A Decentralized Approach for Determining Configurator Placement in Dynamic Edge Networks

Ilir Murturi
Distributed Systems Group
TU Wien
Vienna, Austria

Mohammadreza Barzegaran
DTU Compute
DTU
Kongens Lyngby, Denmark

Schahram Dustdar
Distributed Systems Group
TU Wien
Vienna, Austria

***Abstract*—In today's IoT infrastructures, increasingly newly added computational resources at the edge of a network are added to acquire faster response and increased privacy. Such edge networks bring an opportunity for deploying edge application services in proximity to IoT domains and the end-users. In this paper, we consider the problem of utilizing various computational resources established by multiple heterogeneous edge devices in dynamic edge networks. A new lightweight decentralized mechanism (i.e., *configurator*) is required to monitor an edge infrastructure to enable deploying, orchestrating, and monitoring edge applications at the edge. In this setting, one critical task is to determine the node where the configurator should be placed (deployed) and run (executed) at the edge. In this paper, we propose an efficient approach that solves the configurator's placement problem on the most suited edge device in a given dynamic edge network. Our approach supports the system coping with the dynamicity and uncertainty of the environment and adapts based on the configurator's service quality. We discuss the architecture, processes of the approach, and the simulations we conducted to validate its feasibility.**

***Index Terms*—Edge Computing, Internet of Things, Decentralized, Resource Management**

I. Introduction

The emergence of Edge computing has introduced edge devices as an intermediary entity between applications and the Internet of Things (IoT) deployments, providing data or control facilities to the participating IoT devices. The Edge computing paradigm enables to shift processing from the cloud to the edge devices located at the logical extremes of a network - close to the user [1]. Accordingly, these intermediate devices take responsibility for processing data promising to satisfy the stringent requirements prevalent in IoT systems, including high availability, performance, and privacy [2]. However, edge devices are usually considered resource-constrained with limited resources, referring to their different computational capabilities, including storage or processing facilities. For instance, providing a service for image processing or deploying edge applications (i.e., IoT applications) on a single edge device poses many limitations and a set of challenges in terms of processing capabilities, storage, and communication bandwidth. To this end, edge devices do not exist in isolation and must be able to collaborate with other edge devices. Thereby, interaction with other edge devices enables extending the scope of available resources and satisfying the computational requirements of real-time edge applications at the edge of the network.

To overcome these shortcomings, edge devices found nearby can form an edge-to-edge network, respectively, an edge overlay network. Edge-to-edge collaboration provides many benefits. First, edge devices can exchange information about available resources within their scope in a peer-to-peer (P2P) manner. For example, several edge devices deployed in a smart neighborhood will share their functionality descriptions based on their privacy preferences. Thereby, each edge device will be able to utilize available resources at the edge of the network [3]. Second, multiple edge devices provide a seamless opportunity to enable deploying edge applications at the edge. For example, an edge application divided into multiple *services* can be mapped in the edge network where participating nodes may run a particular service and satisfy application requirements [4]. Thereby, forming such edge networks enable relieving complex processings by distributing application services among available edges. However, such edge networks are heterogeneous and very volatile environments. This introduces new challenges where edge devices may require to migrate application services to other edge devices to fulfill continuously changing application demands. Hence, deploying applications in such heterogeneous infrastructures require novel techniques for resource management at edge.

A common approach for resource allocation in edge infrastructures is to assign services to the available edge devices by considering several factors such as processing capability, bandwidth, or energy. Such techniques often employ centralized architecture where a static edge device is a *master device* that acts as a gateway, monitors resources, and runs scheduling algorithms for generating deployment strategies. Similarly, several approaches have been proposed to enable application placement in edge networks [5], [6]. However, many research papers determine the master device statically. They do not address issues when the Quality of Service (QoS) between the master device and the other client nodes is degraded due to the high utilization or high end-to-end latency. In contrast to the mentioned works, a distributed approach [7] inspired by the functionality of an auction house has been proposed to enable IoT application deployment at the edge. However, the proposed solution faces latency issues, and it considers a limited number of nodes in the topology.

In practice, statically placing a set of functionalities (e.g., planning, controlling, or monitoring resources) on a single de-

vice may be feasible in small and non-dynamic edge networks. However, in dynamic and large-scale edge networks, such an assumption is rarely accurate and may result in inefficient resource utilization and network overheads. This often occurs due to the dynamicity of edge networks, which may change continuously over time. Such changes are caused by unexpected node joining/leaving, high utilization, or node and link failures. Thus, to ensure efficient deployment and orchestration of edge applications (e.g., elasticity and migrating services), we require real-time controlling and monitoring of the edge infrastructures (i.e., node hardware values). Hence, to overcome these challenges, we need a decentralized mechanism that acts as a *resource manager* and a *control mechanism* closer to the edge. Such a decentralized mechanism enables deploying and orchestrating applications and monitoring infrastructure at dynamic edge networks. From now onwards, we refer to this mechanism as the *configurator* (explained in Section II-C).

In this setting, a critical task is to determine the node where the *configurator* should be deployed and executed at the edge. Therefore, in this paper, we propose an efficient decentralized approach that identifies the most suitable node to run the configurator in dynamic edge networks (Section II-B). Our approach consists of an architecture and processes that enable the placement of the configurator. The proposed method allows the execution of the configurator on the edge device with the highest computing performance, the lowest workload, and the best overall bandwidth. An edge device that executes the configurator mechanism becomes a configurator node. Notably, when the configurator node is overwhelmed, a custom event is triggered to find another suitable node that can handle the current workload. To validate the feasibility and scalability of the approach, we have implemented a prototype and simulate the configurator's placement at the edge.

The rest of the paper is structured as follows. Section II gives an overview of our Edge-Cloud ecosystem, the configurator, and motivation. Related work is considered in Section III. Section IV describes the architecture modeling, resource utilization modeling, and the makespan to run and transfer the configurator at the edge. Section V describes in detail the proposed algorithms in charge of determining configurator in manager mode in the edge network. Section VI provides the simulation results to evaluate the proposed solution. Finally, Section VII concludes the paper and outlines future work directions.

II. Background and Motivation

In this section, we first introduce the background of the Edge-Cloud ecosystem and the edge neighborhood. Then, we give a short overview of the configurator's aim in edge networks. Finally, we present our motivation scenario. Note that throughout the paper, we may use interchangeable notations for edge networks such as edge neighborhoods.

A. Edge-Cloud Ecosystem

We consider our proposed edge ecosystem [3], [8], which is composed of a three-tier layer architecture: edge, fog, and cloud, respectively. The proposed ecosystem aims to enable collaboration between edge devices, information, and people to create an IoT platform that supports the development of new edge applications. Figure 1 shows three types of collaboration in our proposed ecosystem: edge-to-edge, edge-to-fog, and cloud-to-fog collaboration. The edge layer represents the internal environment where several edge devices connect and form an edge neighborhood. In this layer, each such device contains various resources that could aid in building different edge applications.

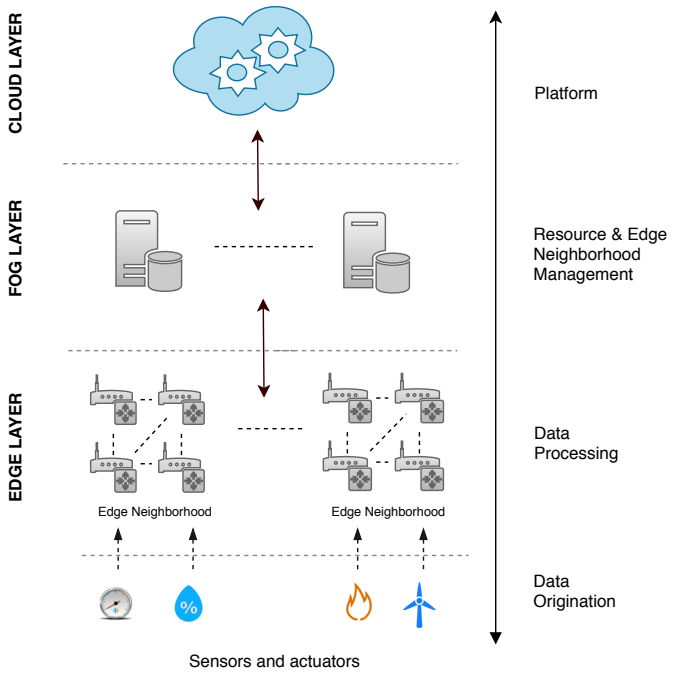

Fig. 1. An overview of the Edge-Cloud ecosystem.

The network's upper layer represents the external environment where several fog nodes are connected, offering computation and storage resources for edge neighborhoods. In the edge-to-fog scenario, an edge device that executes the configurator enables the communication between the internal network (edge neighborhood) and external network (fog network). Whenever there are not enough resources at the edge neighborhood, the configurator node may request to use the fog network's external resources.

The third layer of the network represents the platform's environment, and edge applications can be downloaded and deployed in edge neighborhoods. Additionally, the third layer serves as an environment where heavy tasks that cannot be computed at the edge or fog are moved to the cloud for further processing. In this paper, we aim to solve the challenges introduced at the proposed ecosystem's edge layer.

B. Edge Neighborhood

A typical Edge computing system includes heterogeneous, resource-constrained, and geographically distributed comput-

ing resources [9]. Several approaches have been proposed aiming to build and organize computation nodes in the edge network [10]. To meet our desired objectives, we consider a similar organization type of the edge network proposed in [11]. In the mentioned paper, nodes are organized in clusters, and the network can easily scale. Moreover, the maximum cluster sizes are configurable and the number of clusters to which one node may belong. We assume that each cluster has a limited number of nodes in our system, and each node belongs to only one cluster. We limit the number of nodes in clusters in order to handle the complexity introduced when monitoring nodes in the edge infrastructure [12]. It is worth noting that to orchestrate edge applications at the edge, we require to monitor hardware resources at different edge nodes or their end-to-end latency between them. In Figure 2, we present an example of an edge neighborhood comprised of twenty nodes organized in three clusters.

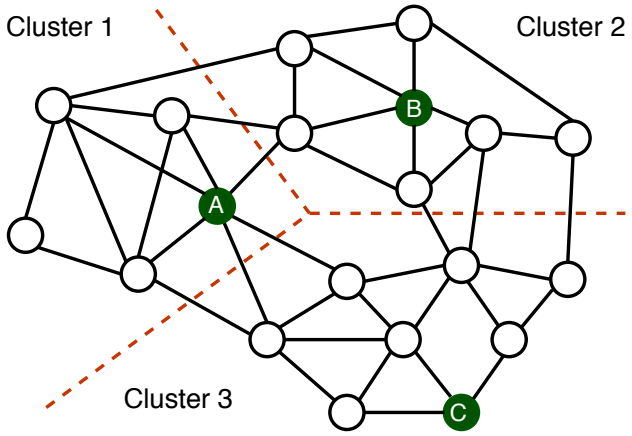

Fig. 2. An example of an edge neighborhood comprised of twenty nodes organized in three clusters.

In Figure 2, each cluster have a leader node (e.g., node A, node B, node C). We assume that leaders act as superpeers [13], and each of them stores contact details (i.e., IP address, etc.) of the other leaders. Leaders may store some other information regarding the other nodes in the neighborhood. Similarly, edge nodes that belong to the same cluster store information for each other and know their cluster leader and the configurator node at any time. In Section V-A, we explain in detail the process of finding the cluster leader. Note that we treat an edge neighborhood as already given, and to build the edge network is out of the scope of this paper. Our focus is on determining the most suitable node to place and execute the configurator on heterogeneous and dynamic edge infrastructures. Therefore, issues related to the edge network, such as joining/leaving nodes, organizing nodes in clusters, and operational aspects are orthogonal to our approach. In future work, similar to the method mentioned above, we plan to build the edge neighborhood based on the Kademlia Protocol [14].

C. The Configurator

To enable application deployment in a decentralized manner, each edge device will require information regarding an edge infrastructure's current condition. However, it is computationally demanding to monitor resources from each edge device throughout the network. Therefore, in this paper, we provide a solution to determine which node must take the responsibility to act as a *resource manager* as well as a *control mechanism* for controlling deployment aspects and orchestrating applications at the edge. We refer to this mechanism as the *configurator*, and the method proposed in this paper is one of the main components of the configurator. The configurator is a decentralized mechanism which is a lightweight software application aiming to enable deploying, orchestrating edge applications (i.e., elasticity, migration), and monitoring resources at the edge. To achieve such objectives in a decentralized manner, we conclude that each edge device needs to have a set of functionalities embedded through a configurator. Therefore, we define two operation modes of the configurator: i) *manager mode* (i.e., resource manager and control mechanism) and ii) *edge mode* (i.e., worker node).

In an edge neighborhood, there is one and only one configurator running in the manager mode. The configurator node instantiates a *local edge agent* on each cluster leader nodes to enable monitoring resources and orchestrating applications locally. Such a local agent provides information to the configurator node regarding the available resources on its cluster. Once the user requests installing an edge application (i.e., downloaded from the cloud), the configurator node determines which cluster meets the application requirements and deploys it in a specific group (if the cluster is not specified beforehand). Additionally, the configurator helps to orchestrate applications between clusters, fog, and cloud whenever they cannot scale locally due to the limited resources. Nonetheless, it remains the future work to provide a complete solution for the configurator and a full solution stack for edge applications that are dynamically distributed, elastic, resilient, and run natively in the Edge–Cloud continuum.

D. Motivation

We envision a smart city scenario where city administrators can build and customize edge networks by deploying various services. In this scenario, the edge infrastructure is structured based on the districts in the city. Each district represents an edge neighborhood where thousand of computation nodes are deployed as well as sensors, actuators, and mobile devices. An edge neighborhood can be composed of multiple groups of nodes (i.e., each group may represent a neighborhood in the district). This brings an opportunity to customize environments depending on the available resources, e.g., if there are sensors for gathering air quality data in a particular neighborhood, a specific monitoring service can be deployed. For example, in a particular neighborhood, residents may complain about noise pollution. Thereby, city administrators may add new sensors connected to edge devices for gathering real-time data about the noise pollution in the affected area. Afterward, an edge

application can be deployed closer as possible to the data source enabling monitoring noise pollution, processing data, and notifying the relevant authorities if noise exceeds the upper bound limit [15]. A situation may arise when the application QoS requirements are violated (e.g., node failures, overloaded devices, high end-to-end latency, etc.). Thus, the application services need to be scaled/moved in other nodes. Notably, controlling and monitoring mechanisms in such environments remains a challenging task to implement. These issues arise due to excessive peculiarities of the edge network (i.e., heterogeneity, unavailability, and limited resources). Thus, it is evident that we require proper techniques that enable orchestrating, monitoring applications, and monitoring edge infrastructure in a decentralized and dynamic manner. Our proposed approach aims to shift such functionalities closer to the edge and dynamically placing them in the most suitable nodes.

III. Related work

Current literature in Edge computing recognizes and briefly discusses types of communication [16]. Notably, P2P approaches have shown great potential to handle edge infrastructures in a scalable manner [17]. Therefore, a lot of research has been conducted in this context, resulting in many approaches that aim at organizing edge nodes using different communication types [18]. In contrast to the mentioned papers, Yi et al. [5] proposes Latency-Aware Video Edge Analytics (LAVEA) system and discuss factors that impact the feasibility of realizing practical Edge computing systems.

According to [10], the communication type of a platform affects the functionality of the final applications deployed at the edge infrastructure. In networks organized in P2P, it is assumed that participating nodes are equal in terms of their computation capabilities. In such an organization type, resource heterogeneity is not taken into account. Notably, such environments have attracted many research papers to propose various fault-tolerance systems (e.g., [19]). In contrast to the proposed approaches, hierarchical communication type organizes nodes in layers according to the node resource capabilities [20]. In this type of organization, the node in the highest level of the hierarchy is responsible for the network's global coordination. Similarly, in [21], fog nodes in the network are organized hierarchically. However, such approaches determine a coordinator statically, which resides in the cloud. In contrast, we propose a solution that automatically determines configurator placement in dynamic edge networks.

Resource allocation and management have been widely studied both in cloud and fog computing [16]. In Fog computing many factors have been considered including time (e.g., computation [22], communication time [23]), cost (e.g., networking [24]), deployment [25], resource coordination [26], or execution [27], which have been found to play important roles in resource and service provisioning. Skarlat et al. [28] proposes a framework called FogFrame, which aims to deploy and execute various workloads in the fog infrastructure. The proposed approach organizes nodes in the network with the cloud residing in the hierarchy's highest level.

Notably, none of these approaches considers the dynamic nature of edge networks. Scheduler and monitoring components are placed statically in the highest level of the hierarchy (i.e., cloud or fog devices) that are assumed to be powerful devices. However, when such nodes are fog devices, they may suffer from being overwhelmed and fail to process further deployment requests or fail to monitor edge infrastructure. Moreover, the QoS between the master and the other nodes may be degraded due to the high utilization or high end-to-end latency. As a result, our proposed solution aims to shift orchestrate and monitor functionalities closer to the edge and dynamically place them in the most suitable nodes. Such a solution makes edge networks *autonomous environments* and less dependent on centralized nodes that are located far away.

IV. Architecture and Resource Utilization Modeling

This section discusses architecture modeling, resource utilization modeling, and the makespan to run and transfer the configurator at the edge. Our assumption is that edge devices are single-core processors. We consider only single-core processors avoiding the need for local mapping of multi-core processors in edge devices. This is attributed to the different workloads that each core may have in time.

A. Architecture Modelling

We model our neighborhood as a graph $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ is a set of vertices that represent clusters, and $\mathcal{E}$ is a set of edges that captures the physical link between the vertices. We assume that the graph $\mathcal{G}$ is connected, i.e., there is always at least one path for every pair of vertices that connects them. Each vertex $\nu_i \in \mathcal{V}$ is a cluster (see Section II-B). A cluster ν_i is composed of a set of nodes Γ_i with individual functionalities. Each node $\gamma_j \in \Gamma_i$ is assumed to be a single-core processor that has workload denoted with w_j and a computation factor denoted with cf_j. A computation factor cf determines how fast the received data can be processed on the device (i.e., represent the core's clock speed).

We assume that the nodes in a cluster are connected in a P2P manner. We define a function $< \mathcal{W}_i, cf_i >= \mathcal{P}(\nu_i)$ which finds the leader ρ_i in the cluster ν_i that has the lowest workload. The function returns the $\mathcal{W}_i$ and cf_i of the cluster leader (see Section V-A). The workload w of the device is the value that shows how much the CPU core is utilized. We use the Worst-Case Execution Time (WCET) to determine the maximum time it takes to execute a given piece of code (i.e., task) on a given device with cf_i. For instance, consider a task with the WCET of 2 ms and runs on a device with a core with the $cf_1 = 0.8$. The WCET of the task would be 1.6 ms. Thus, devices with a higher cf execute tasks faster rather than those with lower ones.

Each edge $\epsilon \in \mathcal{E}$ is a full-duplex physical link and associated with a bandwidth β_ϵ. We also define a function $< \mathcal{R}, \alpha_{i,j} >=$

$\mathcal{O}(\nu_i, \nu_j)$ which returns a path $\mathcal{R}$ between the clusters ν_i and ν_j which has the maximum total bandwidth $\alpha_{i,j}$.

We model the configurator as a single task with a known WCET, which can be scaled for a given core with the associated computation factor. We use $\mathcal{T}$ for the WCET of the configurator, and the size of the configurator is denoted with $\mathcal{L}$. An example of conceptual architecture is depicted in Figure 3, and details of the nodes are described in Table I. Notice, the bandwidth values in Table I are given in Mbps.

TABLE I
CONCEPTUAL ARCHITECTURE SHOWN IN FIGURE 3

Cluster ($\mathcal{V}$)	Comp. factor cf_i	Available Bandwidth ($\alpha_{i,j}$)
ν_1	1.1	$\alpha_{1,2}$ =22.67 $\alpha_{1,3}$ =32
ν_2	1.0	$\alpha_{2,1}$ =22.67 $\alpha_{2,3}$ =1000
ν_3	1.7	$\alpha_{3,1}$ =32 $\alpha_{3,2}$ =1000

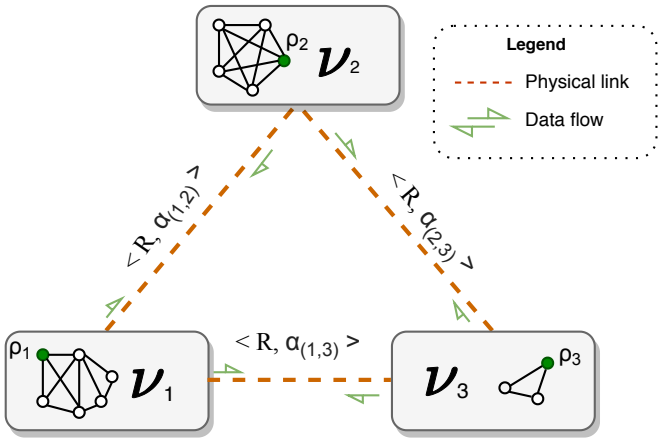

Fig. 3. Conceptual architecture.

The conceptual architecture has three clusters (i.e., cluster leader) that are connected with three links. Each link is associated with a bandwidth value, as presented in Table I. The values for the bandwidth of links $\alpha_{i,j}$ are randomly assigned (see in Table II). This assumption can be replaced with a function relying on *Assolo* [29], which enables collecting bandwidth probes.

TABLE II
THE QOS PROFILES OF COMMUNICATION LINKS [30]

Profile	Latency	Download	Upload
4G	53 ms	22.67 Mbps	16.97 Mbps
VDSL	60 ms	60 Mbps	6 Mbps
WLAN	15 ms	32 Mbps	32 Mbps
Fiber	5 ms	1000 Mbps	1000 Mbps

B. Modelling Resource Utilization

Edge networks are loosely coupled distributed systems and heterogeneous environments. Resource utilization of edge devices may change rapidly. Each edge device is equipped with limited physical resources such as computational processing, memory, and network bandwidth. The current resource utilization of edge devices (i.e., CPU, RAM, and storage) can be calculated with various approaches, as presented in research works [31], [32]. Such hardware information can be collected using *Hyperic Sigar* [33]. To meet our desired objectives, we adopt a similar approach based on the fuzzy theory [32], which represents the current utilization of resources through utilization scores (see in Table III).

TABLE III
NODES IN THE ARCHITECTURE SHOWN IN FIG.3

Leader	Current workload $\mathcal{W}$	Utilization Score $\mathcal{U}_w$
ρ_1	90%	0.90
ρ_2	52%	0.52
ρ_3	18%	0.18

We use a similar membership function to represent the utilization of resources defined as follows:

- $\mathcal{U}_w$ - The fuzzy subset and node CPU utilization score is represented as:
  - $Light : (0.1 < U_w < 0.49)$,
  - $Medium : (0.49 < U_w < 0.89)$,
  - $Heavy : (0.9 < U_w < 1.0)$.
- $\mathcal{U}_\beta$ - The fuzzy subset for bandwidth utilization is calculated as a percentage used from the total available bandwidth. Similar to the CPU utilization score, bandwidth utilization is represented as light, medium, and heavy.

The scores $\mathcal{U}_w$ and $\mathcal{U}_\beta$ are randomly generated and used in a function which determines when to trigger an event to start the process for placing configurator in a most suitable node (Section V).

C. Makespan

The WCET of the configurator is calculated for a base CPU, which has the computation factor cf_j. If the configurator is submitted to the node (edge device) γ_j with the cf_j, the execution time $\mathcal{C}_j^W$ can be calculated as in the given equation (1).

$$\mathcal{C}_j^W = \frac{\mathcal{T}}{cf_j} \tag{1}$$

We consider $\mathcal{C}_{\nu_{i,j}}^t$ to be the commutation time for transferring the configurator of size $\mathcal{L}$ from ν_i to ν_j which is calculated with given equation (2)

$$\mathcal{C}_{\nu_{i,j}}^t = \frac{\mathcal{L}}{\alpha_{i,j}} \tag{2}$$

The time required to execute the configurator and to transfer data between the edge devices is the makespan measured as in the formula (3).

$$\mathcal{C} = \mathcal{C}_j^W + \mathcal{C}_{\nu_{i,j}}^t \tag{3}$$

D. Cost Function

We define a cost function for choosing the most suited cluster in the neighborhood. The cost function gets the time required to execute the configurator $\mathcal{C}_i^W$ on the cluster leader ν_i, the time required to transfer the configurator from the current cluster ν_j to the cluster ν_i denoted with $\mathcal{C}^t_{\nu_{i,j}}$, and the workload of the cluster leader ν_i denoted with $\mathcal{W}_i$. The most suited cluster in the neighborhood has the minimum cost. The coefficients c_1, c_2 and c_3 are determined at design time.

$$\phi = c_1 \times \mathcal{C}_j^W + c_2 \times \mathcal{C}^t_{\nu_{i,j}} + c_3 \times \mathcal{W}_i. \tag{4}$$

V. The approach

In this section, we present two main processes of the proposed approach: *i)* the process of finding the leader node in a cluster (i.e., we refer to as the *Cluster Leader Algorithm*), and *ii)* the process of determining configurator placement (i.e., we refer as the *Mapper Algorithm*). The proposed method represents one of the main components of the configurator. Afterward, we discuss the scalability of the configurator. Finally, we present an example of the approach through a demonstration.

A. The Cluster Leader Algorithm

The cluster leader algorithm chooses a cluster leader, which is assumed to have a limited number of nodes. The first participant node of the cluster is automatically elected as a leader node. There are three ways on how the algorithm can be triggered: *i)* by the configurator node, *ii)* the cluster leader, and *iii)* by the cluster nodes. An election is triggered by a cluster leader when it suffers constant high processor utilization and wants to transfer leadership to another node. The algorithm uses control messages to check all the nodes in the cluster and find the minimum workload. In case the leader node fails, each node can start an election on its own at any time. The initiating node sends a control message with a predefined value for $\mathcal{T}$ to all nodes that will participate in the election; typically, those are all nodes associated with the local node's cluster. Otherwise, when the configurator node triggers the algorithm, it sends the current value $\mathcal{T}$ and finds the most suitable node to execute the configurator in the manager mode. The reason behind bounding the number of nodes in clusters has been already explained in Section II-B.

The cluster leader algorithm, as presented in Algorithm 1, runs on each cluster node separately. Essentially, the algorithm gets the current workload $\mathcal{W}$ (line 2), the computation factor of the core cf (line 3), and then updates the workload of the core as if it runs the configurator (line 4). Besides, a unique random signature (i.e., SHA-1 hash) is used to be broadcast along with the prediction of workload to all the nodes in the cluster (lines 5-6). Such a random number called $Signature$, helps to make the process and transmitted messages unique. We refer to the transmitted message as a *control message*.

The cluster leader algorithm determines whether the leader is itself or not in the limited time (e.g., *100 ms*), and it is defined as a $Deadline$ (line 8). The deadline value is specified at system design time. If the leader is not found in the given period, then the previous leader continues to lead the cluster unless if the leader has failed. Nevertheless, when a message from another node is received I (line 9), then the algorithm compares the workload of the received message with the workload of itself $\mathcal{W}$ (line 10). Once the received message has a better workload than the node itself, the algorithm stops since a better node is known in the cluster. In case of equal workload, the signature helps choose the leader node as given in (line 14). The cluster leader node is the one on which the algorithm is still running since it has the lowest workload. Afterward, the algorithm returns $\mathcal{W}_i$ and cf_i of it as an output. It is worth noting that we consider energy-powered edge devices in the edge neighborhood. For environments with energy-restricted devices, the number of nodes in clusters should be kept lower, or a more lightweight solution can be considered to elect a cluster leader (e.g., the Bully Algorithm [34]).

Algorithm 1: Cluster Leader.

Input : ν_i
Output: $\mathcal{W}_i, cf_i$

```
1  t ← 0
2  W ← GetcoreWorkload()
3  cf ← Getcorecf()
4  W ← W + T/cf
5  Signature ← Random()
6  Broadcast_To_All(W, Signature)
7  Solution_Found ← False
8  while t < Deadline OR !Solution_Found do
9      I ← Receive_Message()
10     if getWorkload(I) < W then
11         Solution_Found ← True
12     else if I = W then
13         M ← Received_Signature(I)
14         if M < Signature then
15             Solution_Found ← True
16         end
17     end
18 end
19 if !Solution_Found then
20     return W, cf
21 end
```

B. The Mapper Algorithm

We design the mapper algorithm by considering the edge device's current workload, bandwidth, and the current configurator workload. The algorithm finds the solution for executing the configurator in the manager mode in the most suitable node (see Algorithm 2). The problem we have is formulated as follows. Essentially, we have given *i)* a network graph $\mathcal{G}$, *ii)* a configurator, *iii)* a function $\mathcal{P}$ for finding the cluster leader, and *iv)* a function $\mathcal{O}$ which returns a path $\mathcal{R}$ with maximum bandwidth between two clusters.

Algorithm 2: Mapper.

Input : $\mathcal{G}$, $\mathcal{O}$, $\mathcal{R}$
Output: ν_{best}

1 $\nu_{this} \leftarrow getCurrentCluster()$
2 $\nu_{best} \leftarrow$ `null`
3 $\phi_{best} \leftarrow 0$
4 **for** `each` $\nu_i \in \mathcal{E}$ `and` $\nu_i \neq \nu_{this}$ **do**
5 $< \mathcal{W}_i, cf_i >= \mathcal{P}(\nu_i)$
6 $\mathcal{C}^{W}_{\nu_i} = \frac{\mathcal{T}}{cf_i}$
7 $< \mathcal{R}, \alpha_{i,this} >= \mathcal{O}(\nu_i, \nu_{this})$
8 $\mathcal{C}^{t}_{\nu_{this,i}} = \frac{\mathcal{L}}{\alpha_{i,this}}$
9 $\mathcal{C}_i = \mathcal{C}^{t}_{\nu_{this,i}} + \mathcal{C}^{W}_{\nu_i} + \mathcal{W}_i$
10 **if** ν_{best}`is Null` **then**
11 $\phi_{best} \leftarrow \phi_i$
12 $\nu_{best} \leftarrow \nu_i$
13 **else if** $\phi_{best} \geq \phi_i$ **then**
14 $\phi_{best} \leftarrow \phi_i$
15 $\nu_{best} \leftarrow \nu_i$
16 **end**
17 **end**
18 `Set`$(\nu_{this}, EdgeMode)$
19 `Set`$(\nu_{best}, ManagerMode)$

A triggered event causes the overloading of the configurator node (i.e., the processor is overloaded). Such an event triggers our algorithms' execution, which seeks to determine the most suitable cluster ν_{best} where the configurator should be mapped to and be executed on in the manager mode.

The most suitable cluster ν_{best} should be determined such that *i)* there is always one and only one configurator running in the manager mode on the network, *ii)* the most suitable cluster ν_{best} has a good workload and the minimum execution time $\mathcal{T}_{e_j}$, and *iii)* the most suitable cluster ν_{best} has a good available bandwidth $\mathcal{C}_\beta$ and minimum transferring time for the configurator $\mathcal{C}_{e_{i,j}}$. The leader of the most suitable cluster ν_{best}, denoting with ρ_{best} is captured with $\mathcal{P}(\nu_{best})$.

When the mapper algorithm is triggered, the configurator checks all clusters in the neighborhood (line 17) for a node that can host the configurator. The algorithm calls the function $\mathcal{P}$, which executes Algorithm 1 for getting the cluster leader (line 5). In case the cluster leader cannot respond, it starts a new election and finds the leader of the cluster (see Section V-A). The cluster leader's computation factor is used to calculate the configurator's execution time on it (line 6). The function $\mathcal{O}$ is called to find the path from the cluster that currently runs the configurator to the cluster being checked (line 7). The total available bandwidth of the path $\alpha_{i,this}$ is used to calculate the configurator's transfer time (line 8). The total transfer time and execution time of the configurator are calculated in line 9. Afterward, the algorithm finds the most suitable cluster, which has the minimum execution and transfer time for the configurator, as well as the maximum available bandwidth and minimum workload (line 13). Eventually, if the determined cluster is not the existing one, the configurator will be set to the *ManagerMode*.

The process is activated each time when the configurator node in the manager mode is overwhelmed. It's worth noting that the CPU of an edge device may fluctuate up and down due to the various workloads. We consider a *threshold* to avoid such a situation. The default threshold is configured to alert when CPU utilization exceeds 90% for more than 30 seconds. Therefore, to identify to what degree an edge device is overloaded, we define rules in the fuzzy database. The rules are given as follows: whenever the CPU hits *heavy* utilization (see Table III) or *heavy* bandwidth utilization, the mapper algorithm is triggered to find a new edge device that can run configurator with the current workload.

C. Scalability

We propose the notion of the cluster to make the mapper algorithm scalable. The proposed method breaks the process of finding the most suitable node in the network into: *i)* finding the leader of the cluster, and *ii)* finding the most suitable cluster. The mapper algorithm can map the configurator to the best cluster, respectively, to the cluster leader in the finite time, as the network becomes bigger. As shown in Section IV-D, the most suitable cluster is the one that has relatively good available bandwidth that can be used to transfer the configurator too. In Section VI, we show that the time required to transfer and activate the configurator on the most suitable cluster even when the number of nodes and clusters increases is in limited time.

D. Demonstration of the Mapper Algorithm

In this part, we present an example of our proposed approach; respectively, we demonstrate the mapper algorithm (Algorithm 2). In our example, we assumed three clusters create a neighborhood. The cluster ν_1 is hosting the configurator, i.e., configurator in the manager mode, respectively, configurator node ρ_{best}. At the same time, the other clusters are running the configurator in edge mode.e. Let us assume that the current workload of the configurator node ρ_{best} of the cluster ν_1 is 34%, the available bandwidth is 84 Mbps, and the computation factor is 1.1.

TABLE IV
CONFIGURATOR IN THE OVERLOADED NODE ρ_{best} IN CLUSTER ν_{best}

Leader (ρ)	**Workload** ($\mathcal{W}$)	**Configurator**	Cost (ϕ_{best})
$\nu_1(\rho_{best})$	94%	Manager	0.921
$\nu_2(\rho_2)$	53%	Edge	0.641
$\nu_3(\rho_3)$	34%	Edge	0.457

Initially, an event makes the cluster ν_1 overloaded, respectively, the workload on the configurator node ρ_{best} exceeds 94% of processor utilization as given in Table IV. After the CPU remains utilized for more than 30 seconds, another event is triggered, alerting the processor's heavy usage. Such an event triggers the mapper algorithm on the configurator

node ρ_{best} in the cluster ν_1. The algorithm iterates over all the network clusters, namely the cluster ν_2 and ν_3. The algorithm finds its leader for each cluster, using the function $\mathcal{P}$, its workload, and its bandwidth. For the cluster ν_2, the leader is the node ρ_2, and its workload is 53% and has 90 Mbps of available bandwidth, with the computation factor of 1. For the cluster ν_3, the output would be the leader node ν_3 with the workload of 34%, the available bandwidth of 45 Mbps, and the computation factor of 1.05.

We assume that the current size of the configurator is 10 Mbits, the current utilization is 10%, and coefficients for the cost function are equal to 1 (see eq. 4 in Section IV-D). The algorithm chooses the cluster ν_3 with the lowest total cost of 0.457 compared to the cost of 0.641 for the cluster ν_2. Finally, the algorithm sets itself to the EdgeMode as well as it sets the configurator running on the cluster ν_3 to the ManagerMode.

VI. Evaluation

In this section, we first introduce our evaluation setup environment, prototype details, and limitations. Next, we experimentally evaluate the effectiveness of the approach by running multiple experiments and checking the mapper algorithm's behavior in different situations.

A. Setup, Prototype Details, and Limitations

For evaluating the proposed approach, we develop a prototype that implements core functionalities to determine configurator placement at the edge. The prototype is written in MATLAB, and it is deployed on a laptop with a Core i7 CPU at 2.8 GHz and 16 GB of RAM. For the evaluation, we run 100 experiments per number of clusters and show results from various cases. These results are in terms of: *i)* analyzing the scalability and time required to determine configurator placement, *ii)* analyzing the overhead for transferring and execution time of the configurator, and *iii)* analyzing the overhead of bandwidth usage (Section VI-B).

The current version of the prototype shows the feasibility of the proposed approach in determining the configurator placement at the edge. However, some values such as measuring bandwidth, hardware related metrics, and configurator data size are randomly generated. Notably, such parameters are not necessary for this simulation since our goal is to show the feasibility and the scalability of the configurator without causing overheads. The current version of the prototype does not provide the configurator's failure mechanism. In future work, we plan to adopt a similar approach [19], where the configurator data is stored in a distributed manner among the edge nodes.

B. Experiments and Results

We evaluate our proposed method on five test cases which have different sizes. The size of test cases progressively increases and reaches 200 clusters that have at least 28000 nodes. For simplicity, in this work, we assume that the maximum number of nodes in clusters is defined not to be more than 250 nodes. Thus, for each experiment, we generate a random number of nodes in the clusters. In the smallest test case, we have 10 clusters with 1864 nodes, and in the largest test case, we have 200 clusters with 28255 nodes. The five test cases and their relative number of clusters, including the number of nodes, are depicted in Figure 4. The blue box shows the minimum number of nodes in a cluster, the red box shows the average, and the green box shows the maximum number of nodes generated randomly for 100 iterations. Figure 4 is an objective of the proposed approach, as discussed in Section V-C, which advocates the scalability of the configurator at the edge neighborhood.

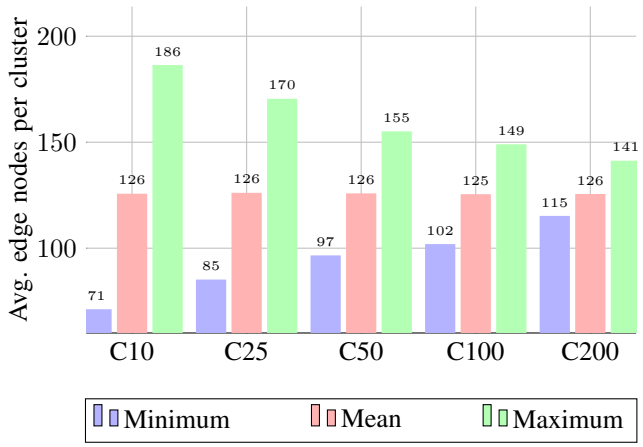

Fig. 4. The number of clusters and their nodes in five test cases. The test cases have respectively 10, 25, 50, 100, 200 clusters, and for each test case, we iterate the simulation 100 times.

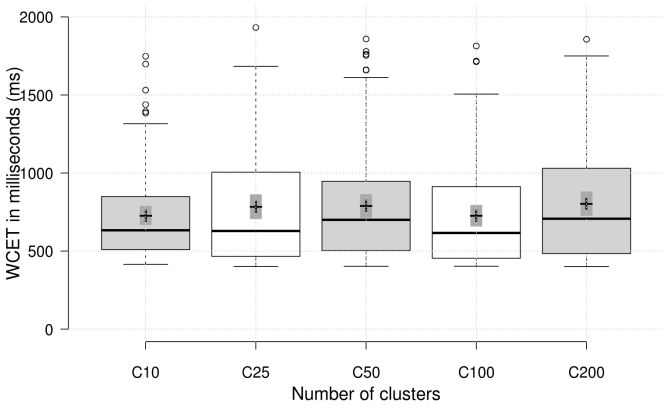

Fig. 5. The WCET of the configurator executed on the most suitable cluster leader in the neighborhood. The minimum and average WCETs are almost the same for the five test cases, though the maximum differs, which implies bandwidth's impact on choosing the most suitable cluster.

We also evaluate the WCETs of the configurator executed on the most suitable cluster leader. The result of the evaluation is depicted in Figure 5. The results show that the proposed algorithm has almost found the core that has the most suitable computation resources in all test cases. The average WCET of the configurator on the test cases are nearly the same and has the value of 657 ms. However, the WCET of the configurator reaches almost 1700 ms in test cases with 25 clusters and with

an average of 3153 nodes. Notably, the result shows also to have the best cost function that imposes to have an excellent available bandwidth.

In Figure 6, we show the maximum and minimum bandwidth usage during all simulations in the edge neighborhood. The maximum and minimum values vary depending on the test case, though the results match the relevant results of the WCETs. Evidently, these experiments' results show that adding more edge devices/clusters in the problem instance does not increase the network and computation consumption. This is in line with the recent literature review [35], which suggests that the edge-based systems need to operate at large-scale networks.

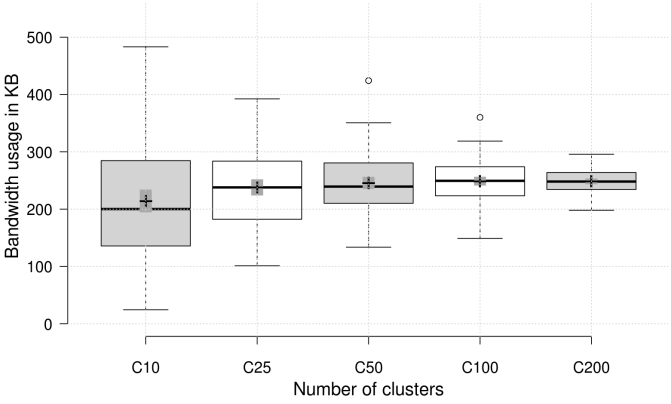

Fig. 6. The total bandwidth usage in five test cases in the edge neighborhood. The average values are almost the same.

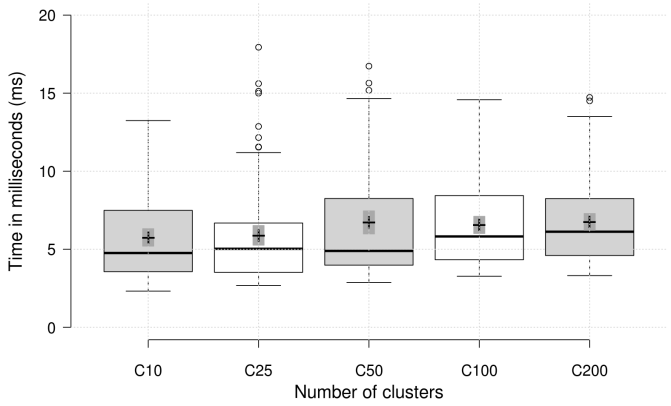

Fig. 7. The time required to activate and transfer configurator data on the hosting node. The average values are almost the same.

We evaluate the proposed algorithm in terms of the activation time of the configurator. The results are depicted in Figure 7. The activation time is the required time for the configurator to be transferred to the most suitable cluster leader and executed on it as a manager. We measure this duration by having the neighborhood's topology, routing, and the relative bandwidth of the links and the leader's computation factor. The results show that the activation time is bounded and has almost the same value in all test cases. Even when the number of clusters and the nodes is progressively increased, the activation time is bounded. Notably, the activation time varies marginally in all test cases and iterations, implying the configurator's scalability.

VII. Conclusion and Future Work

Compute and storage resources at the edge of the network are used to bridge the gap between the Cloud and IoT domains to facilitate low-latency and highly resilient applications. However, the broad range of IoT application requirements concerning latency and QoS, combined with the heterogeneous and dynamic nature of edge networks, make it particularly challenging to orchestrate, deploy, and operate such applications. To overcome these challenges, we introduce a decentralized mechanism called *configurator*. In this setting, one critical task is to determine configurator placement at the edge. Therefore, in this paper, we propose an efficient decentralized approach that determines the most suited edge device to execute the configurator in a given dynamic edge network. We have implemented a prototype and evaluate the proposed approach's feasibility by simulating configurator placement at the edge.

We claim that the configurator at the edge paves the way for utilizing available resources, leading to accomplish the promised high quality and low-latency services. Despite the promising results, this paper is only a small step towards the configurator's operationalization, aiming to achieve efficient resource utilization in edge networks. Regarding future work, we first plan to build the edge neighborhood based on the Kademlia. It remains to provide a complete solution for the configurator and a full solution stack for edge applications that are dynamically distributed, elastic, resilient, and run natively in the Edge–Cloud continuum. Finally, some assumptions made regarding workloads and the bandwidth will be replaced with the mentioned approaches in future work.

Acknowledgment

Research partially supported by the Smart Communities and Technologies (Smart CT) at TU Vienna and the EU H2020 Marie Skłodowska-Curie grant No. 764785 FORA–Fog Computing for Robotics and Industrial Automation.

References

[1] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[2] Nianyu Li, Christos Tsigkanos, Zhi Jin, Schahram Dustdar, Zhenjiang Hu, and Carlo 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.

[3] Ilir Murturi, Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. Edge-to-edge resource discovery using metadata replication. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–6. IEEE, 2019.

[4] Antonio Brogi, Stefano Forti, and Ahmad Ibrahim. How to best deploy your fog applications, probably. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 105–114. IEEE, 2017.

[5] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–13, 2017.

[6] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards qos-aware fog service placement. In *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pages 89–96. IEEE, 2017.

[7] Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. Decentralized resource auctioning for latency-sensitive edge computing. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 72–76. IEEE, 2019.

[8] Schahram Dustdar, Cosmin Avasalcai, and Ilir Murturi. Edge and fog computing: Vision and research challenges. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 96–9609. IEEE, 2019.

[9] Cosmin Avasalcai, Ilir Murturi, and Schahram Dustdar. Edge and fog: A survey, use cases, and future challenges. *Fog Computing: Theory and Practice*, 2020.

[10] Vasileios Karagiannis. Compute node communication in the fog: Survey and research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT*, pages 36–40. ACM, 2019.

[11] Vasileios Karagiannis, Stefan Schulte, Joao Leitao, et al. Enabling fog computing using self-organizing compute nodes. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2019.

[12] Salman Taherizadeh, Andrew C Jones, Ian Taylor, Zhiming Zhao, and Vlado Stankovski. Monitoring self-adaptive applications within edge computing frameworks: A state-of-the-art review. *Journal of Systems and Software*, 136:19–38, 2018.

[13] Gian Paolo Jesi, Alberto Montresor, and Ozalp Babaoglu. Proximity-aware superpeer overlay topologies. In *IEEE International Workshop on Self-Managed Networks, Systems, and Services*, pages 43–57. Springer, 2006.

[14] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[15] Xhevahir Bajrami and Ilir Murturi. An efficient approach to monitoring environmental conditions using a wireless sensor network and nodemcu. *e & i Elektrotechnik und Informationstechnik*, 135(3):294–301, 2018.

[16] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of everything*, pages 103–130. Springer, 2018.

[17] Adrien Lebre, Jonathan Pastor, Anthony Simonet, and Frédéric Desprez. Revising openstack to operate fog/edge computing infrastructures. In *2017 IEEE international conference on cloud engineering (IC2E)*, pages 138–148. IEEE, 2017.

[18] Federico Rizzo, Giovanni Luca Spoto, Paolo Brizzi, Dario Bonino, Giuseppe Di Bella, and Pino Castrogiovanni. Beekup: A distributed and safe p2p storage framework for ioe applications. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 44–51. IEEE, 2017.

[19] Frank HP Fitzek et al. On network coded distributed storage: How to repair in a fog of unreliable peers. In *2016 International Symposium on Wireless Communication Systems (ISWCS)*, pages 188–193. IEEE, 2016.

[20] Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized iot service placement in the fog. *Service Oriented Computing and Applications*, 11(4):427–443, 2017.

[21] Djabir Abdeldjalil Chekired and Lyes Khoukhi. Multi-tier fog architecture: A new delay-tolerant network for iot data processing. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[22] Deze Zeng, Lin Gu, Song Guo, Zixue Cheng, and Shui Yu. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712, 2016.

[23] Fatemeh Jalali, Kerry Hinton, Robert Ayre, Tansu Alpcan, and Rodney S Tucker. Fog computing may help to save energy in cloud computing. *IEEE Journal on Selected Areas in Communications*, 34(5):1728–1739, 2016.

[24] Lin Gu, Deze Zeng, Song Guo, Ahmed Barnawi, and Yong Xiang. Cost efficient resource management in fog computing supported medical cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 5(1):108–119, 2015.

[25] Fahed Alkhabbas, Ilir Murturi, Romina Spalazzese, Paul Davidsson, and Schahram Dustdar. A goal-driven approach for deploying self-adaptive iot systems. In *IEEE International Conference on Software Architecture (ICSA 2020)*, pages 1–11. IEEE, 2020.

[26] Christos Tsigkanos, Ilir Murturi, and Schahram Dustdar. Dependable resource coordination on the edge at runtime. *Proceedings of the IEEE*, 2019.

[27] Mohammed A Hassan, Mengbai Xiao, Qi Wei, and Songqing Chen. Help your mobile applications with fog computing. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking-Workshops (SECON Workshops)*, pages 1–6. IEEE, 2015.

[28] Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. A framework for optimization, service placement, and runtime operation in the fog. In *2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC)*, pages 164–173. IEEE, 2018.

[29] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. Assolo, a new method for available bandwidth estimation. In *2009 Fourth International Conference on Internet Monitoring and Protection*, pages 130–136. IEEE, 2009.

[30] Antonio Brogi, Stefano Forti, and Ahmad Ibrahim. Deploying fog applications: How much does it cost, by the way? In *CLOSER*, pages 68–77, 2018.

[31] Jiayi Zhou, Kun-Ming Yu, Chih-Hsun Chou, Li-An Yang, and Zhi-Jie Luo. A dynamic resource broker and fuzzy logic based scheduling algorithm in grid environment. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 604–613. Springer, 2007.

[32] Fan-Hsun Tseng, Ming-Shiun Tsai, Chia-Wei Tseng, Yao-Tsung Yang, Chien-Chang Liu, and Li-Der Chou. A lightweight autoscaling mechanism for fog computing in industrial applications. *IEEE Transactions on Industrial Informatics*, 14(10):4529–4537, 2018.

[33] Hyperic Sigar. https://github.com/hyperic/sigar/wiki/overview, 2020. [Online accessed: January 2020].

[34] Jayshree Surolia and Mahesh M Bundele. Design and analysis of modified bully algorithm for leader election in distributed system. In *International Conference on Artificial Intelligence: Advances and Applications 2019*, pages 337–347. Springer, 2020.

[35] Carla Mouradian, Diala Naboulsi, Sami Yangui, Roch H Glitho, Monique J Morrow, and Paul A Polakos. A comprehensive survey on fog computing: State-of-the-art and research challenges. *IEEE Communications Surveys & Tutorials*, 20(1):416–464, 2017.