tasks to the boundary of an individual group. As a consequence, only one group can be chosen from the preferences sent by a participant. Recalling our motivating example, a participant E_1 receives in the advertisement message the motivation example application model and based on its own decision strategies creates the following set of preferences $P = \{p_1, p_2, p_3, p_4\}$, where p_i contains a list of preferred tasks, e.g., $p_1 = [t_3, t_4]$, $p_2 = [t_1, t_2]$, $p_3 = [t_1, t_4]$ and $p_4 = [t_2, t_3]$. As a consequence, the coordinator can choose tasks from a single group to be mapped on a participant node; a group is obtained using a decision strategy, thus assuring that a group will not exceed the available resources of that particular node (i.e., the owner of P).

Let us consider that the coordinator chooses p_2 as the best group sent within P. In this case, choosing p_2 means that every other group from P cannot be satisfied, since it will exceed the available resources of that node. However, notice that there is a common task t_2 with p_4. Hence, we must guarantee that, if t_2 is mapped first on E_1, we do not block the tasks from p_2 or p_4 since with the current information, both offers can be chosen. Now, if t_1 is mapped on E_1, only then the remaining tasks from p_4 are blocked. In contrast, if t_3 is chosen, then p_2 is blocked. The general formula is shown in Formula 3, where *n* represents the total number of node's preferences received.

$$\begin{aligned} \text{prefConstaint}: \bigwedge_{i=1}^{n} ((p_1 \vee p_2 \vee p_3 \vee p_4) \\ \wedge (p_1 \implies !\,(p_2 \vee p_3 \vee p_4)....)). \end{aligned} \quad (3)$$

e2eConstraint. Finally, the last component of the encoding ensures that the deployment meets the latency SLA of the application; *e2eConstraint* captures rules that account for the latency in the e2e delay. To instrument a complete application, the developer should additionally account for its

execution overhead as well. As a result, Formula 4 ensures that the sum of the communication latency l_{t1t2} is less or equal than the total required SLA, where n_e represents the number of edges.

$$\text{e2eConstraint} : \sum_{i=1}^{n_e} l_i \leq \text{SLA}. \tag{4}$$

By combining the aforementioned constraints, we obtain the complete formula $\mathcal{F}$ used by the coordinator to find a satisfiable allocation according to application requirements:

$$\mathcal{F} : \text{taskFacts} \wedge \text{domainFacts} \wedge \text{prefConstraints} \wedge \text{e2eConstraint}. \tag{5}$$

Solving $\mathcal{F}$ incurs an energy cost, something which has to be accounted for since we target possibly resource constrained edge settings. The energy cost amounts to the execution of an SMT solver against the formula – later, we demonstrate that it is quite feasible to do so on single-board computers for relevant problems. The energy draw depends on the CPU power draw to solve $\mathcal{F}$ – full CPU usage for certain amount of time, depending on the problem size. Furthermore, executing the deployment policy also introduces a communication cost, for which we can calculate bounds for the exchanges required for each stage. In the *advertising stage*, the coordinator node sends a message to all participants and waits for their preferences. This stage requires a total of *2*m* messages, where *m* is the number of nodes. In the *deployment stage*, the coordinator informs only the nodes that will receive tasks, with a maximum of *m*.

4.2 Decision policy module

The decision policy module concerns strategies that a participant uses to create a set of preferences P for an advertised application. As previously mentioned, these strategies enable the collaborator to create groups of preferred tasks based on their own preferences and current internal state. As such, nodes' preferences are enforced since every decision is made locally without sharing information with other nodes in the network. Besides the feature of considering collaborator's task preferences, the strategies play a fundamental role in the overall functionality – they ensure coverage of tasks. Generally, to ensure that the application coordinator receives at least one preference for every advertised task, a consensus model is typically preferred where participant nodes communicate with each other to decide for what tasks to share their resources. However, in this scenario, there is an increased communication overhead and a node does not take decisions by itself; forcing the node to make compromises according to the preferences of other nodes. As such, in the following we outline some indicative strategies that participants may use to create P.

We especially note that the coordinator has no control over the participants' task preferences; in our conception, they are free to contribute (any) resources by sending tasks preferences of the advertised application. The rationale of giving participants *free rein* about their resource contribution to the system is as follows. Every participant may decide to adopt four default, indicative tactics to increase the number of groups sent in an instance of P. These intend to aim for greater coverage without requiring any information from other nodes. Each tactic has a different role in creating a group of preferred tasks. Hence, we conceptually group them based on their role in two different strategies.

Maximization Strategy. The first strategy aims to maximize the number of tasks, placed in a group (i.e., p_i), by utilizing all the available resources of a node, using two tactics. The first tactic is based on the well-known *0-1 knapsack* dynamic programming algorithm. We note that this fits well the context since it yields the near-optimal solution. However, although near-optimal, this tactic has high computational demands – as we focus on task allocation, we consider efficient tactic development as an interesting avenue of further investigation. An alternative can use heuristics to approximate knapsack-like solutions. The second tactic adopted in this strategy is based on a random selection of tasks. Notice that even though the overall strategy offers great coverage, these tactics do not consider dependencies between tasks.

Dependencies Strategy. The second strategy aims at creating more custom groups that take into account dependencies between tasks. For this reason, we employ two graph-theoretical algorithms as tactics, *strongly connected components* and *fan-out*. The former finds the largest strongly connected component into the application model and builds a group selecting tasks from the component until reaches the maximum available resources. In contrast, the latter selects tasks from the biggest edge fan-out found in the DAG.

It is important to stress that these strategies are indicative to participant nodes and are built to offer suitable node preferences for a wide range of applications. However, because a key driver of the approach is the collaborator's decision to contribute resources, the coordinator does not have any control over what decision strategies may be employed by participants. The above indicative strategies aim to bootstrap choices for a collaborator, which then can amend based on its internal resource sharing rules.

Each collaborator utilizes the strategies above to generate its task preferences within an advertised application. As we observed, the coordinator then proceeds to calculate a satisfiable mapping based on the technique of Sec. 4.1. Note that the preferences of a participant for certain tasks might be fully or partially satisfied, based on application-wide objectives. Energy costs for collaborators can be adjusted by selection of different strategies to capture the nodes' preferences. We especially note that different strategies would be interesting to develop in tandem with incentive mechanisms – in essence, to encourage participants to consider more tasks, something we identify as future work. Finally, the actual tasks have to be deployed in nodes. In practice, this entails downloading containers from a container repository. Costs arising from this are application dependent; size of containers comprising the application tasks and downlink bandwidth are key such factors.

5 Evaluation

To evaluate our technique and accompanying technical framework, we consider two evaluation goals; applicability and performance. For the former, we model four different applications obtained from literature to be deployed at the edge, while for the latter we follow a quantitative approach to evaluate the performance of our technique on

resource-constrained devices. To concretely support evaluation, we realized a prototypical tool based on the CVC4 SMT solver [18] – implementation and technical details can be found at [19] – which is deployed on a resource-constrained device, a single -board computer featuring an ARMv8 1.2GHz CPU, acting as the coordinator. After applicability aspects, we describe the experimental setup and finally discuss the obtained results.

5.1 Applicability: IoT Applications at the Edge

To investigate applicability, we model four applications, usually deployed on devices incapable of executing an entire application locally: (A1) an antivirus application, (A2) a face recognition application, (A3) the public safety motivational example of Section 2, and (A4) a team-building application. The first three applications are obtained from the literature and represent realistic applications – all three applications consists of 5 tasks and 5 edges [5] and have a maximum *SLA = 30*. In contrast, we adopt the fourth to demonstrate the framework's capability to deploy more complex applications, one that consists of 8 tasks and 10 edges with *SLA = 50*.

We consider that deployment may be intended for three different scenarios – offloading, mapping, and job assignment. Offloading refers to deployment of tasks of an application to nearby edge nodes to ensure better functionality and optimize e.g., energy consumption – such a practice is usually employed for mobile applications. The second scenario represents mapping of tasks permanently deployed on multiple nodes found at a certain locale, most useful in case of applications that do not feature mobile edge nodes, such as a smart traffic lights application [13] or do have mobile nodes but a self-adaptive technique ensures correct functionality after changes in the network. Finally, the third scenario considers applications that are instance based, meaning that a deployment occurs only when a request to do so is received. This captures utilization of resources of nearby edge nodes only for a limited period of time. Our technique is capable of deploying application for all three scenarios, however, we consider that, due to its nature, it will provide the most benefits in the context of task offloading; we can enable a resource-constrained device make decisions locally what tasks to offload and where, without the need of a central entity. We select two mobile applications (A1 and A2), one that fits the second scenario (A3) and one instance-based (A4), to be deployed on an edge system comprised of five nodes.

Each task and node is characterized by a set of computational resource requirements, i.e., a tuple (RAM, CPU, HDD, $\{OTHER\}$), where *OTHER* represents a set of special task requirements or special resources a node has, capturing their functionality and capabilities. The tasks which require no other specific resources (i.e., shown with $\emptyset$) can be mapped on any node if there are enough available resources. Furthermore, for every individual application model we randomly distribute on participant nodes a set of available computational resources. When selecting the node's available resources, we consider multiple factors, i.e., (i) the application's size, (ii) the total number of participant nodes, and (iii) the tasks' resource requirements. Note that for evaluation purposes, when choosing the available resources, we seek to create an environment that can host the deployed applications – at the same time, we seek to have limited available resources to challenge the framework. As a result, for applications A1-A3 we select for each resource a value between 5 to 10 units, while for application A4 we choose between 10 to 20 units, as application A4 contains more tasks. In addition, other resources required by the application are deployed on different nodes. For illustration purposes, we adopt a generalized 'unit' for resource quantification – in practice this would be refined per application (e.g., in MB/GB for RAM, GHz for CPU, etc.). Moreover, we assign a communication latency between nodes chosen randomly between 1 and 10 ms. For all application deployments, the coordinator is deployed on an ARMv8 R-Pi3 device featuring a 1.2GHz CPU and 1GB RAM, serving as the edge node. In contrast, the participant nodes are simulated on a machine with a single-core Intel i5 2.3GHz processor. Application models, further details, and evaluation results can be found in the online appendix [20].

Antivirus Application (A1). This application is a representative mobile application, widely used by users on their devices, that behaves like a software antivirus. The application requires: a graphical user interface (GUI) to interact with the user, i.e., task t_0, computational tasks like *scan file* and *compare* that represents the core functionality, and a task that present the output. Furthermore, some tasks may require special resources, e.g., task t_0 requires a set of files to be scanned, and t_5 requires a host node with a display. In Figure 4, one can observe the deployment strategy found, as well as the available resources of each edge node and the resource requirements of each allocated task. The successful mapping (indicated on Figure 4 with the dotted edge nodes) is found in *434 ms* with *SLA = 23*.

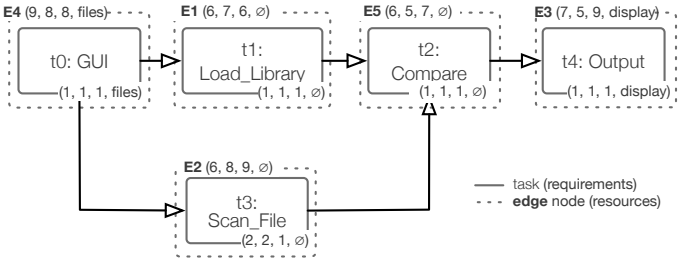


Fig. 4. Antivirus application and deployment (in overlay).

Facerecognizer Application (A2). This application is an image processing application able to identify a face in an image. Two tasks provide the interaction with the user, i.e., t_0 that requires a set of images as input and provides the application's GUI and task t_4 that presents the output on display. In contrast, the remaining tasks require specialized AI hardware such as a GPU or TPU. The requirements of each task and the available resource of participant nodes alongside the satisfiable solution found, is shown in Figure 5. In this casea deployment solution is found in *422 ms*, with *SLA = 22*.

Public Safety Application (A3). This application aids authorities in video analysis in the field. All tasks require special resources – task t_0, represents the source of video data, e.g., a CCTV camera, while task t_1 is capable of

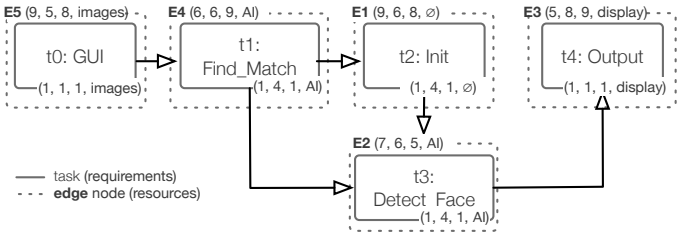


Fig. 5. Facerecognizer application and deployment (in overlay).

detecting motion and requires as input the raw video data received from t_0. Based on the output of t_1, the next two tasks, i.e., t_2 and t_3, detect and track objects – those utilize on machine learning algorithms requiring specialized hardware. Finally, t_4 requires a user-facing node with a display to present the generated results. Their resource requirements and available resources are shown in Figure 6. A satisfiable allocation is found in *407 ms* with *SLA = 17*.

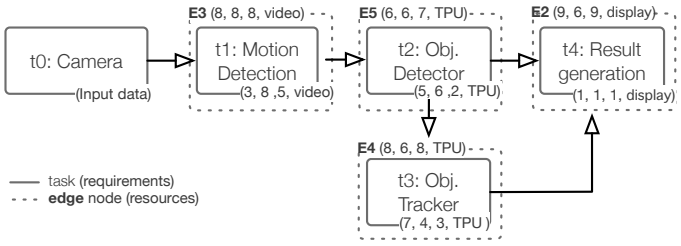


Fig. 6. Public safety application and deployment (in overlay).

Team Building Application (A4). The application aims to help companies finding the perfect team building location based on an analysis of user personal data. Each task resource requirements and the 5 collaborators available resources are shown in Figure 7); task t_0 represents the starting task and symbolizes the need for different types of data required by the next 4 tasks. The tasks that analyze the stored data are: t_1, which performs analysis of the employee's stored health information, t_2 requiring video data as input to perform motion detection, t_3 requiring images on which it performs object detection, and t_4 which does an analysis of the environment. Based on this information, t_5 generates a list of possible locations on which t_6 computes the budget required. Finally, t_7 represents the end task and creates a report suggesting possible destinations as well as an estimation of travel cost. Considering the defined experimental setup, a solution is found in *510 ms* with *SLA = 31*.

5.2 Performance: Experiments Setup and Results

To quantitatively evaluate our task allocation framework, we consider as a performance metric the execution time required to obtain a distribution of tasks to the collaborators. This amounts to the coordinator's ability to find a satisfiable solution at the edge, considering the highly computationally-demanding solving component of our technique. Furthermore, we perform an analysis of the SMT encoding by examining the number of symbols of each part of the encoding presented in Formula 5, i.e.,

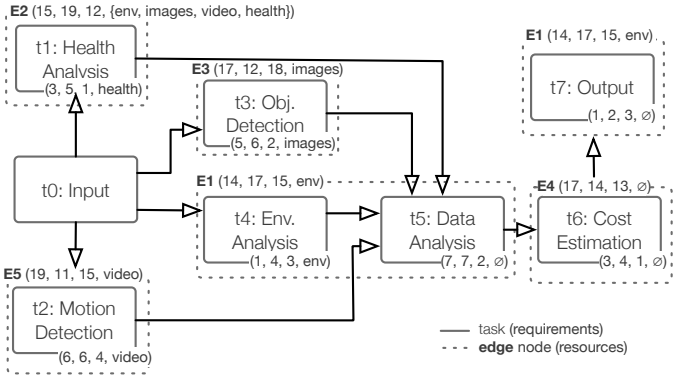


Fig. 7. Team building application and deployment (in overlay).

domainFacts, *taskFacts*, and *prefConstraints*. For this purpose, we design an experimental setup of an application and an edge computing setting.

For the application model, we adopt *montage* [21], a real-world DAG workflow. The application is composed of 24 tasks, each having allocated resource constraints in the range of 1 to 10 units, and 50 edges. Furthermore, to accurately evaluate performance and avoid discarding satisfiable solutions due to randomness in node distribution of resources and other limitations introduced by other factors like SLA and required data, we set the SLA to a large value and ignore data requirements of each task. The overall objective of our experiment setup is to map the application to an edge computing architecture in which we gradually increase the available resources (i.e., by increasing the size of the network). We randomly assign to each edge node a set of available resources chosen in the range of 10 to 20 units.

We adopt the same test environment used in the applicability scenario 5.1. We perform 500 tests for each newly created edge architecture, on which we measure: (i) the percent of successful mapping and exclusively at the edge mapping for different number of participating nodes, (ii) the time required by the coordinator to find a successful mapping of tasks to different size group of participants, and (iii) the number of symbols each part of the SMT encoding requires. Our results are presented in Figures 8, 9 and 10. In Figure 8 we observe that as the number of nodes increases, the successful mapping rate improves. This behaviour is similarly shown in Figure 9, where both the execution time and the number of symbols required by $\mathcal{F}$ increase with the number of participant nodes. Finally, in Figure 10, the relation between the total number of symbols of $\mathcal{F}$ and each individual components is presented.

5.3 Discussion

We have demonstrated that by using our framework, a decentralized resource management in an edge setting can be performed. Furthermore, we note that for all applications considered (Section 5.1), a successful task allocation at the edge (without resorting to cloud resources) is achieved in under *510 ms*, i.e., A1 is determined in *434 ms*, A2 in *422 ms*, A3 in *407 ms*, and A4 in *510 ms*. This demonstrates efficiency over different types of applications in under a second, a duration suitable for three scenarios involving

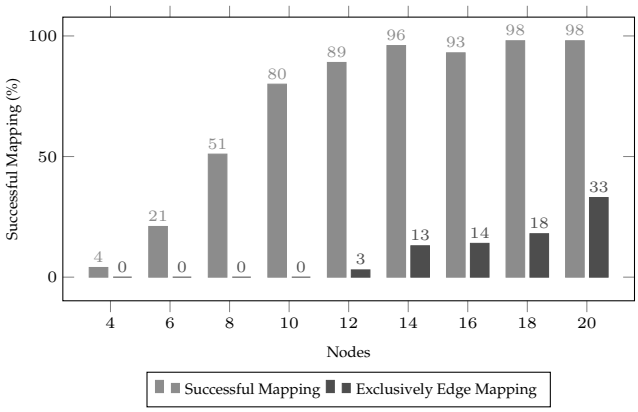


Fig. 8. Successful mapping over number of participant nodes.

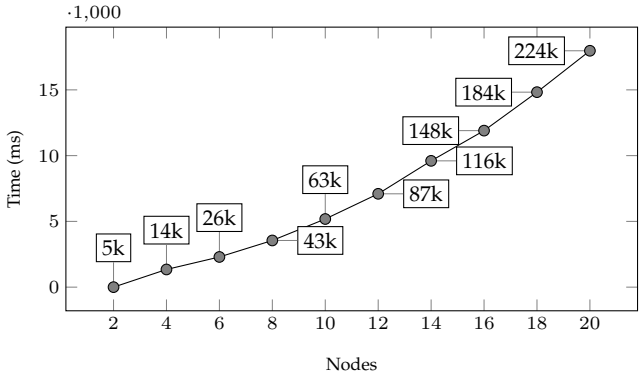


Fig. 9. Mapping time over number of participant nodes and formula size.

offloading, mapping and job assignment. Note that the time it takes to deploy an application is negligible for the overall application's lifecycle, because this happens once before it starts executing and not continuously, unless the network circumstances change; moreover, a deployment time of 500 ms is small compared to the overhead of moving tasks as containers.

Notice that our framework does not perform any optimization, but yields a task allocation strategy that satisfies the considered objectives. As a result, observe that the obtained SLA is not optimized. For example, for A1, it is possible to improve the SLA by mapping dependent tasks

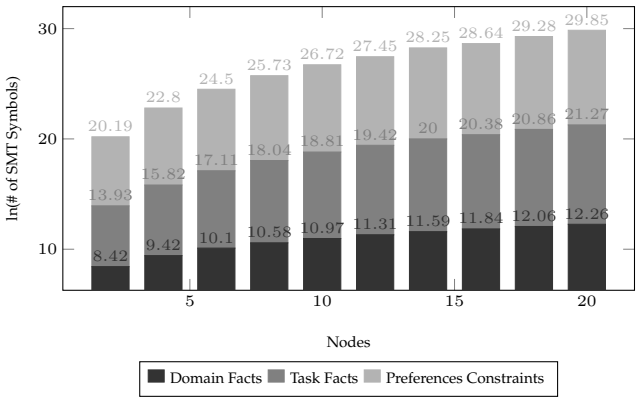


Fig. 10. Contribution in symbols of different problem components to the overall formulae, over increasing node count.

on the same node if there are available resources. In this case, some tasks like t_3 and t_4 can be mapped on the same node E_3, yielding a lower SLA. However, the SMT-based approach may find other solutions, in the case when application constraints are relaxed (and if they exist).

In Figure 8, we have recorded the successful rate of our framework to find an allocation of tasks at the edge with or without the need for computational resources found in the cloud. For each number of participants, we illustrate the total number of satisfiable solutions found and the number of solutions found using only edge resources. As one can observe, the total number of the former is influenced by the available resources shared between the participants. However, the decision strategies used by the participants have a bigger impact on the number of solutions fully mapped at the edge. As a result, to fully utilize the available resources found in an edge architecture, we must optimize the strategies to ensure greater task coverage.

Figure 9 illustrates the impact that an increase in the number of participant nodes has on the execution time required by the coordinator to yield a task allocation. In this case, we can observe that the growth of the number of nodes influences the SMT formula size which ultimately has an impact on the time required to find a solution. Besides the number of nodes and application size, other factors that influence the framework's performance are the e2e latency desired and the nodes' resources, albeit on a lesser scale. Moreover, since all nodes are considered reachable, their connection density, a concern in certain edge settings, has no impact on the framework's performance (Section 3.1). We note that with respect to previous work [9], the memory requirements of the SMT formulae produced are reduced – for instance, consider the reduction of a problem size of 20 nodes with respect to the SMT formula size, where a decrease of *12.5%* in the formula size is obtained. This increase in efficiency grows with the number of participating nodes (e.g., for 4 nodes we see a decrease of *6.67%*, while for 12 nodes a decrease of *12.12%*). Memory is a significant factor because resource-constrained devices typically have limited amounts. The encoding used in this paper represents scalability improvements over previous work [9].

We performed an analysis on the three parts of the encoding, i.e., *domain facts*, *task facts*, and *preferences constraints* to better understand which has the biggest impact on the overall SMT formula size, since their size increases with the number of nodes in the network. In Figure 10 the total number of symbols required for each part is illustrated, on which we apply the base e logarithmic function for presentation purposes. To have a better understanding of the number of symbols of each encoding, for the considered montage graph application of 24 tasks and an edge architecture of 20 nodes, the number of symbols of each is the following; the *domain facts* has a total number of *211K*, the *task facts* has *8K* symbols, and *preferences constraints* has a total of *5K*. Observe that encoding the latency objective in the formula is highly expensive (i.e., 90% of the total number of symbols) since the generated encoding is additive for every task's latency. To properly capture e2e delay, the formula requires all possible task mappings to nodes as well as their associated latency. As a result, the number of nodes and the application size have a greater impact compared to any other factor.

The framework can support any microservice-based IoT application that can be decomposed as a DAG. The first part of the evaluation (Section 5.1) demonstrates applicability on a diverse set of applications, intended to represent different scenarios obtained from the literature like offloading, mapping, and task assignment. These applications have different characteristics such as variable number of tasks and communication links. Furthermore, to quantitatively assess the effect of the major computational-demanding factors – application's size, node's available resources, and number of nodes – we considered a real-world workflow, intended to stress the framework (Section 5.2). We believe this shows that results are generalizable, even in large applications – the extent of which is demonstrated on the time it takes to find deployment strategies. As such, we believe that the framework performance is acceptable for relevant problems – considering that for the experiments performed the coordinator resides on a single-board computer.

We acknowledge the high computational demands of our proposed technique, but we note that it offers guarantees – something demonstrated in the scenarios A1-A4 presented. Results show that the framework can successfully deploy, in absence of cloud resources, realistic applications with distributed resources at the edge of the network. Scaling up to higher numbers of participants is hindered by the sizable encoding of the e2e delay objective (Figure 10). If the focus is on other objectives that scale linearly with the fan-out degree of a service (such as bottleneck avoidance), the overall performance will improve significantly. As a result, we conclude that the proposed technique will perform best in scenarios where only a slice of the entire network is considered for an application deployment (e.g., a disaster scenario or task offloading from a resource-constrained device) or an extra module is introduced that is capable of creating a group of participants, selecting them from the proximity of the coordinator node. Finally, we note that the technique does not guarantee the best latency, only one that is less than the latency requirement specified. However, the deployment solution is guaranteed to be correct (by virtue of SMT), and is obtained rather fast.

6 Related work

Resource management techniques have been sought by the community focusing on different aspects of the problem. Accordingly, we discuss related work from three main perspectives; first, as resource offloading, next as resource allocation, and finally, as resource auctioning.

6.1 Resource offloading

Recent novel directions in distributed systems have demonstrated advanced resource management techniques to offload parts of an application from resource-constrained devices like smartphones, to nearby edge servers (mobile edge computing (MEC) nodes). By offloading tasks, computational demanding applications can run on end-user devices.

A low-latency distributed computation offloading technique aiming at distributing tasks in a pervasive system is proposed in [22]. By combining serverless and edge computing, a fully distributed domain was created consisting of three different entities: (i) clients who wish to offload tasks, (ii) dispatchers who are in charge of distributing the incoming tasks to a group of computers, and (iii) computers which provides the computational resources. The system is divided into two main categories: an online phase where the dispatcher decided the distribution of all incoming task and an offline phase where important functions, like setup of containers or dispatcher configuration are performed. An offloading model based on a heuristic approach is presented in [5], considering parameters like application runtime, battery lifetime, and user cost as its objectives. The offloading technique is composed of three different components, i.e., an application profiler which takes as input the running mobile application and converts it to a DAG structure, monitoring of the remote infrastructure status, and taking an offloading decision based on the input received from the first two components and the user preferences. As a result, a task can be executed either locally or on an edge/cloud node.

6.2 Resource allocation

Besides the offloading area, resource allocation techniques are adopted to use the available resources found on nearby edge devices. With this purpose, a competitive-cooperative game-theoretic resource allocation framework to deploy latency-sensitive application at the edge is developed, ensuring cooperation between nodes by offering incentives based on their work [23]. In this case, edge devices are considered to be rational actors that have no desire of sharing their resources and collaborate with other nodes if proper payoff for their services is not offered.

In [24], a three-level resource allocation technique for edge computing is proposed. The first level optimizes the distribution of data replicas on multiple devices to minimize the execution of a task at the cost of increased data management. Both, the number of replicas and the data allocation is decided based on a context-aware replication technique that accounts for data size, current fluctuation of the system, the available storage, and application characteristics. Next, the second level distributes the tasks based on different scheduling strategies, while the third level monitors the task deployment and adapts data placement if needed. A task assignment technique for data shared MECs is presented in [25]. The solution considers the distributed data found at the edge of the network when tasks are deployed closer to the end-user. To efficiently deploy the tasks, the authors grouped them into two different categories: (i) holistic tasks that cannot avoid raw data transmissions and (ii) divisible tasks that can be processed in a distributed manner.

In [26] a cooperative fog platform ensuring a collaboration between multiple static and mobile fog devices by using a distributed communication model is described. Moreover, to improve the service efficiency of IoT applications, an allocation algorithm is used to select the host based on the characteristics of the system. A similar approach of a mapping algorithm composed of two stages is proposed in [27]. First, the application is divided into multiple different tasks annotated with location information; in the second stage each task is deployed on an edge node based on its location.

6.3 Resource auctioning

With regards to resource auctioning techniques, in scientific literature we can find several proposals that focus on

offloading tasks to edge devices. In [28], an auction-based mechanism to perform resource allocation to MECs and compute the related price for each resource is proposed. The main idea is to have mobile users compete for resources available at the edge servers to execute IoT applications. Once a user submits a request to the nearest edge server, a pricing mechanism computes the price for that particular resource both in the cloud and edge. Based on this, the user can decide if the deployment of their application to the edge benefits them taking into account the cost.

An auction-based solution where users bid for resources and edge servers sell their own for a certain price is described in [29]. For this purpose, a model of a two-sided interaction between a MEC server which plays the role of the seller and the bidders representing the IoT mobile devices is developed. A double auction scheme is used to efficiently map computational resources of a MEC server to the needs of a mobile device by determining the matchmaking between the bidders and sellers. Another auction mechanism for determining the optimal content placement, on mobile edge devices, based on user's bids is presented in [30], where a mechanism that can find true valuations from the users and promote participation.

A reverse auction which considers partial fulfillment of tasks, as well as attribute and price diversity, is proposed in [31]. A framework where each task owner is in charge of hosting his/her auction without the need of collecting global information. The authors offer two different auction schemes, i.e., the cost-preferred auction that schedules tasks according to users' asking price and time scheduled-preferred auction that considers their arrival time. A distributed auctioneer for resource allocations on distributed systems is proposed in [32]. A set of distributed protocols to be shared between multiple participants with the intent to simulate a centralized auctioneer. The main motivation is device trust and their operators' true intentions. By doing a decentralized auctioneer the problem of trust disappears since all operators have a say in how the resources are distributed; eliminating the need of different participant nodes to get an unfair advantage. The solution considers both theoretical and practical implications of a decentralized auctioneer, by using game theoretical perspective as well as limiting the communication overhead.

Overall, different service placement approaches adopt different methods – from heuristic algorithms to ILP, DNN and others. Both the objectives and edge settings of those approaches differ (e.g., optimizing for latency, cost, or resource utilization). The setting that we treat are dynamic edge systems as well as meeting nodes' preferences and providing guarantees. Specifically, our framework differs from three perspectives; (1) we guarantee that if there is a solution possible, it is always found, (2) we maintain device preferences by enabling local decisions, and (3) we perform resource management in a decentralized manner, on resource-constrained devices without requiring knowledge about resources of participants.

7 Conclusion

Taking advantage of available resources closer to end-devices in a service-oriented fashion calls for novel resource management techniques that comply with latency, consideration of nodes' preferences and decentralization demands of contemporary IoT applications. We proposed a novel decentralized resource management technique and accompanying technical framework for deployment of latency-sensitive applications on the cloud-fog-edge continuum; our application coordinator being able to reside on any of the three layers. We specifically focused on deploying applications in the absence of cloud resources, where the coordinator is deployed on a resource-constrained device. We demonstrated that our technique can efficiently utilize available resources at the edge and provide guarantees – if a solution that satisfies latency and task's requirements exists at the edge, it will be found and it will be correct.

Regarding future work, we plan to investigate decision strategies that ensure better coverage with lower computational demands. Furthermore, we intend to incorporate an incentive system to reward participant nodes for sharing their resources, perhaps by enticing them to use strategies particularly efficient to the collective. An extension of the proposed technique may be desired to consider deployment of service models with multiple input and output tasks (i.e., where the invocation or execution path in the service composition has multiple entry and exit points). In this case, each individual execution path may have a certain e2e delay constraint set, yielding a further constraint that task allocation must consider. Similar to the notion of considering resource preferences of edge nodes, security and privacy mechanisms [33] can be further integrated [34]. Finally, to account for dynamic behaviour at runtime, we aim to incorporate task migration techniques.

Acknowledgments

Research leading to these results 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 and the FWF Austria project M 2778-N "EDENSPACE".

References

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

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. June, pp. 30–39, Jan 2017.

[3] F. Wagner, F. Ishikawa, and S. Honiden, "Robust service compositions with functional and location diversity," *IEEE Transactions on Services Computing*, vol. 9, no. 2, pp. 277–290, 2016.

[4] K. Toczé and S. Nadjm-Tehrani, "A taxonomy for management and optimization of multiple resources in edge computing," *CoRR*, vol. abs/1801.05610, 2018.

[5] 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.

[6] J. Ren, H. Guo, C. Xu, and Y. Zhang, "Serving at the edge: A scalable iot architecture based on transparent computing," *IEEE Network*, vol. 31, no. 5, pp. 96–105, 2017.

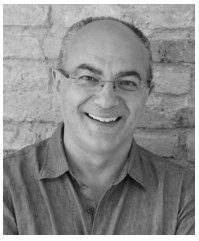
[7] C. Tsigkanos, I. Murturi, and S. Dustdar, "Dependable resource coordination on the edge at runtime," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1520–1536, 2019.

[8] C. Avasalcai and S. Dustdar, "Latency-aware distributed resource provisioning for deploying iot applications at the edge of the network," in *Advances in Information and Communication*, K. Arai and R. Bhatia, Eds. Cham: Springer International Publishing, 2019, pp. 377–391.

[9] C. Avasalcai, C. Tsigkanos, and S. Dustdar, "Decentralized resource auctioning for latency-sensitive edge computing," in *IEEE International Conference on Edge Computing (EDGE)*, 2019.
[10] P. Wang and X. Du, "Qos-aware service selection using an incentive mechanism," *IEEE Transactions on Services Computing*, vol. 12, no. 2, pp. 262–275, 2019.
[11] O. Scekic, H.-L. Truong, and S. Dustdar, "Pringl – a domain-specific language for incentive management in crowdsourcing," *Computer Networks*, vol. 90, pp. 14 – 33, 2015.
[12] C. Barrett and C. Tinelli, *Satisfiability Modulo Theories*. Cham: Springer International Publishing, 2018, pp. 305–343.
[13] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*. Cham: Springer International Publishing, 2014, pp. 169–186.
[14] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.
[15] M. Deng, H. Tian, and B. Fan, "Fine-granularity based application offloading policy in cloud-enhanced small cell networks," in *IEEE Intl. Conf. on Communications Workshops*, May 2016, pp. 638–643.
[16] M. M. Shurman and M. K. Aljarah, "Collaborative execution of distributed mobile and iot applications running at the edge," in *Intl. Conf. on Electrical and Computing Technologies and Applications*, Nov 2017, pp. 1–5.
[17] M. Aazam and E. N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *IEEE Intl. Conf. on Pervasive Computing and Communication Workshops*, March 2015, pp. 105–110.
[18] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "Cvc4," in *Computer Aided Verification*, G. Gopalakrishnan and S. Qadeer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 171–177.
[19] "Decentralized resource management framework," https://github.com/cavasalcai/Decentralized-Resource-Management.git.
[20] "Accompanying models and evaluation artifacts," https://dsg.tuwien.ac.at/team/cavasalcai/projects/ResourceManagement.
[21] 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.
[22] C. Cicconetti, M. Conti, and A. Passarella, "Low-latency distributed computation offloading for pervasive environments," in *Pervasive Computing and Communications (PerCom)*, 2019.
[23] D. Zhang, Y. Ma, C. Zheng *et al.*, "Cooperative-competitive task allocation in edge computing for delay-sensitive social sensing," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, Oct 2018.
[24] M. Breitbach, D. Schäfer, J. Edinger, and C. Becker, "Context-aware data and task placement in edge computing environments," in *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom). IEEE*, 2019.
[25] S. Cheng, Z. Chen, J. Li, and H. Gao, "Task assignment algorithms in data shared mobile edge computing systems," in *2019 IEEE 39th International Conference on Distributed Computing Systems*, July 2019, pp. 997–1006.
[26] A. Kapsalis, P. Kasnesis, I. S. Venieris *et al.*, "A cooperative fog approach for effective workload balancing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 36–45, March 2017.
[27] R. Jain and S. Tata, "Cloud to edge: Distributed deployment of process-aware iot applications," in *IEEE International Conference on Edge Computing*, June 2017, pp. 182–189.
[28] 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.
[29] 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.
[30] X. Cao, J. Zhang, and H. V. Poor, "An optimal auction mechanism for mobile edge caching," in *2018 IEEE 38th International Conference on Distributed Computing Systems*, July 2018, pp. 388–399.
[31] Z. Duan, W. Li, and Z. Cai, "Distributed auctions for task assignment and scheduling in mobile crowdsensing systems," in *IEEE 37th International Conference on Distributed Computing Systems*, June 2017, pp. 635–644.
[32] A. M. Khan, X. Vilaça, L. Rodrigues *et al.*, "A distributed auctioneer for resource allocation in decentralized systems," in *IEEE 36th International Conference on Distributed Computing Systems*, June 2016, pp. 201–210.
[33] N. Li, C. Tsigkanos, Z. Jin, S. Dustdar, Z. Hu, and C. Ghezzi, "Poet: Privacy on the edge with bidirectional data transformations," in *2019 IEEE International Conference on Pervasive Computing and Communications, PerCom 2019, Kyoto, Japan, March 11-15, 2019*. IEEE, 2019.
[34] C. Tsigkanos, C. Avasalcai, and S. Dustdar, "Architectural considerations for privacy on the edge," *IEEE Internet Computing*, vol. 23, no. 4, pp. 76–83, 2019.

Cosmin Avasalcai is research scientist and PhD student working under the supervision of prof. Schahram Dustdar at the Distibuted Systems Group at TU Vienna. After finishing his bachelor studies at Technical University "Gheorghe Asachi" Iasi, Romania in 2015, he received a master's degree in Computer Science and Engineering from Technical University of Denmark (DTU) in 2017. Before joining the Distributed Systems Group, Cosmin worked for two and a half years as a research assistant at DTU, developing modeling and decision making tools for the automotive industry for mixed-criticality applications in dynamic and changeable real-time environments and the Oil & Gas industry for structural monitoring. His current activities include quality of service-aware resource provisioning in edge computing within the Fog Computing for Robotics and Industrial Automation European Training Network project.

Christos Tsigkanos is Lise Meitner Fellow at TU Vienna (Austria). Formerly, he was postdoctoral researcher at the Distributed Systems Group at TU Vienna and at Politecnico di Milano (Italy), where he received (2017) his PhD defending a thesis entitled "Modelling and Verification of Evolving Cyber-Physical Spaces". He holds a BSc degree in computer science from University of Athens (Greece) and a MSc degree in software engineering from University of Amsterdam (the Netherlands). His research interests lie in the intersection of distributed systems and software engineering, and include dependable self-adaptive and cyber-physical systems, requirements engineering and formal verification.



Schahram Dustdar is Professor of Computer Science with the Distributed Systems Group, TU Vienna, Austria and an IEEE Fellow. He was Honorary Professor of Information Systems at University of Groningen, The Netherlands (2004-2010), Visiting Professor at the University of Sevilla, Spain (2016-2017) and Visiting Professor at the University of California at Berkeley, USA (2017). Dustdar is an elected member of the Academia Europaea, where he is Chairman of the Informatics Section. He was recipient of the ACM Distinguished Scientist Award (2009), the IBM Faculty Award (2012), and the IEEE TCSVC Outstanding Leadership Award for outstanding leadership in services computing (2018). He is the Co-Editor-in-Chief of the ACM Transactions on Internet of Things and the Editor-in-Chief of Computing (Springer). He is also Associate Editor of the IEEE Tranactions on Services Computing, the IEEE Transactions on Cloud Computing, the ACM Transactions on the Web, and the ACM Transactions on Internet Technology. He serves on the Editorial Board of IEEE Internet Computing and the IEEE Computer Magazine.