World Wide Web (2020) 23:1127–1175
https://doi.org/10.1007/s11280-019-00722-9

LW-CoEdge: a lightweight virtualization model and collaboration process for edge computing

Marcelo Pitanga Alves, et al. *[full author details at the end of the article]*

Received: 3 February 2019 / Revised: 20 June 2019 / Accepted: 12 August 2019 /
Published online: 7 November 2019

Abstract

Edge Computing is a novel paradigm that extends Cloud Computing by moving the computation closer to the end users and/or data sources. When considering Edge Computing, it is possible to design a three-tier architecture (comprising tiers for the cloud devices, edge devices, and end devices) which is useful to meet emerging IoT applications that demand low latency, geo-localization, and energy efficiency. Like the Cloud, the Edge Computing paradigm relies on virtualization. An Edge Computing virtualization model provides a set of virtual nodes (VNs) built on top of the physical devices that make up the three-tier architecture. It also provides the processes of provisioning and allocating VNs to IoT applications at the edge of the network. Performing these processes efficiently and cost-effectively raises a resource management challenge. Applying the traditional cloud virtualization models (typically centralized) to virtualize the edge tier, are unsuitable to handle emerging IoT application scenarios due to the specific features of the edge nodes, such as geographical distribution, heterogeneity and, resource constraints. Therefore, we propose a novel distributed and lightweight virtualization model targeting the edge tier, encompassing the specific requirements of IoT applications. We designed heuristic algorithms along with a P2P collaboration process to operate in our virtualization model. The algorithms perform (i) a distributed resource management process, and (ii) data sharing among neighboring VNs. The distributed resource management process provides each edge node with decision-making capability, engaging neighboring edge nodes to allocate or provision on-demand VNs. Thus, the distributed resource management improves system performance, serving more requests and handling edge node geographical distribution. Meanwhile, data sharing reduces the data transmissions between end devices and edge nodes, saving energy and reducing data traffic for IoT-edge infrastructures.

Keywords collaboration · data sharing · edge computing · lightweight virtualization · P2P · resource management

This article belongs to the Topical Collection: *Special Issue on Smart Computing and Cyber Technology for Cyberization*
Guest Editors: Xiaokang Zhou, Flavia C. Delicato, Kevin Wang, and Runhe Huang

1 Introduction

The Internet of Things (IoT) is considered one of the main components of the Internet of the future [69]. The IoT is a novel paradigm in which smart objects (things) communicate with each other and with physical and/or virtual resources through an existing network infrastructure, including the Internet [6, 33]. The IoT is attractive to several application domains (e.g., healthcare [16], smart cities [76], smart homes [74], and industry [57]). However, the wide dissemination of IoT applications and the growth of the number of devices connected to the Internet gave rise to many challenges regarding the infrastructure for supporting these applications. In this context, paradigms grounded on virtualized infrastructures, such as Cloud Computing [5], Cloud of Things (CoT) [17], Cloud of Sensors [60], Fog/Edge Computing [9, 10], and most recently the Edge Mesh [59] approach, have emerged as solutions for supporting the needs of storing, processing, and distribution of data generated by IoT devices.

In the last years, the Cloud Computing paradigm [5] has been advocated as a promising solution to tackle most of the IoT issues regarding reliability, performance, and scalability by playing the role of intermediary between smart objects and applications that use data and resources provided by these objects. Indeed, with its vast computational capacity, cloud computing comes hand-in-hand with IoT to act as backend infrastructure for processing and long-term storage the massive amount of data produced by IoT devices. The integration between IoT and Cloud has been addressed by several works and gave birth to the paradigm called Cloud of Things (CoT) [2, 4, 11, 17, 21, 60]. CoT extends the traditional Cloud models (IaaS, PaaS, and SaaS) to include novel models as, for instance, Sensing as a Service [63], Sensing and Actuation as a Service [23] to name a few.

Despite its advantages, the Cloud provides centralized services that are unsuitable to meet requirements of some types of IoT applications. There are still problems unsolved on the application side, mainly concerning the latency in the data transmissions between the devices and the Cloud, mobility, geo-distribution, energy efficiency, location-awareness, among others [9, 10, 13, 53, 56, 79]. To overcome these open issues, the Edge Computing paradigm has recently emerged as a solution for delivering data and real-time data processing closer to the data source [77].

Fog/Edge Computing [1, 9, 10, 43, 46, 50, 65] is a computing paradigm strongly based on virtualization [40] of physical edge devices, allowing the decoupling of the applications from both the edge infrastructure and the *end devices*. To achieve this goal, the physical infrastructure is abstracted through a set of virtual entities. The **virtual entity** represents the resources available at the Fog infrastructure (e.g., data, computation, and communication capabilities) to users. The ***end devices*** are devices capable of performing sensing and actuation tasks (e.g., a temperature sensor, a relay actuator). According to Yi et.al [77], the **physical edge devices** (also called Fog or Edge nodes) encompass both resource-poor devices (e.g., access points, smart routers, switches, base stations and smart sensors), and more powerful computing nodes as Micro Datacenters, and server clusters such as Cloudlet [62] and IOx [20]. These devices are heterogeneous regarding (i) their physical capabilities/resources (vertical heterogeneity) and (ii) their provided services (horizontal heterogeneity) [32]. Moreover, they can provide resources for services at the edge of the network and execute tasks which were previously assigned to the cloud (e.g., data collection from *end devices*, data storage, data preprocessing, and data filtering), thereby decreasing the data traffic to the cloud servers. Therefore, the Fog Computing already meets some requirements of emerging IoT applications such as low latency, energy efficiency, and location-awareness. It is worth noting that the terms “Fog”

[8] and "Edge" [31] come from different communities, and sometimes, have a slightly different meaning for expressing the same type of computation. Thereby, we are adopting the term "Edge" for the remaining of this paper to denote the tier (providing computation, storage and communication capabilities) between physical objects (things/sensors) that act as data source, and the remote cloud data centers.

Although Edge computing already meets some requirements of emerging IoT applications (e.g., low latency, energy efficiency, and location-awareness), several other challenges need to be dealt with to fully benefit from the synergy of these two technologies. Among such challenges, we can mention dealing with the high heterogeneity (of devices and applications), resource-constraint (in comparison to the cloud), mobility and geo-distribution of the edge devices, and promoting a fair and cost-effective consumption of the available resources. In our work, we focus on the challenges of designing a suitable **virtualization model,** of managing the **collaboration process** among edge devices and solving **resource management** issues. Our goal is to leverage the state of the art in Edge Computing and CoT paradigms to meet requirements such as heterogeneity, mobility, and geo-distribution of devices, scalability of services, and low response time [3, 9, 32, 50]. The addressed challenges and the respective requirements to be met are described in the next Section.

1.1 Current challenges in edge computing

In this Section, we describe the research challenges found in the current literature regarding the virtualization model, collaboration process, and resource management for Edge Computing, which are within the scope of the contributions of this paper.

Virtualization is widely used and well addressed in Cloud Computing. However, in the context of Edge Computing, it is necessary to consider the specific features of the edge devices, such as heterogeneity, resource-constraint and geographical distribution (geo-location) [50, 65, 77], and propose novel models tailored to this new scenario. The geo-location of edge nodes at the edge of the network has advantages compared to cloud nodes by allowing taking computing closer to data sources. However, this geo-location also brings challenges regarding managing and deployment of the applications. The heterogeneity of edge nodes hinders both the development and maintenance of the services provided by these nodes regarding the complexity and amount of generated code, besides the time of coding [15, 30]. Since edge devices are more restricted in resources than cloud nodes, Edge Computing requires lighter virtualization models. Moreover, the design of a new virtualization model for the edge must also consider the requirements of emerging IoT applications, such as low latency and fast response to events.

Another challenge faced when proposing a virtualization model for edge computing is how to perform the data sharing between the virtual entities [59]. The data sharing can bring benefits to the system such as improving the energy saving (thus increasing the lifespan of sensors) and reducing the bandwidth consumption. Santos et al. [61] proposed a virtualization model where Virtual Nodes (VN) are provided to store and share (with other VNs) raw sensing data generated from one or many sensor devices or processed data using a data fusion technique. In this model, the VNs can only interact with each other within the same workflow, where a workflow represents the set of application tasks. Thus, whenever there is a dependency between data inputs and outputs of the application requests, the VNs will need to share their data. This model represents a typical dataflow approach, where the result of the processing (data output) from a node is the input of processing (data input) of another node

[38]. Such approach is passive and data sharing only occurs when requested, i.e., a VN does not automatically send its data to other VNs which share the same datatype. Therefore, the challenge is to make the data sharing an active process using a collaboration process in Edge Computing, allowing a VN to share its data with neighboring VNs, efficiently and transparently, to improve the sensor lifespan (energy saving).

Regarding the collaboration process, several works in Edge Computing use the hierarchical model to share resources, since such a model makes easy the set-up of collaboration between edge nodes [10, 13, 46, 66, 72]. According to Mahmud et al. [46], this model is typically based on the concept of master-slave, in which the master edge node controls functionalities, processing load, data flow, etc. of its subordinated slave nodes. Although this approach facilitates the collaboration, it has at least two significant drawbacks. The first drawback concerns the strong dependency on the master node. In case of failures of the master node, the communication with slave-nodes is interrupted, leaving the underlying network inaccessible. This problem generates the second drawback by requiring procedures to re-setup the hierarchy structure which can consume many processing resources, bandwidth, and energy, thus affecting the capability of the system to meet the application requests. According to Mahmud et al. [46], the master-slave approach when assembled in a cluster model or Peer-to-Peer (P2P) is inadequate in real-time data processing environments since the communication between master-slave produces high bandwidth consumption. A feasible solution regarding such drawbacks is to organize the edge nodes in groups, i.e., in a neighborhood, and promote a node-to-node communication (within the neighborhood), without the need of a node acting as the communication controller. Therefore, since the hierarchical approach has its drawbacks in a dynamic scenario as the Edge Computing environment, we argue that, for adequately tackling this challenging issue, the flat P2P approach [46] is a feasible solution for the collaboration process between edge nodes. In such approach, the closeness defines the connection between nodes, thereby reducing the latency and saving bandwidth during the communication and indeed improves the response time to meet, for instance, the request for a latency-sensitive application.

Concerning resource management, it encompasses two primary activities, namely resource allocation and resource provisioning. Resource provisioning is responsible for preparing the infrastructure to host and instantiate the virtual entities to further meet the requests issued by the applications [66, 72, 78]. In turn, resource allocation allocates the resources required by virtual entities to adequately meet the workload of the multiple applications using the edge infrastructure in a given moment [22, 26]. The solutions for resource management are well established in the Cloud Computing field. However, in a scenario involving a mix of IoT and edge nodes, the centralized model of the Cloud is unsuitable since the edge nodes are distributed along the edge of the network. Several issues remain open in this regard [22, 59, 72, 75]. Recent works have been proposing [61, 66, 72] models to deal with resource management in Edge Computing. However, these works adopt centralized or hierarchical models to support the distributed decision-making at the edge of the network. In these models, the master node is responsible for identifying the edge slave-node capable of receiving the workload. The slave node is the one in charge of provisioning the required resources to run the workload. Therefore, existing solutions are not fully distributed, and present the drawbacks beforementioned. We argue that, for properly tackling this challenging issue, it is necessary to distribute the resource management entirely at the edge of the network.

1.2 Goals of our work

In order to contribute for providing solutions to the challenges mentioned in Section 1.1, and advance the state of the art in Edge Computing, in this paper we propose and evaluate LW-CoEdge, a new virtualization model improved with a collaboration process. The main goal of this work is to design a lightweight virtualization model for Edge Computing, which is based on the virtual node concept as proposed by Olympus [60, 61], and enhanced with a flat P2P collaboration process to allow both the data sharing and the distributed resource management at the edge of the network.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 describes our lightweight virtualization model and the collaboration process. Section 4 describes the heuristic algorithms for our distributed resource management and P2P collaboration. Sections 5 and 6 discuss the performed evaluation and final remarks.

2 Related work

In this Section, we describe the works related to ours in the light of the following main aspects: (i) computing paradigm and virtualization model, (ii) P2P collaboration, (iii) resource management, and (iv) architecture (two or three-tiers). At the end, we present a classification of the described papers.

2.1 Computing paradigm and virtualization model

Two recent works brought significant advances to the Cloud of Sensors (CoS) field [45, 61]. Madria et al. [45] proposed a centralized virtualization model, which encompasses Virtual Sensors and provides *sensing as a service* for the users (SaaS). Our proposal differs from [45] since we implement a decentralized virtualization model tailored to meet requirements of emerging IoT applications such as low latency and location-awareness. The authors in Santos et al. [61] extended their original design of Olympus [60] to create a three-tier CoS infrastructure by including the Edge tier to provision Virtual Nodes (VN) at the edge of the network. Our proposal differs from Olympus in two essential aspects. First, we provide a process of collaboration between the VNs to actively share fresh data with neighboring VNs. Thus, we avoid re-reading the sensors to get the same data, thereby improving response time, latency, bandwidth, and sensor lifespan. Second, Olympus defines the VN as a program capable of performing a set of information fusion techniques based on application requirements. Unlike Olympus, our model provides predefined types of VNs representing each data type provided to serve the application requests. This is important to favor the collaboration process among virtual nodes. Sahni et al. [59] present a novel computing approach named Edge Mesh integrating the best characteristics of the Cloud Computing, Edge Computing, and Cooperative Computing into a mesh network of edge devices and routers to decentralize decision-making tasks instead of sending them to be processed by a centralized server. It enables collaboration between edge devices for data sharing and computation tasks as well as the interaction between different *end devices*. However, the authors present several open issues for implementing the communication between different types of devices. Some open issues are how and which data are shared between edge devices, and the appropriate local to execute the intelligence of the application at the edge of the network. Our proposal leverages the advances promoted by the

Edge mesh approach and addresses the related open issues. We implemented a P2P collaboration process for enabling the data sharing between VNs and a resource management process distributed between edge nodes. Our data sharing process is smart, because it considers the data requirements of other VNs before replicating the data. Moreover, the distributed resource management provides for each edge node the ability to make decisions and engage neighboring edge nodes to allocate or provision VNs whenever it is necessary.

2.2 Collaboration

According to Mahmud et al. [46], the implementation of P2P collaboration can be hierarchical and non-hierarchical (flat). Mobile Fog [35] is a PaaS that provides a high-level programming model based on events. Applications use Mobile Fog to send messages between nodes or to get sensing data from an active sensor, besides other functions. Mobile Fog uses a hierarchical network topology of edge devices. It distributes the processes through the cloud (the root node of a hierarchy), fog (intermediate nodes) and edge (leaf node). Our proposal differs from the Mobile Fog approach in two fundamental aspects. First, we adopt a flat P2P collaboration between edge nodes to avoid the drawbacks related to the hierarchical model. In a flat P2P model, if a physical node fails only the VNs provisioned in this physical node are lost and the other VNs continue working. Second, our proposal provides VNs that are already pre-programmed to perform the essential operations to deliver sensing data or perform actuation. Thus, the end-users access the VNs by including in their applications a simple call to submit the requests to our CoT system, without the need to implement the VN behavior from scratch. Shi et al. [64] provide a non-hierarchical P2P view of Fog Computing that connects the cloud of sensors and smart devices via mobile devices. Moreover, the infrastructure offers and consumes resources and services of mobile devices through the REST pattern using the CoAP protocol, thereby promoting the dissemination of data between users in a decentralized way. Unlike Shi et al. [64], we designed a new virtualization model capable of running VNs for providing sensing data or performing actuation in response to the application requests directly at the edge of the network. Moreover, we implemented a process of collaboration to allow VNs to share data with their neighboring nodes without the user mediation, thereby improving the request response time and saving bandwidth.

Concerning the collaboration between edge nodes, Taleb et al. [68] introduced the “Follow Me Edge”. It is a concept based on Mobile Edge Computing (MEC) providing a two-tiered architecture to enable the containers migration across edge nodes according to the localization of their mobile users. Although the container migration emerges as a feasible solution for the mobility requirement, the authors claim that the selection of the appropriate technique to perform the migration is a vital task to avoid both communication latency and data synchronism problems. Our proposal differs from [68] by providing a flat P2P collaboration to share only the sensing data between edge nodes. Thus, we avoid transferring huge container images through the network, since each edge node already provides its services (VN container), thereby saving bandwidth and decreasing latency.

2.3 Resource management

Recent works on resource allocation and provisioning have been presented to take advantage of Edge Computing to enable workload offloading from the Cloud servers to edge devices, i.e., closer to the users. Zenith [75] is a novel resource allocation model

for Edge Computing. It uses Weighted Voronoi Diagrams (WVD) [36] to divide a geographic area composed of several Micro Datacenters (MDCs). Then it allocates the infrastructure resources (from the MDC with the lowest communication latency) bought by the service providers to run their services. Our model applies the ideas behind the Zenith approach by using the WVD algorithm to divide an Edge network into regions and build the neighborhoods of edge nodes. Unlike Zenith, our model assigns virtual nodes running in the edge node to provide either raw or processed data in response to the application requests. Wang et al. [72] present the Edge Node Resource Management (ENORM), a framework for handling the application requests and performing the workload offloading from the Cloud to running at the Edge network. ENORM addresses the resource management problem through a provisioning and deployment mechanism to integrate an edge node with a cloud server, and an auto-scaling tool to dynamically manage edge resources. Although we provide a resource management approach inspired by ENORM, our proposal is fully decentralized at the edge network. Such feature enables the edge nodes to find or provision the best VNs for providing either raw or aggregated sensing data, or performing actuation in response to the user application requests arriving from the cloud or the edge of the network. Skarlat et al. [66] present a

Table 1 Summary of the characteristics of Related Works

Papers	Degree of centralization			Two tier	Three tier	Lightweight	Considers the Edge tier	P2P Collaboration		Collaboration Process	
	I. Centralized	II. Decentralized	III. Hybrid					Hierarchical	Non-hierarchical (Flat)	Data sharing	Distributed Resource Management
Hong et al. [20]					✓		✓	✓			
Madria et al. [27]	✓			✓							
Santos et al. [36]			✓	✓							
Shi et al. [40]		✓							✓		
Skarlat et al. [42]	✓				✓			✓		✓	
Santos et al. [37]			✓		✓		✓				
Sahni et al. [35]		✓			✓		✓		✓	○	○
Taleb et al. [43]				✓		✓	✓				
Xu et al. [49]		✓					✓				
Wang et al. [48]	✓			✓			✓				
Munir et al. [31]					✓		✓				
LW-CoEdge		✓			✓	✓	✓		✓	✓	✓

hierarchical architecture of fog colonies for resource provisioning and orchestration in both the cloud and fog that takes into account the available resources in Fog/IoT scenarios. This architecture encompasses a cloud-fog control middleware that controls fog cells at the IoT tier through a fog orchestration control node. Fog cells are software components serving as access points to the IoT devices. They can receive and execute tasks (e.g., data analysis, data storage, monitoring, data transfer), allocate resources, and communicate to the fog orchestration control node to propagate tasks. The ideas behind this framework have a point of convergence with our proposal regarding performing data sharing. However, our approach implements the data sharing using the flat P2P collaboration, i.e., a full decentralized approach.

2.4 Architecture

Munir et al. [50] present the Integrated Fog Cloud IoT Architecture (IFCIoT), a fog-centric architectural paradigm that integrates IoT, Fog, and Cloud into the three-tier architecture to support five service layers, namely the Application, Analytics, Virtualization, Reconfigurable, and Hardware. It is designed to enhance system properties including performance, energy efficiency, latency, scalability, and to provide better-localized accuracy for IoT and cyber-physical systems (CPS) applications. Despite all the innovation, IFCIoT does not show how the fog nodes can collaborate with each other for data sharing, for instance. Our proposal differs from the IFCIoT by providing a light-weight virtualization model through virtual nodes and a collaboration process between the fog nodes that is fully distributed.

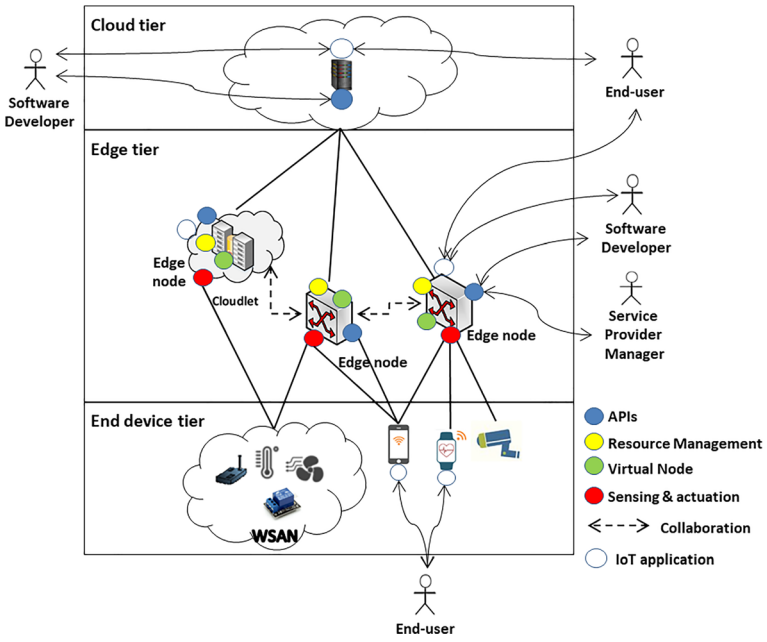


Figure 1 The three-tier architecture

2.5 Classification

Table 1 summarizes the main characteristics, contributions and open issues of the related works and includes the features of our work. In this table, the ✓ symbol indicates a feature supported by the respective work, and the ○ symbol means an open issue.

3 LW-CoEdge model

This Section presents LW-CoEdge, our proposal for a lightweight virtualization model enhanced with a collaboration process for Edge Computing. Initially, we detail the three-tier architecture considered in this paper (Section 3.1) and then we describe the virtualization model (Section 3.2). Next, we present the collaboration process (Section 3.3) that encompasses the data sharing and the distributed resource management.

3.1 The three-tier architecture

Figure 1 depicts the architecture adopted in this work to support the virtualization model. We assume an ecosystem composed of three tiers: (i) Cloud tier (CT), (ii) Edge tier (ET), and (iii) End device tier (EdT).

The Cloud tier (CT) is the top level of the architecture. It includes a set of cloud nodes $CN = \{cn_1, cn_2, \ldots, cn_n\}$. Each cn_i represents physical data centers that provide elastic resources on-demand (e.g., processing, storage, energy, and bandwidth). Hence, the CT can provide virtually unlimited resources to perform high-performance computing such as time-based data analysis and long-term storage of high amounts of data.

The Edge tier (ET) is the middle level and encompasses a set of edge nodes $EN = \{en_1, en_2, \ldots, en_n\}$ placed geographically closer to the data sources, i.e. they are closest to the End devices tier than the cloud nodes. Moreover, each en_i provides the same resources of cn_ibut in a smaller scale, i.e. en_i is a less powerful device than cn_i.

Finally, the End devices tier (EdT) is the bottom level and encompasses a set of *end devices* $Ed = \{ed_1, ed_2, \ldots, ed_n\}$ deployed over a geographical area. Each ed_i is heterogeneous regarding the processing speed, total memory, and energy capacity. Besides, it is able to provide sensing data and/or perform actuation tasks. Thus, the EdT may comprise smart devices such as a smartwatch or smart phone, for instance, or smart sensors connected and composing one or more Wireless Sensor and Actuator Networks (WSANs).

In our architecture, we assume that the cloud nodes (CNs) and edge nodes (ENs) host the majority of computational entities in our work, namely: APIs, Virtual Nodes, Resource Management components, and sensing & actuation components responsible for managing the interactions between the VNs and the End devices tier. The CN and EN host the APIs that expose the application entry point and the management services of the framework. The entry point is the service accessed by the users to submit the application requests. Concerning the components hosted at the EN, they are responsible for handling the application requests by performing tasks to provide sensing data (acquired from *end devices*) or to perform actions (on the *end devices*). All computational units are detailed in the following sections.

Furthermore, we identified the main actors in typical CoT infrastructures. These actors can be either humans or software systems. The software system can be either a web site (a Web Portal), or a web-service, or a mobile app that interacts with our CoT system by calling the

APIs for sending request for information from the monitored environment or to perform some actuation. Seismic monitoring, traffic monitoring, and temperature control are examples of typical IoT applications (accessed via web site or deployed as a mobile app). The human actor is a user of the system playing roles such as Software Developer, End-user, and Service Provider Manager. The **Software Developer** is the actor in charge of programming the software above-mentioned. He/She is also responsible for programming the end devices to send sensing data to the CoT system. The **End-user** is the actor in charge of operating the software to perform some action (sending a request to our system). The software operation occurs either via a Web Portal or via mobile devices (e.g., smartphone, tablet, smartwatch) in which the software is installed. The last human actor is the **Service Provider Manager** (SPM). The SPM is responsible for configuring the CoT system through a set of managing APIs to describe the datatype(s) used by the Virtual Node to meet the requests, besides configuring the neighborhood of edge nodes. The SPM can specify a simple datatype or combine datatypes to provide more accurate or more complex data for the application. Moreover, the SPM can engage one or many *end devices* to compose a datatype. In the next Section, we depict the virtualization model, the relation between its elements, and the formal description of the Virtual Node, the application request, and the Datatype descriptor used in the virtualization.

3.2 The virtualization model

LW-CoEdge provides the services of the physical infrastructure (things, edge, and cloud) to users and their applications by means of a virtualization model. Our proposed model leverages the Edge tier for the provision of **Virtual Nodes** (VNs) closer to the data sources (*end devices*)

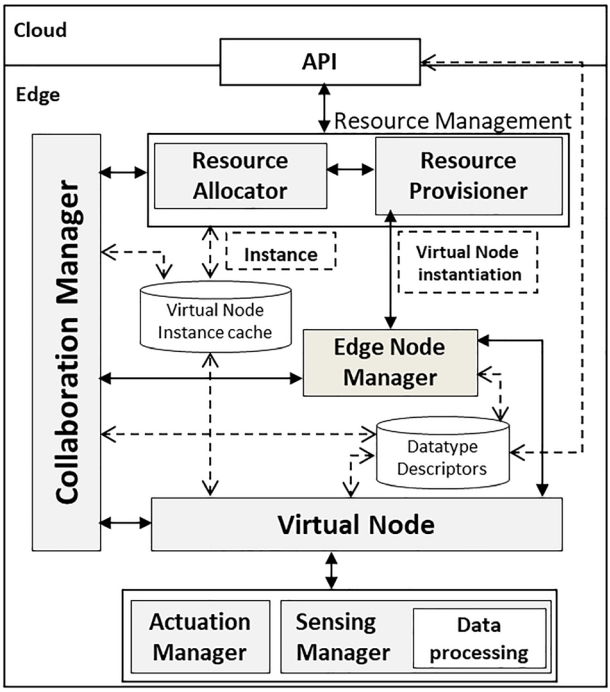


Figure 2 Virtualization model and collaboration process

besides providing a collaboration process to support distributed intelligence and decision-making at the edge tier. In addition, it promotes the decoupling of user's applications from the physical details of both edge nodes and *end devices*. Thus, we leverage the VN as the core computational unit of the virtualization model to provide sensing data or perform actuation in response to the application requests. LW-CoEdge comprises the process for VN instantiation along with three main sub-processes, namely (i) a process to perform the resource management (encompassing allocation and provisioning), (ii) a process for managing sensing and actuation tasks, and (iii) a process for managing the collaboration among nodes. Figure 2 summarizes the components implementing these processes and the relationships among them.

3.2.1 The definition of the application request and the system operation

Regarding the system operation, **End-users** operate software (web site or application) to perform actions that generate requests to the system using the system API. This API also provides the functionalities to list the available datatype descriptors for the **Software Developers** in their geolocation or for the entire system (when requested via cloud). From this list, the **Software Developers** choose the datatype to be used in their requests.

The request represents an abstract command to get sensing data or perform actuation in order to fulfill the functional requirements of the application. We describe the request (Figure 3a) as a tuple *Request* = (*datatype*, *param*, *callback*). The *datatype* denotes the type of requested data (it can be either a simple, or a complex type or an actuation command), and its specification is presented further below. In this property, the **Software Developer** sets only the *id* related to the datatype descriptor chosen previously from the repository. The *param* represents a set of properties required by the Virtual Node (VN) to process a datatype, and the *callback* is a notification procedure. Figure 3b presents an example of code representing the request of a *simple* datatype. The VN uses the callback address to deliver the processing response with either a data stream or the status of the actuation command to the request issuer. The properties of the *param* depend on the *datatype*. The *datatype* of the type *simple* encompasses the properties *fr (*data freshness threshold in milliseconds), *rtth* (it denotes the expected response time threshold in milliseconds) and the *sr* (sampling data rate). Finally, the *datatype* of the types *complex and actuation command* encompass only the property *rtth* since they do not obtain data from the sensor.

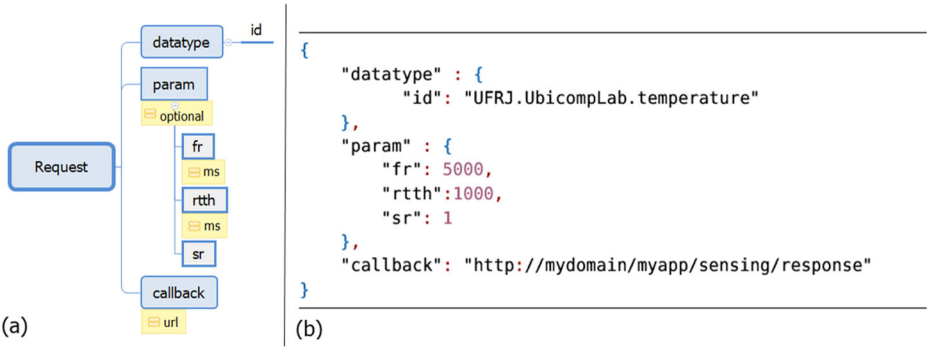


Figure 3 **a** The representation of the application request. **b** Application request example

The submitted requests arrive at the system through the entry point at the Cloud or via an edge node. When requests arrive via Cloud, the system must forward each request to the nearest edge node from the application request location. Therefore, the adoption of an algorithm to find the closest edge node is required. Nevertheless, the creation of the algorithm is out of the scope of our current work. Hence, we suggest the use of existing approaches in the literature to accomplish the task. Nishant et al. [54] and Xu et al. [75] are examples. Regardless the entry point in our CoT system, the requests are met by the virtualization system at the nearest selected edge node.

Inside the virtualization system, the **Resource Allocator** (RA) is the component in charge of the resource allocation process for allocating a Virtual Node (VN) in response to the application request. First, the RA receives a submitted request via API and then, searches in the cache of instances for a VN whose datatype matches the datatype requested by the application. When the VN is found, the RA forwards the request for the respective VN to execute the tasks and provide the requested datatype. In cases where the RA is not able to allocate a VN instance, or the selected one is busy fulfilling other requests, then the RA always invokes the **Resource Provisioner (RP).** The RP is the component in charge of the resource provisioning process. It is responsible for provisioning a new VN instance (whenever the RA did not find one available) or to scale-up an existing VN instance (when the current one is busy but has available resources). Moreover, the RP registers the new VN instances into the cache of instances and returns the new instance to the RA. The new (or scaled up) VN instance is created/configured to meet the datatype stated in the received request. The instantiation (deploy) and scale-up tasks are the responsibility of the **Edge Node manager**. **Edge Node manager** is an important component for abstracting the physical details of the edge nodes by providing services to support the operations of both the collaboration process and the resource provisioning. Besides the previous tasks, the Edge Node manager also allows un-deploying and scaling down the VN resources by checking the availability of edge node resources and other services, for instance, the load of edge node settings. Furthermore, the Edge node manager checks the availability and loads the datatype descriptors in the repository before performing its tasks. Thus, when the edge node does not have enough resources to instantiate the VN, the RP invokes the **Collaboration Manager** to identify a neighboring node to provision a new VN. However, if a neighbor cannot be selected, the request is refused and the RP throws a warning message to the RA. In parallel, the RP uses the **Collaboration Manager** that interacts with existing VNs to register the new VN instance for the purposes of performing the data sharing. The **Collaboration Manager** is the component in charge of enabling the collaboration process (Section 3.3) at the Edge tier.

At the time when a VN receives a request to process, it interacts with the **Sensing Manager** and the **Actuation Manager** components. These two are responsible for managing all interactions between the VN and the End devices tier. As the tasks of the VN are finalized, the VN sends the result of the processing to the request issuer by using the callback URL. For instance, when an actuation command arrives, the **Actuation Manager** sends it to be executed by the end device in the physical environment, and the result is sent back to the request issuer.

The importance of the **Sensing Manager** and the **Actuation Manager** is to abstract the complexity of dealing with the highly heterogeneous devices (both from the Edge tier and End device tier) regarding their physical capabilities/resources (vertical heterogeneity) and services (horizontal heterogeneity) [32] as well as all the communication processes. They are used to retrieve data (raw or processed) or to perform actuation on the physical environment depending on the requested datatype. Moreover, the **Sensing Manager** implements the **Data**

processing sub-process. It is responsible for abstracting the complexity of dealing with operations for getting sensing data from the physical devices, and operations on data (persistence, update, delete, and retrieval) in the historical databases maintained at the Edge tier.

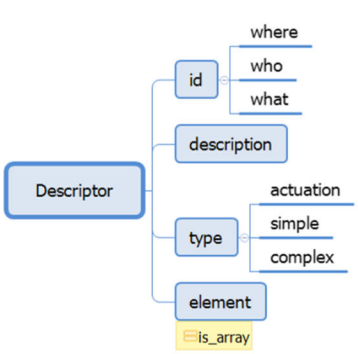
A concern regarding the VN operations is optimizing the use of resources, mainly the *end devices* energy. The VN always must check for the possibility of reusing data in the cache (memory) when the request is for getting sensing data, thus avoiding unnecessary access to the End devices tier. Therefore, the data freshness [12] is an essential requirement which a VN must verify when it is processing the request. The data freshness can be described according to Bouzeghoub [12] as "*The time elapsed from the last update to the source.*" In order to validate the data freshness, the VN checks the request time against the last acquisition date time of the provided data ($freshness = request\ time - acquisition\ datetime$) before accessing the **Sensing Manager**. Then, when the last data delivered is within a valid range time of data freshness ($request.param.fr \geq freshness$), the VN sends the data from the cache to the application. Otherwise, the VN interacts with the **Sensing Manager** to get fresh data before forwarding it to the application. In the end, the VN invokes the **Collaboration Manager** to share the new data with its neighbor VNs, thereby avoiding other VNs to access unnecessarily access the edge device tier for getting the same data. The criteria for defining the neighborhood of VNs are defined in Section 3.3.

3.2.2 The datatype descriptor

The datatype is a fundamental element used for the Virtual Node (VN) operations. The datatypes are specified by the **Service Provider Manager** (SPM) and they are stored in the **Datatype descriptors** repository. A datatype descriptor or simply *Descriptor* describes the elements to be used by the VN to get simple data or complex data or the outcome of an actuation task. Also, a *Descriptor* may be related to several VN instances. However, each VN instance refers to one *Descriptor*.

$$Descriptor = (id, description, type, element) \tag{1}$$

Per definition (1), each Descriptor (Figure 4a) is a tuple in terms of the properties id, description, type, and element. *Id* is the unique identification; *description* is a high-level description to help the Software Developers understand the type of data being provided. In this property, the SPMs can provide any technical level information that they deem to be




```
{
  "id": {
    "where": "UFRJ",
    "who"  : "UbicompLab",
    "what" : "temperature"
  },
  "description": "Laboratory Temperature (celsius)",
  "type": "simple",
  "element": [
    {
        "value":"DeviceID1"
    }
  ]
}
```

(b)

Figure 4 **a** The representation of the datatype descriptor. **b** Datatype descriptor example

relevant to the Software Developer, such as sensor accuracy and the periodicity of measurement. All these extra information are obtained from the infrastructure components used to interact with the physical environment; *type* denotes the type of information provided; and *element* represents an array where its elements are described further below. Figure 4b represents a datatype descriptor for a simple datatype.

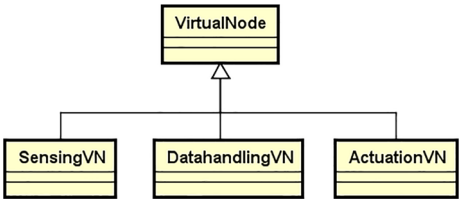
The datatype descriptor identification (*id*) is a fundamental property to facilitate its identification by users. Therefore, an efficient and friendly naming schema to identify a data type is needed. However, such a naming mechanism for the Edge Computing paradigm has not been built and standardized up till now. For this reason, we envision to create the identification using the naming technique depicted in Shi et al. [65] which encompasses location (**where**), role (**who**), and data description (**what**). For instance, the id value "*UFRJ.UbicompLab.temperature*" identifies the descriptor used by the VN to provide raw data of temperature from the Ubicomp laboratory at our university, UFRJ. Indeed, this friendly naming style allows users to identify the data type descriptors registered in the catalog easily using a search mechanism.

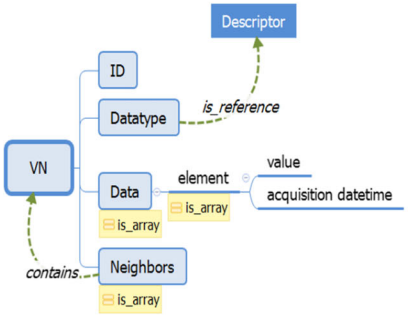
The *type* is described using three pre-defined values, namely: **simple**, **complex** and **actuation**. The **simple** type means that the information provided is a raw data (e.g., temperature), the **actuation** means an actuation operation (e.g., turning on/off a lamp), and **complex** represents a workflow. Regarding both **simple** and **actuation** types, each $\boldsymbol{element_i} \mid i \geq 1$ represents a physical device used to either get sensing data (sensor device) or perform actuation (actuator device) respectively. Concerning the *complex* type, $element_i \mid i = 1$ represents only an instance of the workflow. The workflow is used to specify a data flow containing other data types (either simple or complex) that VN instances use to get data (either raw or processed) as input, perform some processing, and generate information as output. These tasks are similar to the processing performed in a Complex Event Processing (CEP) engine [19]. Therefore, we envisioned using a CEP engine for executing the workflow. Nevertheless, the creation of the workflow is out of the scope of our current work.

3.2.3 The virtual nodes

In our virtualization model, the **Virtual Node** (VN) is the core computational unit. Our virtualization model was designed to circumvent two drawbacks identified in the model adopted in Olympus [61]. The first drawback concerns the design of the VN and the second is the lack of collaboration between VNs. Furthermore, our model design is simplified based on the integration of lightweight virtualization [47, 48] approach and the microservice pattern [42, 70], thereby defining VNs as "**lightweight-nodes**". This new VN is provided as a microservice as it is small, high decoupled, and owns a single responsibility. Moreover, the VN is packaged in lightweight-images thereby facilitating its distribution and managing at the edge nodes.

Figure 5 Type of Virtual Nodes





(a)


```
{
  "ID": "EN1.UFRJ.UbicompLab.temperature",
  "Datatype": {
    "id": {
      "where": "UFRJ",
      "who": "UbicompLab",
      "what": "temperature"
    },
    "description": "Laboratory Temperature (celsius)",
    "type": "simple",
    "element": [
      {
        "value": "DeviceID1"
      }
    ]
  },
  "Data": {
        "DeviceID1": [
          {
            "value": "18.0",
            "acquisitiondatetime": "2018-03-29 13:25"
          }
        ]
  },
  "Neighbors": []
}
```


(b)

Figure 6 **a** The representation of the Virtual Node. **b** Virtual Node example

The first drawback is tackled by simplifying the VN concept as proposed by Olympus. Our VN is defined as an abstract class (Figure 5) providing common attributes and functionalities to design predefined types of VNs for each provided datatype. A datatype can represent either a raw data, or a processed data (representing a complex event or the result of a data fusion procedure), or an actuation result. Therefore, there is a VN to represents the simple datatypes (SensingVN), a VN to represent the actuation datatypes (ActuationVN), and a VN to represent complex datatypes (DatahandlingVN). They are depicted further below.

$$VN = (ID, Datatype, Data, Neighbors) \tag{2}$$

The second drawback is fulfilled by extending the VN concept to allow the collaboration among VNs for the purpose of data sharing using the collaboration process.

Per definition (2), we describe each VN (Figure 6a) as a tuple in terms of the properties ID, Datatype, Data, and Neighbors. *ID* is the virtual node identification; *Datatype* is a link to the datatype descriptor, and it is defined as a tuple $is\_referenc = R_a$, where $R_a = id$ representing the unique Descriptor identifier; *Data* represents the resources provided by the VN; and the $Neighbors = \{vn_1, \ldots, vn_n\}$ represents a set of neighboring VN connected to the VN that shares the same data (values). Figure 6b represents a VN providing a *simple* datatype of temperature.

The *ID* is a relevant property used to identify the VN instances into the repository in each edge node. In the same way as in the datatype identification, we need of an efficient and friendly naming schema for the VN identification that facilitates the collaboration. Therefore, we describe the identification of the VN as a tuple $VNID = \{hostname, Descriptor(id)\}$, where *hostname* is the unique label (assigned by the network administrator) identifying the edge node connected to a network, and *Descriptor*(*id*) denotes the datatype identification provided by the VN. This decision aims to facilitate the collaboration during the task of identifying which edge node the VN is executing since the same datatype can be provided on different edge nodes.

Let's use the prior datatype identification as an example, "*UFRJ.UbicompLab.temperature*". Considering that our system can provide the datatype in the edge nodes EN1 and EN2, the related IDs will be "***EN1***.*UFRJ.UbicompLab.temperature*" and "***EN2***.*UFRJ.UbicompLab.temperature*".

Regarding the resources provided by the VN (*Data* property), it can be either raw data (e.g., temperature, humidity, luminosity, presence, etc.) or data processed by a CEP engine, such as the description of an event of interest as, for instance, a Fire Detection. Finally, the *resource* can also be information relating to the result of the actuation command (actuation type). Moreover, the *acquisition datetime* represents the date and time in which a data has been acquired. Hereafter, we describe the predefined types of VNs.

The VN of the type **Sensing** represents the *simple* datatype and provides a stream of raw data sensed by one or many physical nodes. This VN has a set of properties $p : p = (simple, fr, sr, callback)$. *Simple* property denotes the information to process, *fr* means the data freshness threshold in milliseconds, *sr* the sampling data rate, and *callback* is an operation invoked by the VN to deliver data to the request issuer whenever it finishes the processing. The stream of raw data is retrieved either from historical databases maintained at the edge tier or by a direct connection with the physical nodes at the sensor tier. However, the latter option depends on the value of the data freshness property.

The VN of the type ***Actuation*** provides actuation capabilities over the physical environment and has a set of properties $p : p = (command, callback)$. *Command* denotes the type of actuation function provided by the VN, and the *callback denotes* an URL address used by the VN to notify the request issuer when the operation finalizes along with its status (e.g., processed or not).

The VN of the type ***Data handling*** is used to provide value-added information to meet user requests by employing information fusion techniques [60]. This is done by executing predefined queries on sensing data using a Complex Event Processing (CEP) engine. The VN represents the *complex* datatype and has a set of properties $p : p = (complex, callback)$. *Complex* represents the query to be executed, and *callback* is used by the VN to notify the request issuer when the CEP has completed its operation along with the query response (data stream).

In this Section, we presented the virtualization model, its main components, and the relationship between them. Moreover, we described the Virtual Node, the application request and the Datatype descriptor. In the next Section, we describe the collaboration process.

3.3 Collaboration process

We designed a novel flat P2P collaboration process entirely distributed at the edge tier. This process is used in two activities of the CoT infrastructure operation. First, in the data sharing activity that allows a VN to actively share its fresh data with neighboring VNs. Second, in the distributed resource management that provides for each edge node the capability of decision-making to engage neighboring edge nodes to allocate or provision VNs whenever it is necessary.

Our approach uses the flat P2P model in opposition to the hierarchical model [35] to overcome the drawback regarding hierarchical models and avoid the high dependence on the master node (or controller node – a single point of failure). Also, we avoid the communication overhead to create and maintain the hierarchy. Therefore, our VNs and resource management are independent of each other from an operational point of view, i.e., each edge node has its own resource management to allocate or provision one or many VNs. By following our approach, the edge nodes are homogeneous regarding their functionalities even though heterogeneous regarding their capabilities.

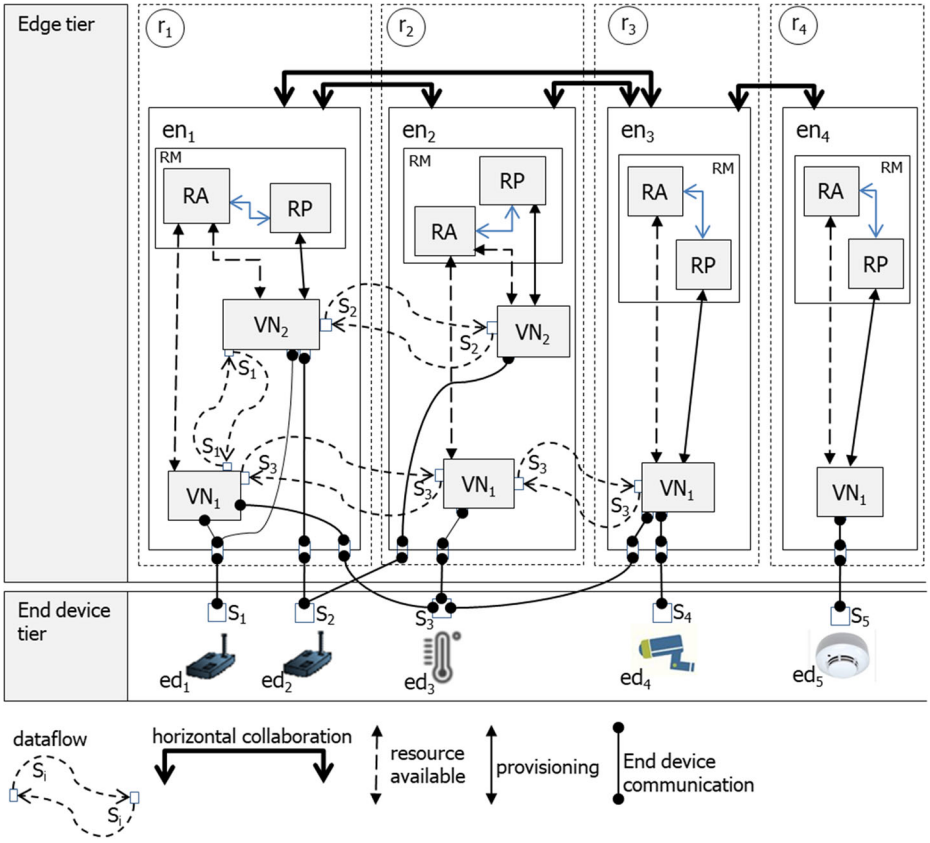


Figure 7 A flat P2P collaboration for data sharing and resource management

A challenge regarding our approach is how to organize the edge nodes in a neighborhood to enable the collaboration process. Therefore, we define the neighborhood concept using (i) the closeness between the edge nodes and (ii) the semantics of the data provided by them. The geographic closeness is used to take advantage of the location-aware capability of the edge nodes and helps decreasing the latency and bandwidth consumed between the edge nodes. The data semantics is used to determine which data the VNs can share with each other to reduce the access and the data transmissions between end devices and edge nodes, thereby saving energy and bandwidth. Thus, we need to apply techniques to identify which edge nodes can be neighbor to each other and which data to be shared.

In the first case (closeness), we adopted the Weighted Voronoi Diagrams (WVD) [36] to divide a geographic area (Edge network) into regions to promote the collaboration among edge nodes. The WVD was adopted as a solution to build the neighborhood because it is widely used in the areas of GIS (Geographic Information Systems), sensor networks and wireless networks for making placement decisions. To use the WVD algorithm, we need to provide the edge nodes geographic locations on the map. This location in the WVD is called "site." Moreover, the algorithm allows setting the weight for each site that represents a value determining the distance between the site A and site B, for instance. As our approach focuses

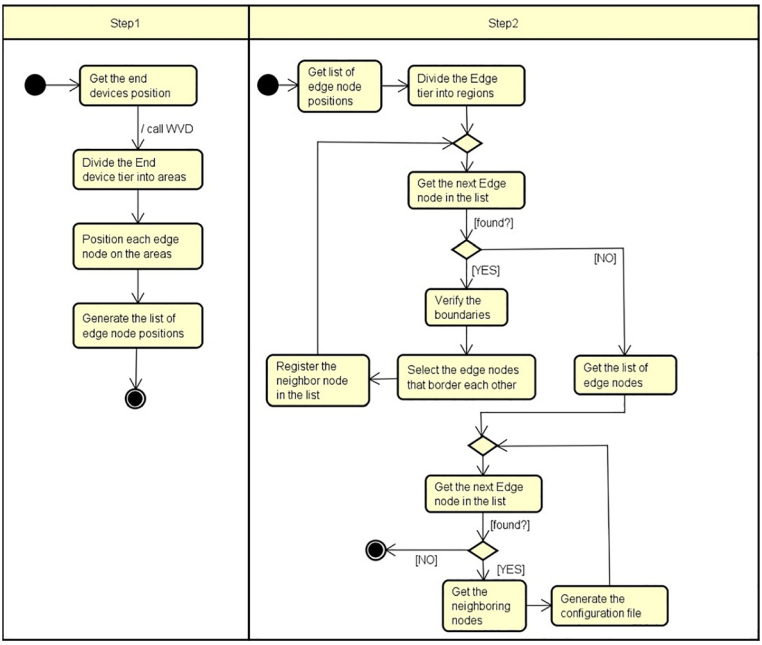


Figure 8 The setup process to generate the neighborhood

on each edge node belonging to the Edge network which is connected to the underlying network of *end devices*, our problem *P* is how to divide the Edge network into regions of edge nodes to establish the neighborhood. Figure 7 illustrates our collaboration process.

To better understand its operation, we will detail how the essential elements are distributed both at End device tier and Edge tier besides how the neighborhood is defined. The specifications of the Edge and *End device* tiers regarding their elements are the same presented in Section 3.1. Regarding the End devices tier, our example encompasses the *end devices* ed_1, ed_2, ed_3, ed_4 *and* ed_5. The Edge tier encompasses four edge nodes en_1, en_2, en_3 *and* en_4. Each edge node en_i is connected to one or many *end devices*. In our example, en_1 is connected to *end devices* ed_1, ed_2 *and* ed_3; en_2 is connected to *end devices* ed_2 *and* ed_3; en_3 is connected to *end devices* ed_3 *and* ed_4; and, en_4 is connected to *end devices* ed_5.Moreover, each edge node en_i can host and run one or many virtual nodes $VN = \{vn_1, vn_2, \ldots, vn_n\}$. Each vn_i can provide sensing data obtained from or perform actuation on one or many *end devices*. For instance, in the $en_1$the VN_1 is connected to the End device $ed_1$and ed_3 whereas the VN_2 is connected to the End device ed_1 and ed_2.

In this work, we provide a process to support the Service Provider Manager (SPM) activities regarding the setup of the collaboration among edge nodes. The setup encompasses two main steps: (i) dividing the End device tier into areas and positioning the edge nodes inside these areas, (ii) building the neighborhoods. The activity diagram in Figure 8 illustrates the process.

The SPM executes the first step when there are new *end devices* and/or edge nodes to be integrated into the CoT infrastructure. Thus, the SPM can determinate the best position for each edge node (at the Edge tier) regarding the *end device*. Initially, we assume that the best localization for each edge node is the closest to the *end devices* which it will be connected to, thereby minimizing the latency of communication between them. In order to achieve this goal,

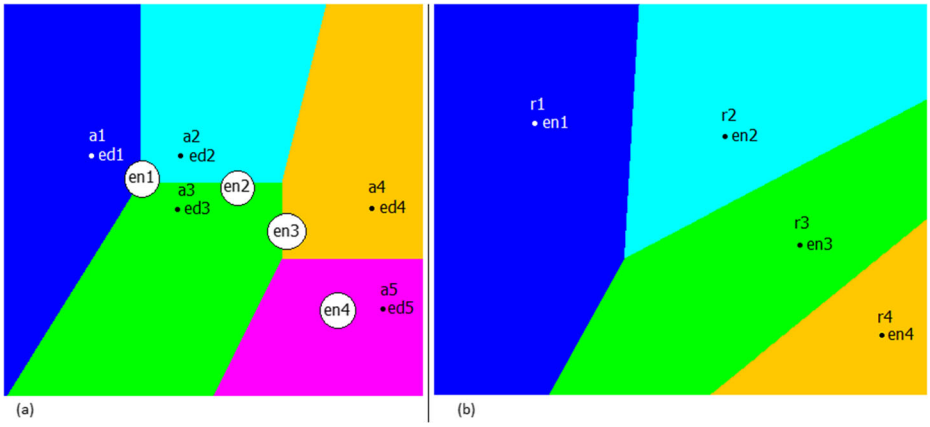


Figure 9 **a** The WVD of the End device tier with the edge nodes positioned. **b** The WVD with four edge nodes as site

the SPM obtains the position of each *end device* at the End device tier (EdT). Next, the WVD algorithm is invoked to divide the EdT into areas $A = \{a_1, a_2, \ldots, a_n\}$ using the *end device* localization as site. Thus, each area $a_i \in [1, A]$ can contain one or many edge nodes. Lastly, the SPM places each edge node on the areas and generates a list of the edge nodes positions. Figure 9a presents the WVD with the edge nodes positioned. In our example, we placed the en_1 on the boundary of the areas a_1, a_2, *and* a_3 to enable the connection to the *end devices* ed_1, ed_2 *and* ed_3. The en_2 is placed on the boundary of the areas a_2, *and* a_3 to enable the connection to the *end devices* ed_2 *and* ed_3. The en_3 is placed on the boundary of the areas a_3, *and* a_4 to enable the connection to the *end devices* ed_3 *and* ed_4. Finally, the en_4 is placed on the area a_5 to enable the connection to the *end devices* ed_5. Therefore, the position of each edge node respects, besides the distance, which type of data is provided by the *end device(s)*.

The second step of the setup process can be executed independently of step one. The SPM uses this step to determine the neighborhoods. Upon getting the location of each edge node at the Edge tier (ET), the SPM invokes the WVD algorithm to divide the ET into regions $R = \{r_1, r_2, \ldots, r_n\}$. Moreover, the edge node localization is used as the site, and each site has the weight(s) determined by a distance D, where $D = d_{lm}$ represents the distance between en_l and en_m. Also, each region $r_i \in [1, R]$ only contains one edge node. Next, for each edge node en_i, the border of its area is verified regarding the other edge nodes. So, the edge nodes that border a given edge node $en_i$are selected and included in its list of neighboring nodes. For instance, Figure 9b illustrates the ET divided into four regions by the WVD algorithm, and the **border between them defines the neighborhoods**. Thus, the edge node en_1 has en_2 *and* en_3 as neighbors. The edge node en_2has en_1 *and* en_3 as neighbors. The edge node en_3 has en_1, en_2 *and* en_4 as neighbors. Lastly, the edge node en_4 has the edge node en_3 as neighbor. After defining the neighborhoods, for each edge node, a configuration file containing its neighboring nodes is generated, and then the setup process finalizes.

From now onwards, the virtual nodes (VN) can collaborate with each other for data sharing by means of a dataflow approach. In the dataflow programming model, the application is represented as a graph [32]. In the graph, the nodes can be (i) data producers, (ii) data consumers, and (iii) processing units (or intermediary). The producer node only produces outputs and represents the start of the dataflow. The consumer node only consumes inputs provided by another node and represents the end of the flow, whereas the processing units are

nodes developed for processing the inputs and produce outputs. Hence, in the context of the dataflow model, we can classify the VN as a processing unit, since our VN is able to receive/provide sensed data from/to its neighboring VNs.

In addition to the approaches and techniques before mentioned, we also need to establish which data a VN can share with its neighbors, thereby defining the semantic collaboration. The data sharing may be either total or partial. In the first case, the VN sends all its data to the neighboring VNs. In turn, in the partial model, the VN sends only the sensing data related to the end device(s) (e.g., a sensor) connected to neighboring VNs. For instance, in the model of Figure 7, the VN_2 of $en_1$is connected to *end devices* ed_1 *and* ed_2 whereas the VN_2 of $en_2$is connected to the ed_2. Thus, VN_2 of the en_1 sends only the sensing data (S_2) related to ed_2 for the neighboring VN_2 of en_2 and vice-versa. Therefore, our model avoids the communication overhead by sending only the necessary data between the VNs.

Furthermore, our collaboration process allowed designing a novel model of distributed resource management (RM). This model takes advantage of the best features of the flat P2P collaboration so that each edge node has the capability of decision-making to allocate the best resource to meet the application request. Thus, whenever the edge node does not have enough resources to instantiate the VN, the RM will identify a neighboring node with available resources to provision a new VN. However, if a neighbor edge node is not able to be selected, the application request is refused.

In this Section, we presented our collaboration process. It focuses on meeting the requests of emerging IoT applications besides providing a model where both the virtual nodes and the resource management components are independent of each other from an operational point of view. To achieve the collaboration goal, initially, a process was presented to create the neighborhood of edge nodes. Next, we described the main features of the data sharing between virtual nodes and the decentralized resource management between edge nodes. In the next Section, we present heuristic algorithms implementing our distributed resource management and P2P collaboration with data sharing.

4 Heuristic algorithms for the distributed resource management and P2P collaboration

In this section, we present the proposal of four heuristic algorithms implementing our distributed resource management and flat P2P collaboration processes. We are using a heuristic method since our problem involves placement decisions of which edge node will meet the applications requests to maximize utility regarding response time (latency) and bandwidth consumed. According to Xu et al. [75], this class of problem is NP-Hard, and the heuristic algorithms are often used to solve it [41, 44]. We define some premises to drive the algorithm development, that are: (i) each edge node has its datatype repository, and all components of the architecture deployed; (ii) the information about which is the neighbor node of a specific node is already available, and it was generated using the second step of the setup process (Figure 8); (iii) during the collaboration, the perimeter of actuation should be limited by only looking at the neighboring nodes regarding the node in which the request arrived; (iv) the application requests a single datatype; (v) the requests arriving in the system are served immediately and in order of arrival; (vi) the resource availability of the edge node should always be checked before provisioning the new virtual node; (vii) the resource availability of the VN should always be checked before meeting the request; (viii) the infrastructure should try to scale-up the resources

of the VN to meet more requests before provisioning another VN; (ix) the VN identification (VNid) is unique and should be combined using the datatype id and edge node hostname aiming to facilitate the collaboration process.

Our resource management is based on the algorithms proposed by Wang et al. [72] and also by Hong et al. [35] with several modifications described as follows. First, we have modified the model from hierarchical to flat for addressing the collaboration described in Section 3.3. Second, we have enhanced the resource management with P2P collaboration (Section 4.3). The third modification concerns how the edge nodes deal with the application requests. To handle the requests, Wang et al. [72] enable workload offloading from cloud servers to run on the edge nodes. Our proposal adopts a different approach: the infrastructure meets the applications requests by allocating virtual nodes running at the edge tier. Thus, we save bandwidth consumption and avoid network overhead, thereby favoring the execution of the emerging IoT applications that are latency-sensitive, for instance.

A remarkable feature of our algorithm is to work on-demand. As user application requests arrive in any edge nodes, the Resource Allocator (RA) component (Section 4.1) receives the requests and allocates virtual nodes (VN) to meet them. The Resource Provisioner (Section 4.2) only provisions a new VN instance whenever it is requested by the RA. As the edge nodes are resource constrained devices, this strategy avoids the additional provisioning of VNs, thereby saving resources and avoiding affecting their original (primary) functions, for instance, routing whenever the edge node is a smart router. Furthermore, we limit the actuation perimeter of our algorithm only to look at neighboring nodes regarding the node in which the request arrived. Therefore, in this P2P collaboration, the requests are forwarded only

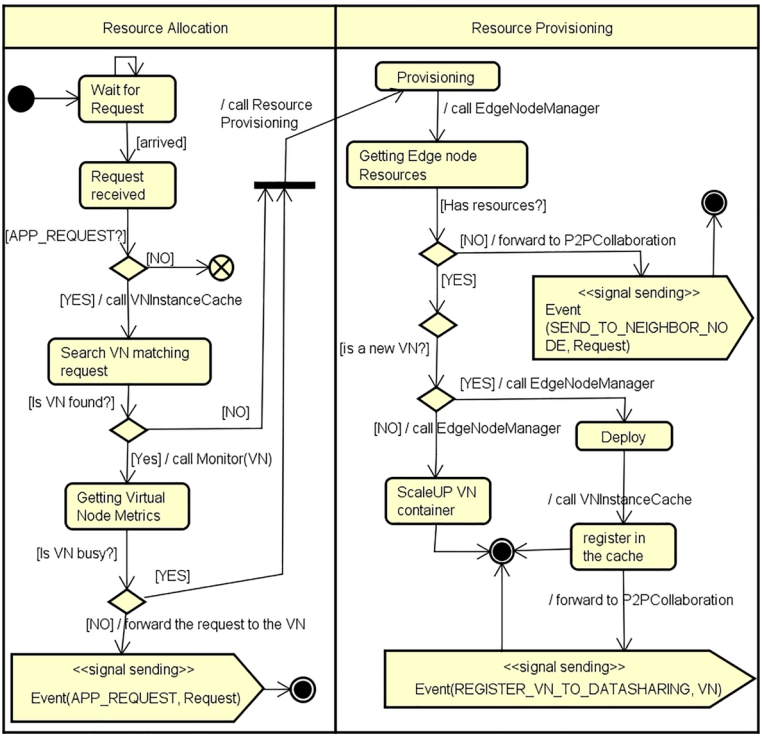


Figure 10 Resource management activity diagram

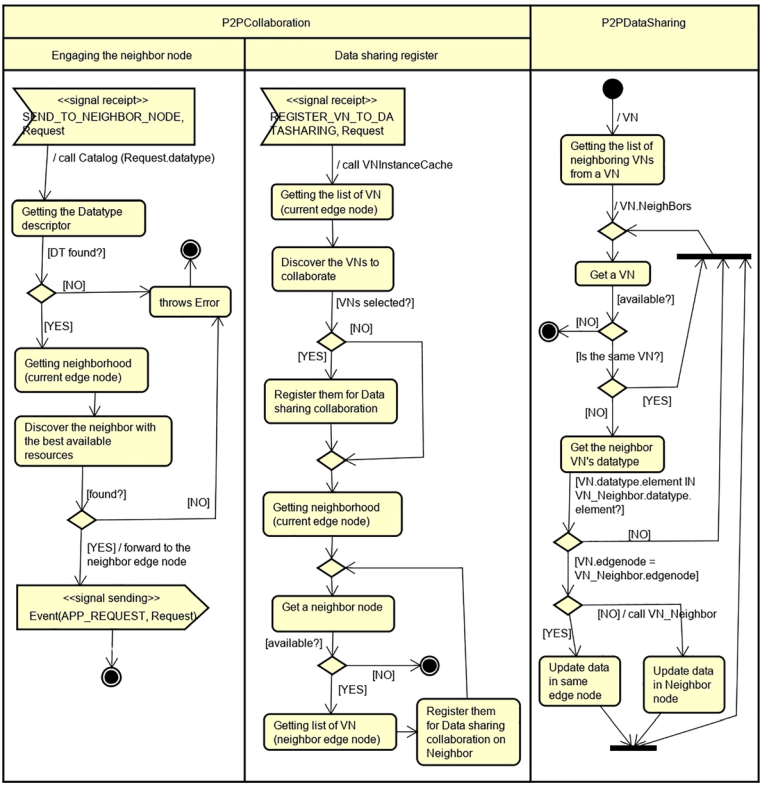


Figure 11 P2P Collaboration and P2P Data sharing activity diagram

to one hop regarding the entry edge node. To best understand our algorithms, we present two UML activity diagrams that describe their primary activities and the relationship between them. Figure 10 represents the distributed resource management logic (Sections 4.1 and 4.2) and Figure 11 the P2P collaboration (Sections 4.3 and 4.4). They are explained in details in the next Sections.

4.1 Resource allocation

Resource Allocation is a process in charge of executing the tasks to allocate resources necessary to meet the workloads of one or several applications [22]. In our virtualization model, the Resource Allocator (RA) component is responsible for performing this process. Following the virtualization approach, it has the main tasks of finding and allocating virtual nodes (VNs) to meet application requests. Moreover, the RA can invoke the Resource Provisioner component to provision new VNs whenever it is necessary. **Algorithm1** implements all the logic steps of our RA. The RA listens for incoming requests from the application (lines 21–30). Upon receiving the request, the RA verifies the type of message, and if it is *APP_REQUEST*, the *handleRequest* procedure is executed to meet it (line 1).

The *handleRequest* procedure performs a search in the instances repository for a Virtual Node (VN) matching the received request (line 3). If no VN exists matching the request, the RA invokes the Resource Provisioner (RP) to provision a new VN instance (lines 4–5).

Moreover, if the selected VN has its resources busy managing other requests, the RP is also invoked (lines 7–10) either to increase the VN resources (e.g., memory) or to provide another VN. As VN instances are chosen (or provisioned), the RA forwards the requests to the respective VN to process them (lines 12–16). It is worth noting that the catch command (line 17) is responsible for capturing any processing error whereas the throw exception (line 18) is the command in charge of throwing the error messages. These commands have the same goals in all algorithms.

Algorithm 1: ResourceAllocator

```
   Initialization: this.run();
   // Finding an instance of Virtual Node matching with the request
 1 private procedure handleRequest(Request)
 2   try
 3     vnInstance ← call VNInstanceCache.search(Request.datatype);
 4     if (vnInstance == null) then
 5       vnInstance ← call ResourceProvisioner.provisioning(null, Request);
 6     else
         // Getting runtime metrics' object from the selected Virtual Node
 7       metrics ← call Monitor.getVirtualNodeMetrics(vnInstance);
 8       if (metrics.isResourceBusy()) then
 9         vnInstance ← call ResourceProvisioner.provisioning(vnInstance,
            Request);
10       end
11     end
12     if (vnInstance ≠ null) then
13       asynchronous
14         sendMessage(vnInstance, Event(APP_REQUEST, Request));
15       end
16     end
17   catch(e)
18     throw exception (e.message);
19   end
20 end
21 private procedure run()
22   asynchronous
23     while (true) do
24       event ← receiveEvent();
25       if (event.type == APP_REQUEST) then
26         this.handleRequest((Request)event.message);
27       end
28     end
29   end
30 end
```

4.2 Resource provisioning

In the context of this paper, Resource Provisioning is the process responsible for provisioning new Virtual Nodes (VN) or empowering the VN with extra resources to meet new received requests. The Resource Provisioner (RP) is the component in charge of performing this process. **Algorithm2** implements our RP logic. The algorithm starts when the Resource Allocator invokes the RP through the *provisioning* function passing two parameters, namely *CurrentVirtualNode* and *Request* (line 1).

Initially, the RP invokes the *EdgeNodeManager* component to verify the resources of the edge node for hosting a new VN instance (line 2). *EdgeNodeManager* is the component in charge of providing a set of methods for abstracting the access to physical information of the edge nodes besides controlling the VN life-cycle. When the edge node has enough resources available (line 3), the next step is either deploying a new VN container or to re-configure (scale-up) an existing one. In case of *CurrentVirtualNode* parameter is *null* (line 4), the RP instantiates a new VN by invoking the *containerDeploy* method from the *EdgeNodeManager* component (line 5). Next, the new VN is cached and also registered for purposes of data sharing collaboration (lines 6 and 8). At the end, the VN instance is returned (line 10). When a *CurrentVirtualNode* parameter is not *null,* the scale-up method from the *EdgeNodeManager* component (line 12) is invoked to increase the VN container with extra resources for improving its performance to meet more requests. If the scale-up worked fine, the current virtual node instance is returned (lines 13–14). Otherwise, if the scale-up fails or the container does not have enough resources, the RP forwards the request to the **P2P collaboration** for finding a neighbor edge node capable of meeting it (line 20). The RP always return the *null* value when a VN instance cannot be created (line 22).

Algorithm 2: ResourceProvisioner

```
1  function provisioning(CurrentVirtualNode, Request) : VirtualNode
2      hasResource ← call EdgeNodeManager.hasResource(Request.datatype);
3      if (hasResource) then
4          if (CurrentVirtualNode == null) then
               // Deploy a new Virtual Node container
5              vnInstance ← call EdgeNodeManager.containerDeploy(Request.datatype);
               // Registering the new instance of the VN in the cache
6              call VNInstanceCache.register(vnInstance);
               // Registering the new VN for data sharing collaboration
7              asynchronous
8                  sendMessage(P2PCollaboration,
                   Event(REGISTER_VN_TO_DATASHARING, vnInstance));
9              end
10             return vnInstance;
11         else
               // Reconfigure the Virtual Node container
12             result ← call EdgeNodeManager.scaleUp(CurrentVirtualNode);
13             if (result == true) then
14                 return CurrentVirtualNode;
15             end
16         end
17     end
       // Collaboration with the neighborhood for finding an Edge Node to meet the request
18     try
19         asynchronous
20             sendMessage(P2PCollaboration, Event(SEND_TO_NEIGHBOR_NODE,
               Request));
21         end;
22         return null;
23     catch(e)
24         throw exception (e.message);
25     end
26 end
```

4.3 P2P collaboration

P2P collaboration component implements the essential services to enable the collaboration process at the Edge tier. Among the services, we can mention: (i) forwarding of requests to be served by a neighboring edge node, (ii) determining the best neighboring edge node to meet a request and (iii) registering a VN to enable the data sharing collaboration. **Algorithm3** implements the logic of these services. During the initialization process, there is a call to the **EdgeNodeManager** component to get the edge node neighborhood and the needed resources to deploy a VN. Next, the run

Algorithm 3: P2PCollaboration

```
Initialization: Neighborhood ← call EdgeNodeManager.edgenode.neighborhood;
                neededRes ← call EdgeNodeManager.neededRes;
                this.run();
1  private procedure sendToNeighborNode(Request)
2      try
3          neighborEdgeNode ← getNeighborNode(Request);
4          sendMessage((neighborEdgeNode)→ResourceAllocator, Event(APP_REQUEST,
           request));
5      catch(e)
6          throw exception (e.message);
7      end
8  end
9  private function getNeighborNode(Request) : EdgeNode
10     datatype ← call Catalog.getDescriptor(Request.datatype.id);
11     CandidateNeighborNodes ← ∅;
12     foreach (NeighborNode ∈ Neighborhood) do
13         hasConnectedDevices ← NeighborNode.hasConnectedDevices(datatype);
14         if (hasConnectedDevices) then
15             CandidateNeighborNodes.add(NeighborNode);
16         end
17     end
18     BestNeighborNode ← bestResource(Request.datatype, CandidateNeighborNodes);
19     if (BestNeighborNode == null) then
20         throw exception ("No neighboring node available!");
21     end
22     return BestNeighborNode;
23 end
24 private function bestResource(Datatype, CandidateNeighborNodes) : EdgeNode
25     if (CandidateNeighborNodes == ∅) then
26         return null;
27     end
28     needMem ← neededRes.get(Datatype.type).getPhysicalMemorySize();
29     BestCandidateNeighborNodes ← ∅;
30     foreach EdgeNode ∈ CandidateNeighborNodes do
31         EdgeNodeRes ← call (EdgeNode)→Monitor.getNodeResources();
32         EdgeNodeRes.EdgeNode ← EdgeNode;
33         EdgeNodeRes.Latency ← getLatency(EdgeNode);
34         if (needMem ≤ EdgeNodeRes.Resources.FreePhysicalMemorySize) then
35             BestCandidateNeighborNodes.add(EdgeNodeRes);
36         end
37     end
38     if (BestCandidateNeighborNodes == ∅) then
39         return null;
40     end
41     Collections.sort(BestCandidateNeighborNodes).orderby CPU, memory available, latency;
42     return BestCandidateNeighborNodes[0].EdgeNode;
43 end
```

method (line 56) is performed to listen for incoming events sent by the Resource Provisioner (lines 60 and 63) and EdgeNodeManager (line 66).

Upon receiving the event, the algorithm verifies the type of event. When the event is *SEND_TO_NEIGHBOR_NODE*, then the *sendToNeighborNode* method (line 1) is executed. The *sendToNeighborNode* selects a neighboring node through the private method *getNeighborNode* (line 3), and then forwards the Request to the neighboring ***Resource Allocator*** (lines 4). The *getNeighborNode* (lines 9–23) is used to provide a neighboring node with the best resources for hosting and running the virtual node. Initially, the method seeks an instance of the datatype descriptor in the repository by invoking the ***Catalog*** component using the id property (from the *Request* object) (line 10). Next, the candidate neighboring nodes are selected, i.e., the nodes that provide the requested data (lines 12–17).

Next, the **bestResource** private method is invoked to find the best neighbor among the candidate edge nodes belonging to its neighborhood (line 18). In case of the best neighbor does not exist, an exception message is thrown (lines 19–21). Initially, when the **bestResource** method runs (line 24), it obtains the amount of memory required to process the request (line 28). Next, for every candidate node, its available physical resources (e.g., memory, CPU) are obtained (line 31) using the monitor component. Such memory information is used to verify if the candidate node has enough available memory to process the received request (line 34). When the candidate node has

```
   // P2PCollaboration continue...
44 private procedure registerVNtoDataSharing(NewVirtualNode)
   |  // inside of the same node
45 |  VirtualNodesInstances ← call VNInstanceCache.getListInstances();
46 |  foreach Vn ∈ VirtualNodesInstances do
47 |  |  VerifyIntersection(NewVirtualNode, Vn);
48 |  end
   |  // verifying in the neighboring nodes
49 |  foreach NeighborNode ∈ Neighborhood do
50 |  |  VirtualNodesInstances ← call (NeighborNode)→VNInstanceCache .getListInstances();
51 |  |  foreach Vn ∈ VirtualNodesInstances do
52 |  |  |  VerifyIntersection(NewVirtualNode, Vn, NeighborNode);
53 |  |  end
54 |  end
55 end
56 private procedure run()
57 |  asynchronous
58 |  |  while (true) do
59 |  |  |  event ← receiveEvent();
60 |  |  |  if (event.type == SEND_TO_NEIGHBOR_NODE) then
61 |  |  |  |  this.sendToNeighborNode((Request)event.message);
62 |  |  |  else
63 |  |  |  |  if (event.type == REGISTER_VN_TO_DATASHARING) then
64 |  |  |  |  |  this.registerVNtoDataSharing((VirtualNode)event.message);
65 |  |  |  |  else
66 |  |  |  |  |  if (event.type == RELOAD_CONFIG) then
67 |  |  |  |  |  |  Neighborhood ← call EdgeNodeManager.edgenode.neighborhood;
68 |  |  |  |  |  |  neededRes ← call EdgeNodeManager.neededRes;
69 |  |  |  |  |  end
70 |  |  |  |  end
71 |  |  |  end
72 |  |  end
73 |  end
74 end
```

available physical memory to process the request, it is selected and stored into the list of the best candidate nodes (lines 34–36), otherwise, it is discarded. Moreover, network latency also is measured (line 33). Then the list of candidate edge nodes is sorted based on the criteria of the number of available CPUs, free memory and lower latency. In the end, we find and select the node with the best resources on top of the list (line 42). Whenever the received *event* is *REGISTER_VN_TO_DATASHARING*, the *registerVNtoDataSharing* method is executed (line 44). This method is responsible for registering a new virtual node instance to participate of the data sharing process. First, the process executes the register inside of the same edge node (lines 45–48). Next, the register occurs on the neighboring nodes that can collaborate (lines 49–54), i.e., the neighboring edge nodes running Virtual Nodes that provide the same datatype elements. In order to check if a VN can collaborate, the ***VerifyIntersection*** private method is used to check the intersection of the elements of the *NewVirtualNode* and existing *VirtualNode*.

Finally, when the received event is *RELOAD_CONFIG*, the settings related to the neighborhood and resources needed are reloaded (line 67 and 68). This event is essential when new data about the neighborhood are made available (insertion or exclusion of neighbor nodes) through a manual or dynamic process.

4.4 P2P data sharing

P2P Data Sharing is responsible for enabling a Virtual Node (VN) to share data with its neighboring VNs. **Algorithm4** implements the logic of this service.

Algorithm 4: P2P Data Sharing

```
1  procedure shareData(VirtualNode, element, data)
2      foreach VN_NeighBors ∈ VirtualNode.NeighBors do
           // Avoiding cross-reference between virtual nodes
3          if (VirtualNode ≠ VN_NeighBors) then
4              if (element ∈ VN_NeighBors.Datatype.element) then
5                  NeighborNode ← VN_NeighBors.getHostName();
6                  if (VirtualNode.getHostName() == NeighborNode) then
7                      VN_NeighBors.setData(element, data);
8                  else
9                      asynchronous
10                         call (NeighborNode)→VN_NeighBors.setData(element, data);
11                     end
12                 end
13             end
14         end
15     end
16 end
```

The VN calls the *shareData* method to share fresh data obtained directly from (physical) sensor devices or received from other VNs (line 2–15). The data sharing occurs only among elements of the same type (line 4), i.e., according to the data sharing constraint presented in Section 3.3. When the previous restriction is valid, and the neighboring VN belongs to the same edge node, the data are share locally (lines 6–7); otherwise, the data are sent to the VN in the neighboring edge node (line 10).

In this section, we presented a set of algorithms that implement our distributed resource management and the flat P2P collaboration. First, we detailed a set of premises used to drive the implementation. Next, we presented two algorithms found in the literature that served as

the base for our solution as well as the types of adaptations needed in these algorithms to support our model. Last, we presented the four main algorithms and a detailed explanation of their operations.

5 Evaluation

We used the Goal Question Metric (GQM) methodology to help in the design of the performed evaluation. The GQM [7] is a hierarchical approach (three levels) based on the premise of deriving software metrics from questions and goals. According to the methodology, first, we have to define the goals. The goal represents the conceptual level related to the measurement purpose (what object and why), the perspective (what aspect and who), and the environmental characteristics (where). Second, each goal needs to be refined into several questions. A question represents the operational level used to define the object of measurement (product, process, resource). Finally, the metrics represent the information at the quantitative level that should be collected to answer one or many questions. According to Basili et al. [7], the metrics can be either objective or subjective. Objective metrics represent data that depends only on the object that is being measured and not on the point of view of who is measuring it. Subjective metrics represent data that depends on the object that is being measured as well as the point of view of who is measuring it.

In Section 5.1, we describe the goals, questions, and metrics defined to be used in our evaluation. In Section 5.2, we describe the overall parameters of the scenarios considered in our experiments. In Section 5.3, we describe the three experiments performed to answer each of the questions defined using the GQM methodology, including the responses to the defined questions.

5.1 Goals, questions and metrics

The goals, questions, and the derived metrics used in our work are presented as follows.

Goal G1 is to **analyze** LW-CoEdge, **for the purpose of** evaluating its resource management algorithms, **with respect to** the performance in meeting application requests, **from the viewpoint of** the end-user.

Goal G2 is to **analyze** LW-CoEdge, **for the purpose of** evaluating its collaboration mechanisms, **with respect to** the impact of meeting a request through a neighboring edge node regarding the bandwidth, the latency of communications and energy saving, **from the viewpoints of** both the end-user and the underlying computational infrastructure.

Table 2 GQM Questions

Question	Goal (G)
Q1 Does the proposed approach for collaborative Resource Management help meeting all application requests submitted to the system?	G1
Q2 Does the collaboration among virtual nodes help decreasing the data traffic between edge nodes and *end devices*?	G2
Q3 Does the collaboration among virtual nodes help saving energy of the *end device* nodes?	G2
Q4 Does collaboration between virtual nodes help meeting requests within the application-defined response time threshold?	G2

Table 3 Summary of the metrics used to answer the questions

Metric (M)	Description	Question
M1 REQ_MET	Number of requests met.	Q1
M2 REQ_NOT_MET	Number of requests not met.	Q1
M3 AVG_DT_EN	Average of data traffic generated among edge nodes during the collaboration.	Q2
M4 AVG_DT_ENED	Average of data traffic generated between edge nodes and *end devices*.	Q2
M5 ED_ENERGY	End node energy consumption.	Q3
M6 REQ_RTTH_INV	Total of requests met that violated the response time threshold desired by the application.	Q4
M7 AVG_DT_TRAFFIC_KB	The average in KB of data traffic to meet a request.	Q2

From these two goals, we raise four relevant questions (Q1, Q2, Q3, and Q4) described in Table 2 . It is important to mention that the performed evaluation always considered as benchmark for comparison purposes a version of the resource management process with no collaboration among edge nodes.

Table 3 shows a set of metrics that are collected from our CoT system. We used these metrics to answer our posed research questions. To answer the question Q1, we classified application requests within two groups: (a) request met, and (b) request not met. Our system considers that a request is met whenever the CoT system provides the required datatype and the Response Time Threshold (RTTh) desired by the application is satisfied. It is worth noting that the system delivers the results to the request issuer even if the RTTh desired by the application is not respected. The RTTh is an important QoS parameter and depends on the type of application. In PubNub Staff [58], a discussion about the impact of the Real-time applications and the response time desirable is performed. In Yi et al. [77], the authors argue that augmented reality and real-time video analytics are two examples of applications in which a high response time (more than tens of milliseconds) can affect the user experience. Moreover, network latency is a factor that affects the response time of the applications and also needs to be measured.

Table 4 Some parameters used in experiments

Parameter	Value
Edge node core speed	3 × 106 cycles/s
Number of cores of each EN	2 cores
Delay for querying local database (FiWARE)	(0,972 ± 0,019 s)
Maximum freshness	5 (seconds)
Virtual Node output data size	72 Bytes
Application request size	140 Bytes (average)
Sensing data size – (SDsize) – FiWARE	122 bytes
Time to feed the local database	30(seconds)
Number of Edge nodes (EN)	5
Number of End devices	4
Communication latency between ENs	20 ms
The average power consumption of devices	20w

For the purpose of assessing if the received requests are met, we used the metrics REQ_MET (M1) and REQ_NOT_MET (M2) that measure the effectiveness of the resource management (RM) algorithms to meet the application requests arriving in the system. REQ_MET computes the number of requests which were properly met, while REQ_NOT_MET records the number of requests which the RM algorithm was unable to meet during a monitoring period. Hereafter, we present the expression (3) and the expression (4), used to assist in measuring the metrics REQ_MET and REQ_NOT_MET.

$$\mathrm{TTS} = (\mathrm{TIME\_REQ} + \mathrm{TIME\_SPENT\_FW} + \mathrm{COMM\_TIME}) \tag{3}$$

$$\mathrm{f}(DType, RTTH, TTS) = \begin{cases} 1 \mid DType\ is\ provided \wedge TTS \leq RTTH \\ 0 \mid DType\ is\ not\ provided \vee TTS > RTTH \end{cases} \tag{4}$$

Per expression (3), the metric TTS represents the total time spent to meet a given request. It is the sum of the time periods collected through the parameters TIME_REQ, TIME_SPENT_FW, and COMM_TIME. TIME_REQ denotes the time required by our algorithm to handle a request at the same edge node where the application has issued the request. TIME_SPENT_FW measures the time spent since receiving the request, finding a neighbor node, and forwarding the request to the neighbor node. Lastly, COMM_TIME measures the communication time between the node that received the request and the neighboring edge node that will serve the request. It is important to mention that the parameters TIME_SPENT_FW and COMM_TIME are collected only when the collaboration process is executed.

Expression (4) presents a function used to determine whether the request has been met adequately or not (i.e. to calculate REQ_MET and REQ_NOT_MET). The function receives as input the datatype (DType) and the response time threshold (RTTH) from the request, along with the measured value of the TTS metric. The function returns the value 1 when a request has been met, i.e., if the DType required is *provided* in the system and the value of TTS is less than or equal to the RTTH desired by the application. On the other hand, the function returns the value 0 if the DType is not *provided* or TTS is greater than RTTH. Furthermore, our CoT system considers if the datatype is *provided* when the virtual node responsible for processing it has enough resources in term of memory, CPU and the datatype descriptor configured. Otherwise, the datatype is classified as *not provided* and the application request is forwarded to the collaboration mechanism that, if it is active, checks the availability of a neighboring virtual node to process it.

Therefore, the metrics aforementioned help us assessing the effectiveness of our CoT system to meet more requests using the proposed collaborative RM, in comparison to a system without collaboration among edge nodes.

To answer question Q2, it is necessary to assess if the collaboration process helps decreasing the data traffic between edge nodes and *end devices* (thus, contributing to save bandwidth consumption in the CoT system). To do so, we used a set of metrics to evaluate the data traffic in relation to the consumption and saving obtained during the execution of the collaboration mechanisms. The network usage is a concern in this work since its high consumption affects the overall system performance. Thus, we selected the metrics

AVG_DT_EN (M3) and AVG_DT_ENED (M4) to assess the data traffic. The metric AVG_DT_EN (M3), represented by expression (5), has the purpose of monitoring the entire data flow of the collaboration process, i.e., the message exchange (request/response operations) between edge nodes. For example, a message exchange may represent the operation of an edge node sending the application request to the neighboring edge node to process. Each message (including body and headers) has its size calculated and registered in the system.

$$\text{AVG\_DT\_EN} = \text{DT\_REQREC\_APP} + \text{DT\_REQSENT\_NB} + \text{DT\_DTSHARING} + \text{DT\_CALLBACK} \quad (5)$$

Per expression (5), the metric AVG_DT_EN is the sum of the data traffic generated to meet the application request and is collected through the parameters DT_REQREC_APP, DT_REQSENT_NB, DT_DTSHARING, and DT_CALLBACK. The parameter DT_REQREC_APP records the data traffic to receive an application request arriving in the system. The parameter DT_REQSENT_NB records the data traffic during the operation of an edge node to select the neighboring edge node and send to it the application request to process. The parameter DT_DTSHARING records the data traffic during the operation of data sharing among virtual nodes. Lastly, the parameter DT_CALLBACK records the data traffic to send the answer to the request issuer.

In addition to the consumption related to the communication between edge nodes, the data traffic regarding the interaction between edge nodes and *end devices* was also assessed, since it is another part of the CoT system that strongly influences in its overall performance in terms of bandwidth consumption. As aforementioned in Section 3.2, the virtual nodes (VN) can obtain fresh data either from historical databases maintained at the edge tier (local databases deployed into edge nodes), or from a direct connection with the physical nodes at the end devices tier, or from cache system in memory of the VN. Each source of data has a substantial impact on traffic in the network between the edge tier and end device tier. For instance, the less access to physical sensors, the less traffic is generated. Although these strategies are also used without running the collaboration, it is essential to evaluate whether our collaboration, along with the proposed mechanism for data sharing among VNs, helps to decrease the number of accesses to the end device tier, thus, reducing the data traffic. The metrics used for evaluation are described below.

The metric AVG_DT_ENED (M4), represented by expression (6), has the purpose of assessing the data traffic during the interaction between edge nodes and *end devices* to obtain sensing data or perform actuation tasks. In order to provide input to the expression (6), we used the parameter R_ENDEV responsible for recording the number of requests served using sensing data read directly from the *end device*. Moreover, the parameter *SDsize* represents the size of the sensing data in kilobytes (Kb) handled in our CoT system (Table 4). Per expression (6), we calculate the metric AVG_DT_ENED by multiplying value of the parameter R _ ENDEV with the size of the sensing data (*SDsize*).

$$\text{AVG\_DT\_ENED} = \textit{R\_ENDEV} * \textit{SDsize} \quad (6)$$

Therefore, when the collaboration process finalizes, the metrics AVG_DT_EN (M3) and AVG_DT_ENED (M4) help to identify the traffic generated during such a process. In this way, we are able to know the overhead added on the network and determine the effectiveness of our CoT system to reduce the data traffic between edge nodes and *end devices*.

Furthermore, we have selected the AVG_DT_TRAFFIC_KB (M7) metric to provide information to assess if our system is cost-effective regarding the traffic generated in the network to meet a request with the collaboration process enabled and disabled. The metric is calculated using the expression (7). To provide input for the expression, we use as parameters the previously mentioned AVG_DT_EN and REQ_MET metrics.

$$\text{AVG\_DT\_TRAFFIC\_KB} = \text{AVG\_DT\_EN}/\text{REQ\_MET} \tag{7}$$

Therefore, when the collaboration process finalizes, the metrics AVG_DT_EN (M3), and AVG_DT_ENED (M4) help identifying the traffic generated during such a process. Also, the metric AVG_DT_TRAFFIC_KB helps to identify the average in KB of the data traffic to meet a request. In this way, we are able to know the overhead added on the network and determine the effectiveness of our CoT system to reduce the data traffic between edge nodes and end devices.

Regarding the defined question Q3, in order to provide input to answer it, we need to assess if the collaboration along with the proposed mechanism for data sharing among virtual nodes effectively helps to save energy of the *end devices*. To do so, we used the metric ED_ENERGY (M5) that represents the energy consumption of the *end device* to meet the edge node request (e.g., delivering a requested sensing data). Since it is not the purpose of this work to define any novel energy model, we calculated the metric using the expression (8). The expression was defined based on the energy model available in Santos et al. [61].

$$\text{ED\_ENERGY} = (\text{PTM}*\text{NODE\_POMAX}*1.0)*\text{R\_ENDEV} \tag{8}$$

In order to provide input to the expression, we used the parameters R_ENDEV, PTM, and NODE_POMAX. The parameter R_ENDDEV is the same used in expression (6) whose purpose is to provide the number of requests met using sensing data read directly from end devices instead of using data available from local database and memory cache. The parameter PTM represents the processing time to meet the request. Lastly, the parameter NODE_POMAX represents the maximum node power. Therefore, at the end of the collaboration, the metric ED_ENERGY helps to show if the data sharing mechanism decreases the accesses to the end devices nodes to obtain fresh data, thereby saving energy.

Finally, to answer the question Q4, we need to monitor all requests and check if the response time threshold required by the application was met. For this purpose, we used the metric REQ_RTTH_INV (M6) represented by expression (9).

$$\text{REQ\_RTTH\_INV} = \text{f}(\text{RTTH}, \text{TTS}) = \begin{cases} 0 \mid TTS \leq RTTH \\ 1 \mid TTS > RTTH \end{cases} \tag{9}$$

The metric computes the number of requests that, although being met by the system, did not have the response time threshold desired by the application satisfied, i.e., if the value of metric TTS is greater than RTTH. Therefore, this metric helps to assess the effectiveness of the collaboration process to meet the requests within the response time threshold required by the application. For some types of applications (time critical), sending data after the defined threshold is useless. For REQ_RTTH_INV (M6), a lower value indicates that our collaboration is efficient.

In addition, concerning the metrics to evaluate the latency and bandwidth, we assume that each edge node already has the container images deployed to run the respective VN. Thus, we

avoid overheads related to the transfer of image backup of the VN container between edge nodes, since it is further cheaper to request hard disk space to store the images of the container.

5.2 Proof-of-concept implementation

To achieve the goals mentioned in Section 5.1, we developed a Proof-of-Concept (PoC) that concretizes the components of our three-tier architecture (Section 3.1) and implements our algorithms (Section 4). In this section, we describe the PoC and provide details about the software framework used to its development as well as the hardware configuration to execute it.

To build the prototype of our system, we used the Java programming language [37]. We also used the Spring boot Framework version 2.1.3 [67] to build the micro services that represent our VNs as well as all the other components of our infrastructure. To run these components and VNs, we used Docker containers [24]. In the Docker containers, we installed Java 64-bits virtual machine version "11.0.3 2019-04-16 LTS" [37], that is the java version specifically prepared to perform the memory management within Docker containers. All the components of the infrastructure (e.g. resource allocator, resource provisioner, edge node manager, entry point) and the VNs are allocated to run within a separate container. However, the VNs are instantiated on demand. Moreover, the protocol used to perform communication among micro services was REST/HTTP [64], which is the standard among micro services. The messages exchanged among micro services were implemented in JSON [39] format and compacted using the HTTP GZIP [34], in order to reduce the number of bytes transmitted in the network.

The *end device* tier is virtualized by using virtual *end devices* nodes configured into the FiWARE framework [27]. FiWARE is a generic platform providing several components (called Generic Enablers, GEs) to build IoT systems in several application domains. FiWARE reduces the effort necessary for the CoT components to interact with the physical environment. As FIWARE is a generic platform, we have selected only the elements necessary to use in our CoT system, thereby avoiding the overhead of using unnecessary components. Thus, we used the GE Orion Broker [29] and the GE IoT Agent [28] to perform tasks related to the access to the physical environment. The GE Orion Broker allows getting sensing data from the local database. The sensing data are inserted into the local database by the *end devices*. The GE IoT Agent allows managing and communicating with the physical *end devices*. Each virtual *end device* node represents a physical node equipped with a sensor to provide sensing data. Nodes for sensing the ambient light (luminosity) and the temperature are examples. The number of virtual *end devices* nodes configured in our PoC depends on the performed experiment, and they are described in the next sections. Using the FiWARE, the virtual nodes obtain sensing data by invoking the GE Orion Context Broker at the edge device tier. In addition, when the access is to obtain fresh data, the message used to obtain the data directly from the sensors has an average size of 122 bytes (SDsize = 0.1191 KB). Also, the message dispatched (callback message to the client application) as the output of the Virtual Node has an average of 72 bytes of size. In our environment, the average delay measured for querying a sensing data in the Context Broker is $0.972 \pm 0{,}019$ s. We also assumed that all the deployed *end devices* pertain to the same WSN and are homogeneous regarding their capabilities in terms of CPU, memory, and battery. By using FiWARE, our framework is able to support both homogeneous and heterogeneous sensor nodes that are accessed via a single interface. Moreover, in the

performed evaluation, we did not use any actuator devices since to validate the research questions the experiments only required sensor nodes.

Concerning the Edge tier, a total of five edge nodes were simulated by deploying the code in VMware [71] virtual machines using CentOS 7 64-bit operating system [18]. The configuration of these virtual machines comprises 2.3 GB RAM, 10 GB HD space (with 2.5 GB used) and 2 CPUs model i7-8550U of 1.80 Ghz. The Edge nodes are connected using a local network with a link capacity of 10/100Mbps and the latency among them is defined according to the design of each experiment. Moreover, the process of setting up each edge node and its neighborhood is the same process of collaboration represented by Figure 7. We consider that such a process had already been run, previously to the start of the experiments, and the Service Provider Manager had already performed the configuration of the neighborhood of each edge node. In addition, the datatype descriptions were created and deployed in the edge nodes. They represent the sensing data generated from the virtual devices and provided by the respective VN. The same model from Figure 4 was used in the definition of these datatypes. In our implementation, there can only be one VN container of a datatype per edge node due to its multi-tenancy feature [52], but multiple edge nodes can perform the same VN. This multi-tenancy feature allows that one VN instance running in a container can meet multiple tenants (users) to provide temperature given data. Moreover, every VN is enabled to share sensing data with their neighbors VNs in our scenarios. A reference implementation of the LW-CoEdge is available as open-source in https://github.com/mpitanga/lwcoedge.git.

5.2.1 Application description and parameters used in the experiments

The smart city is a typical IoT use case that encompasses several types of applications such as smart building, smart traffic, smoke pollution mapping, smart transportation, to name a few [52, 65]. These emerging IoT-applications demand requirements such as low latency, energy efficiency, and location-awareness. Thus, Edge computing is used to provide flexible, scalable and smooth services closer to data sources and/or users. Therefore, these types of applications are suitable to depict the features of our CoT system.

In our PoC, we simulate a smart building application [49] and execute scenarios to assess both the resource management and the collaboration process proposed in this paper. A scenario represents a possible way to use a system (the interactions between the actor and system) to accomplish a given functionality under a set of conditions [14, 73]. Every scenario in our PoC encompasses a group of experiments in which a client application is used to simulate the end-user operating a system (web or app) responsible for issuing several requests to the CoT system. End users can access the sensing data either via the code of this application deployed in their devices or by invoking the application through a web page. In the experiments, we simulate several users sending requests and assess the scalability of our solution to meet them. Each request is specified based on the model depicted in Figure 3, and it encompasses (i) a single datatype, (ii) the desired value of data freshness in milliseconds, (iii) response time threshold, and (iv) a callback URL. Moreover, every application request arrives via an entry-point located at the edge tier and has an average size of 140 bytes.

Table 4 summarizes the relevant parameters used to the execution of the experiments. A desktop computer equipped with an Intel Core I7-8550U 1.80 GHz - 1.99 GHz processor and 16 GB RAM was used to run the experiments in a controlled environment within the Ubicomp laboratory at UFRJ. In our PoC, the number of VNs and the datatype provided by each VN are

specified and varied for each experiment. In the next sections, we describe the scenarios and the adopted values for specific requirements.

5.3 Design and analysis of the first scenario

In this Section, we describe the first scenario and how we designed and performed the experiments, along with the analysis of the achieved results.

This hypothetical scenario represents the monitoring of a rectangular area of 200 × 200 meters representing an office floor of a smart building where the sensors of the *end device* tier and the edge devices are deployed. In this area, the network of edge nodes has a latency of about 20 ms. Moreover, software components (based on FiWARE figway[1]) were used to simulate physical sensors deployed for monitoring the area, for instance, smoke sensors deployed on the restroom and meeting room. Thus, monitoring applications submit their requests to (a) detect if employees are smoking in an unauthorized area, e.g., within the restroom and meeting rooms; (b) monitor the temperature and humidity of the offices; and (c) monitor the luminosity of the environments. It is worth noting that the sensing data obtained in items (b) and (c) can be used to perform actuation tasks in order to regulate the air-conditioning system automatically and adjust the ambient light. However, the operation of actuation is not the focus of this current PoC. Based on this scenario, we run three experiments (E1, E2 and E3) to answer the research questions mentioned in Section 5.1. In Table 5, we describe the datatypes configured in each edge node to represent the sensing data obtained from the sensors as well as the neighboring edge nodes in this scenario.

In this scenario, the experiment was executed 10 times and every execution was divided into five groups regarding the number of requests sent to the CoT (starting with 200 requests and varying from 200 to 200 up to 1000 requests). For instance, in the first group 200 requests were generated, in the second 400, the third 600, the fourth 800 and in the fifth group 1000 requests were sent to the system. Therefore, in each execution of the experiment, a total of 3000 application requests were produced. Such a method leads to a confidence interval of 95% for the obtained results and it was limited by the performance of the hardware available to perform the experiments. During the execution of the experiments, for each application request we have varied: (i) the edge node receiving the request (entry point), (ii) the required value of

Table 5 Summary of edge nodes and neighborhood configurations

Edge Node	Datatype		Neighbor EN
EN0	DT1	UFRJ.UbicompLab.temperature	EN1, EN2
EN1	DT1	UFRJ.UbicompLab.temperature	EN0, EN2
	DT2	UFRJ.UbicompLab.humidity	
EN2	DT1	UFRJ.UbicompLab.temperature	EN0, EN1,EN3, EN4
	DT2	UFRJ.UbicompLab.humidity	
	DT3	UFRJ.UbicompLab.luminosity	
	DT4	UFRJ.UbicompLab.smoke	
EN3	DT3	UFRJ.UbicompLab. luminosity	EN2, EN4
	DT4	UFRJ.UbicompLab.smoke	
EN4	DT4	UFRJ.UbicompLab.smoke	EN2, EN3

[1] https://github.com/telefonicaid/fiware-figway

Table 6 Generated requests

Datatype	Number of requests	Data freshness (ms)
DT1	747	1000–5000
DT2	744	1000–5000
DT3	764	5000
DT4	745	5000
Total	3000	

data freshness, and (iii) the requested datatype. Although our approach allows each request to have a different value of sensing data rate depending on the application requirements, the current implementation of our PoC does not provide support to this feature yet. Thus, as we are using FiWARE, the sensors were manually configured to automatically push the sensing data to the Orion Context Broker (temporary database) at every 30 s. The "push" is the primary method used in FiWARE for the sensors sending their sensing data. Therefore, we are currently assuming the data sensing rate of 30 s for all the application requests. In Table 6, we present the content of generated and submitted requests. We can see that, for DT1, a total of 747 requests were generated with the values of data freshness varying randomly between 1000 and 5000 milliseconds (ms). A total of 744 requests were generated for DT2 with the values of data freshness also varying randomly between 1000 and 5000 milliseconds (ms). The total of requests for DT3 and DT4 are 764 and 745 respectively and were generated with the fixed value of data freshness of 5000 ms. We used different values of freshness to stimulate the data update into the Virtual node cache from either the sensors or database. Thus, we can assess the impact on data traffic during the tasks of data sharing and obtaining fresh data from the sensors. In order to assess whether the response time threshold (RTTh) desired by the application is being properly met, we are considering a response time value less than a second 1(s). Thus, if a response time greater than 1(s) is achieved, our system accounts the request as not being met. Such a value was defined taking into account that our scenario is not a real-time application that requires low response time like interactive games, intelligent traffic, and healthcare IoT systems [51, 55, 58, 77]. Furthermore, the adopted strategy was to run the system with and without using the collaboration process and measuring its effectiveness to meet the application requests in each case.

5.3.1 Experiment E1

We designed the experiment E1 to answer the questions Q1 and Q4, which assess the capability of the CoT system to meet application requests. According to Figure 12a, for the scenario executed using the resource management algorithm with the proposed collaboration process **enabled**, the CoT system can serve all the application requests submitted. For instance, in the first point, for every 200 requests submitted to the system, all the 200 requests were met using the collaboration process, an efficiency of 100%. This behavior is due to the ability of our algorithms to engage a neighbor node to meet a request that would not be met by the node that received the original request. On the other hand, when the collaboration is disabled the ability of the system to meet the requests is reduced. In figure (a) and (b), we can see in the first point that for each 200 requests submitted to the system it was only able to meet 101 and, not met 99 respectively, i.e., a decrease of the efficiency of 49.5%. The reason for this behavior is that without the collaboration, the node that receives the request but unmet it (either because it

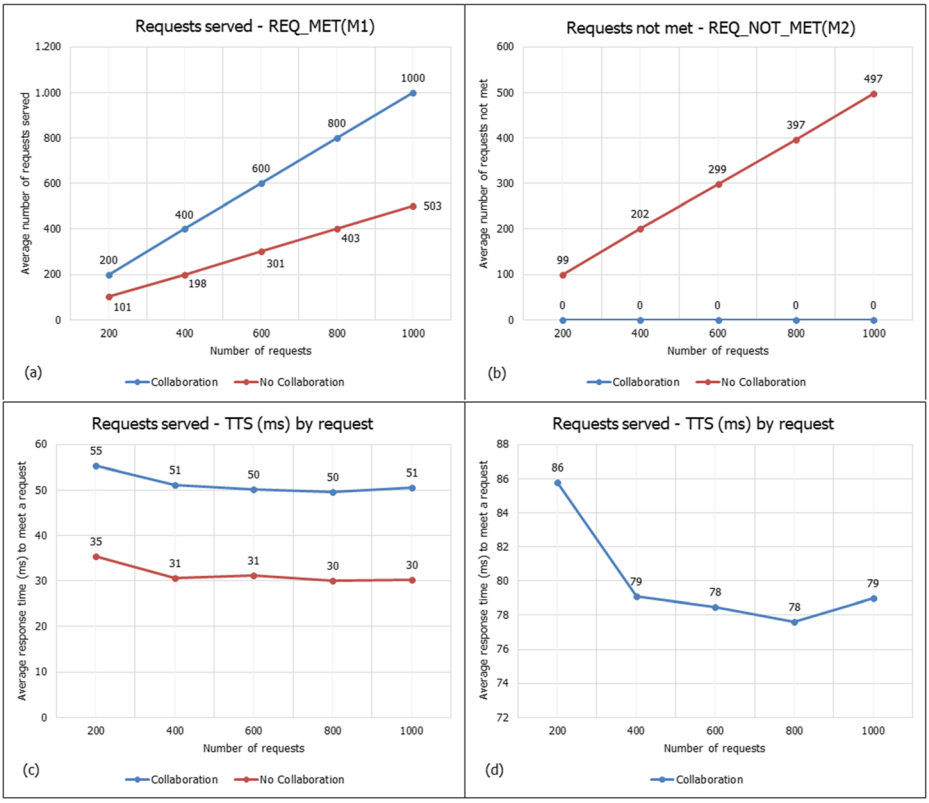


Figure 12 **a** Average number of requests served. **b** Average number of requests not met. **c** Average total time spent (TTS) to meet a request. **d** A view of the average total time spent (TTS) only of the requests met by the collaboration process

does not provide the data or because it is busy), cannot engage its neighboring nodes to forward the request to be met. This behavior can be observed up to the last point in the graph, for instance, the second point shows that for each 400 requests submitted to the system it was only able to meet 198 and not met 202, and so on.

Figure 12c shows the average total time spent (TTS) to meet a request with collaboration and without collaboration. For instance, we can see in the first point that for every 200 requests the TTS with the collaboration process enabled is 55 ms against a TTS of 35 ms without collaboration, an average overhead of 20 ms. In the second point, the TTS with collaboration is 51 ms against a TTS of 31 ms, and so on. In addition, we can observe in this experiment that this added overhead has a greater impact due to the latency values of the network since it is necessary to check the neighboring nodes to find the node with the best resources available to forward the request. This statement regarding the impact of network latency is observed in Figure 12d, in which we can see only the requests served using the collaboration process, i.e., those requests forwarded to be served by a neighboring node. According to the figure, the first point shows that 99 requests (Figure 12b) met using the collaboration process the average total time spent (TTS) to meet a single request was 86 ms. The second point shows that 202 requests met using the collaboration process the TTS to meet a single request was 79 ms, and so on. Moreover, we can observe that from 400 requests the total time spent to fulfill a request presents little variation stabilizing in the range between 78 and

79 ms. This behavior is due to some technical details implemented to increase the performance during the collaboration between edge nodes, for instance, a pool of HTTP connections[2] to re-use connections to avoid the overhead of creating new connections. Despite the latency impact in this experiment, the average response time achieved is less than the defined RTTh value.

From the results presented in this Section, we can conclude that the proposed collaboration process among virtual nodes effectively helped to meet a larger number of requests, with improved performance, i.e., satisfying the response time threshold desired by the application. Although our scenario does not represent a real-time application, the result obtained in these experiments, according to the values presented in PubNub Staff [58], are encouraging to our approach to also serve real-time applications with sensing data. Therefore, these results suitable provide answers to the posed questions Q1 and Q4.

5.3.2 Experiment E2

We designed experiment E2 to answer the question Q2, i.e., to show that our proposed collaboration among virtual nodes helps to reduce the data traffic at the end device tier while

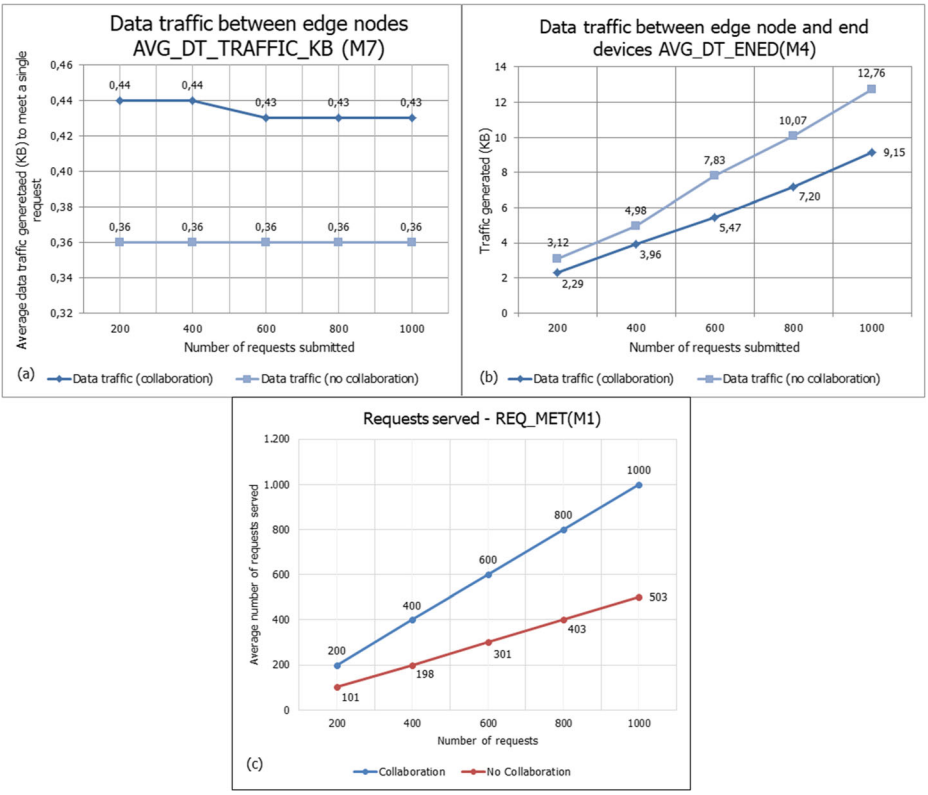


Figure 13 **a** Average data traffic generated during communication between edge nodes. **b** Average data traffic generated during communication between edge node and *end devices*. **c** Average number of requests served

[2] https://hc.apache.org/httpcomponents-client-ga/tutorial/html/connmgmt.html

at the same time meeting more requests. In our CoT system, for each incoming application request, when the collaboration process is enabled and the parameter DType (expression (4)) is classified as *not provided*, this means that the following tasks were executed by the components: (a) discovering of the best neighboring node in terms of the capability of available resources, (b) forwarding the requests to the discovered neighbor node. In addition to the tasks before mentioned, when the collaboration process is activated, the data sharing mechanism is also executed allowing a VN to share fresh data with its neighbor VNs. These tasks generate new messages (control messages), which increase the data traffic in the system. In addition, the amount of control messages executed depends on the number of neighbor nodes connected to the edge node that received the initial request. Also, an additional message is generated whenever it is required to instantiate a new container or to scale-up one.

In order to properly show both the data traffic during the collaboration between CoT components at the edge tier and the reduction of traffic between edge nodes and *end devices* through the proposed collaboration process, we divided the experiment into two parts. The first part, depicted in Figure 13a, calculates the data traffic generated at the edge tier to meet the requests with the collaboration process turned on and off. The data traffic was measured through the metric AVG_DT_EN (M3) that represents the traffic at the Edge tier from the reception of the request, during the communication between CoT components to discover the best neighboring edge node to meet the request, until the delivery of the data to the request issuer. The second part of the experiment, depicted in Figure 13b, has the purpose of showing how much data traffic the proposed collaboration process, along with the data sharing mechanism, was able to decrease during the communication between the edge nodes and *end devices.*

By observing Figure 13a, as expected, the scenario executed with the proposed collaboration process incurs in data traffic overhead due to the control messages required to perform the collaboration. We can also see in Figure 13a and c that there is a relation between the increase of the data traffic, the overhead due to the communication among the edge nodes, and the higher number of requests met. In contrast, when we have fewer requests met, the data traffic also decreases. By using the expression (7) to measure the amount of data traffic (with collaboration and without collaboration) to meet a single request, we can observe from the results obtained that our collaboration process is efficient even presenting an increase of data traffic at edge network. For example, in the first point of the figures (a) and (c), we can see that the average of data traffic (KB) to meet 200 of 200 requests submitted using the proposed collaboration process is 0.44 KB per request, whereas without collaboration the average of data traffic to serve 101 of 200 requests submitted is 0.36 KB per request, i.e., a difference of 18.2% of the data traffic. This behavior can be observed up to the last point in the graph, where we can see that the average of data traffic (KB) to meet 1000 of 1000 requests submitted using the proposed collaboration process is 0.43 KB per request, whereas without collaboration the average of data traffic to serve 503 of 1000 requests submitted is 0.36 KB per request, i.e., a difference of 16.3% of the data traffic. It is worth noting that from the third point up to the last point, the average of data traffic is constant. Moreover, for the CoT system to meet all the 1000 requests without collaboration, the user will need to submit at least 100% more requests, i.e., 2000 requests, which will consume according to the metric AVG_DT_TRAFFIC_KB (M7), on average, 0.53 KB in data traffic per request, i.e., an increase of 23.3% concerning the measured consume using the collaboration to meet the same 1000 requests. This increase is mainly due to the data traffic

generated from the requests received but not met. Therefore, we can conclude from the results before mentioned, that the difference presented of 16.3% of data traffic is low in relation to the user's perception of the system inefficiency to meet requests when the collaboration is disabled.

Furthermore, according to Figure 13b and c, we can observe that there is strong relation between requests met with collaboration **enabled** and the **reduction** of data traffic. For instance, we can see in the first point that for every 200 requests submitted to the system, the communication between edge nodes and *end devices* (to obtain data) produced, on average, 2.29 KB of data traffic using collaboration against 3.12 KB without collaboration, i.e., a decrease of 26.6%. This reduction in data traffic can be observed at all points in Figure 13b and intensifies as more requests are processed. For instance, we can see in the last point of the figure that for every 1000 requests submitted, the data traffic using collaboration was 9.15 KB against 12.76 KB without collaboration, i.e., a decrease of 28.3% in data traffic using the collaboration. This behavior is related to the strategy used by the virtual node (VN) to select the data source, either from the local database or from memory cache. Moreover, the proposed mechanism for data sharing was able to update the fresh data obtained from the sensors by a VN in the cache memory of its neighboring VNs, thereby avoiding the communication with the *end device* tier, and consequently reducing the network traffic.

Answering the question Q2, we conclude from the results presented in this Section that the proposed collaboration process along with data sharing among virtual nodes effectively reduces the data traffic between the edge tier and end device tier while properly meeting the received application requests.

5.3.3 Experiment E3

We designed experiment E3 to answer the question Q3, assessing if the collaboration among virtual nodes helps to save energy for the *end device* nodes. Figure 14a shows the energy consumption of the devices for different similarity degrees of application requests. In the figure, each point on the X-axis represents the percentage (%) of similar requests processed in the CoT system. A similarity between requests occurs when the system receives two or more requests with the same characteristics, i.e., the same datatype, with the same data freshness value, and so on. Therefore, the data sharing feature of our proposal can be used, and sensor data already

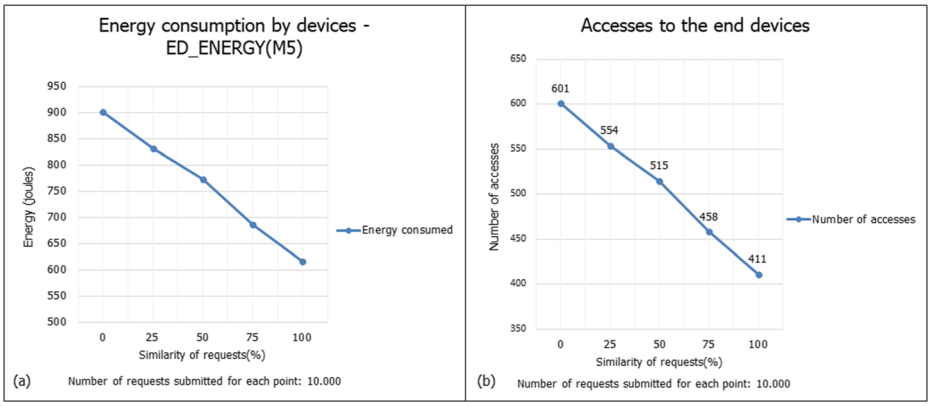


Figure 14 **a** Energy consumption (experiment E3). **b** Number of accesses to the end devices

acquired may be reused to meet new (similar) arriving requests. In this experiment, the requests were generated for the temperature datatype. In order to measure the energy consumption, ten thousand (10000) requests were generated for each point. A 0% of similarity between requests denotes that the ten thousand requests were processed without taking advantage of the data sharing, since there were no common datatypes among the requests. Meanwhile, a similarity of 25% indicates that, from the 10,000 requests submitted, only 2500 requests were processed with the data sharing mechanism being exploited, and so on up to 100%.

By analyzing the results, and responding to Q3, we can conclude that the proposed collaboration process along with data sharing among virtual nodes helps saving energy for the *end device* nodes. In Figure 14a, we can observe that the higher the similarity of the requests, the more energy of the *end devices* was saved. Such behavior happens since our CoT system avoids the continuous access to the *end device* to obtain the same datatype previously acquired and processed. For example, after executing ten thousand requests, we can observe in Figure 14b that when the similarity is equal to 0%, i.e., no similarity, our system accessed 601 times the *end devices* nodes to obtain sensing data. For a behavior where all requests are similar, i.e., equal to 100%, the system performed 411 accesses to the *end devices*, thereby showing a reduction of 31.6%. This performance is obtained by combining our data sharing process and the datatype cache system of the virtual nodes. It is important to notice that performance can vary for more or less, depending on the data freshness configured in the request. In this experiment, all the requisitions were generated with constant values of freshness (Table 4). Therefore, we can conclude that, for a higher freshness value, a lesser amount of accesses to the end devices to obtain sensing data will be performed.

Going back to the goals defined to assess our work, the results of the experiment E1 showed that the goal G1 was properly achieved, since an increase in efficiency of 100% to meet a larger number of requests and with improved performance was observed when the proposed collaboration process was enabled. The achievement of goal G2 can be observed from the results of experiments E2 and E3, since the system was able to effectively reduce both the energy consumption of the sensors as the data traffic during the processing of the application requests received. Therefore, from the result of the three experiments depicted above, we conclude that the goals presented in Section 5.1 were overall achieved.

5.3.4 Comparative analysis

In this Section, we present a comparative analysis of our results in comparison to the work described in [72], where the authors present the Edge Node Resource Management (ENORM), a framework for provisioning and auto-scaling edge node resources. As shown in Section 4, ENORM was used as inspiration to create our distributed resource management algorithm even though the final model and architecture of our proposal are totally distinct. The presented analysis is divided into two parts. The first analysis concerns the performance in terms of response time (Experiment E1) and the second refers to the data traffic generated in the network (Experiment E2).

A. Comparative Discussion on Response Time

Due to the characteristics of our scenarios, we assume that up to one second of response time is adequate to fit the classes of IoT applications considered in the work. We do not assume hard

real-time applications in the scenarios considered in the performed experiments. However, in PubNub Staff [58], the authors discuss the impact of the real-time applications and the response time desirable in online games. The authors argue that the ideal response time for such scenarios is 50 ms, a delay of even 100 ms reduces player performance and if a delay of 150 ms occurs, the gameplay is degraded.

In ENORM [72], the authors present an experiment to assess the performance of their resource allocation and resource provisioning approach using an open-source version of Pokemon go game (iPokeMon[3]). By observing their results using edge servers (Figure 9c in [72]), ENORM achieved an average latency of 100 ms using multiples Belfast edge servers (128), thereby being within the range of 50 and 100 ms. Thus, ENORM is an approach suitable to meet real-time applications. Regarding our proposal, we performed the experiment E1 to show the average response time achieved by using our proposed approach. The results of the experiments have shown encouraging results, even using a more resource-constrained environment (e.g., number of edge nodes) than the one used by ENORM. In our proposal, Figure 12 shows response times close to the ideal values (values between 50 ms and 100 ms) according to the discussion presented in PubNub Staff [58]. By observing the measured times from the requests met with or without the collaboration process, we clearly show that the same QoS requirement in terms of response time was met with fewer resources being consumed. Therefore, our proposal makes more efficient use of the available resources then ENORM.

B. Data traffic

The high data traffic generates bottlenecks in the network and increases the time of communication during the collaboration between the devices, thereby reducing the response time of the applications. Therefore, the reduction of data traffic is a feature necessary in Edge computing.

In ENORM, the authors present results comparing the data traffic between the Cloud tier and the Edge tier. They show a significant breakthrough in reducing data traffic to the Cloud, ranging from 88% to 95% in the performed evaluation using iPokeMon. Such a performance is achieved thanks to ENORM's ability to place the computing closer to the data source, thereby avoiding the sending of all data to the cloud. Moreover, in the experiments with ENORM [72], the data traffic between the servers (cloud and edge) was 12 KB per user, which includes the information necessary to configure the chosen edge node (control messages) and the user data. Regarding our proposal, we performed the experiment E2 to show the data traffic required to meet an application request. The average data traffic to meet the application requests using the proposed collaboration process ranges from 0.43 KB to 0.44 KB per request between the edge nodes. Besides the sensing data requested, such data traffic also includes the control messages for the discovery and selection of the neighbor node to forward the request. Therefore, by comparing with the hierarchical approach adopted in ENORM, our flat model presented a smaller volume in terms of data traffic generated in the network, thereby favoring those applications sensitive to latency.

5.4 Design and analysis of the second scenario

In this Section, we describe the second scenario, the performed experiment, along with the analysis of the achieved results.

[3] https://github.com/Kjuly/iPokeMon

For this hypothetical scenario, we have considered an area with approximately 800 × 400 meters representing a distribution center of a retail company at the Rio de Janeiro city where the sensors of the end device tier would be deployed. At this location, the company stores high valuable goods and cartons that are highly flammable. Due to the climate conditions of the city, the distribution center faces high temperatures, which affect both the equipment used for order processing and the well-being of employees. Thus, the company deployed an application to monitor the environment temperature to regulate the cooling system automatically. Moreover, the system also is used to detect the presence of smoke that may indicate a fire situation caused either by smoking in an unauthorized area or by overheating of an electronic device used in order processing. Data captured from the sensors are displayed on monitors scattered in the distribution center as well as being accessible from mobile devices. In this second scenario, experiment E4 was performed to assess the capability of our CoT system to meet requests in an environment with edge nodes **overloaded** to serve requests. Moreover, we also show how the algorithm performs balancing among the neighbor edge nodes. Thus, this experiment also helps in answering the research questions Q1 and Q4 mentioned in Section 5.1.

5.4.1 Experiment E4

To execute this experiment, we modified relevant system settings in the edge nodes concerning the datatypes, and neighborhood. Unlike the first scenario, the edge nodes are homogeneous regarding the provided data types allowing more effective demonstration of the collaboration behavior (e.g., load balancing) when one or more edge nodes are **overloaded**. In Table 7 we describe the configuration concerning the datatypes and the neighborhood. For instance, the edge node EN0 was configured to provide the datatypes DT1 and DT4 and its neighbors are the edge nodes EN1 and EN4, and so on.

To simulate the behavior of the overloaded edge node, i.e., the resource exhaustion of the VN to meet a request, for this experiment we use the parameter of memory available for processing, since the number of available CPUs is always the same. As our PoC was developed using Java programming language, the low volume of heap memory causes the JVM to run more intensively the garbage collector (GC)[4] operation. The GC operation suspends the execution of the components until the allocated, but unused memory is released, thereby generating a slowness to meet the requests. The minimum value of 14% of available memory (JVM heap) was set for all the edge nodes. Thus, whenever the available memory is less than the minimum configured, our resource management algorithm tries to scale up the

Table 7 Summary of Edge Nodes, datatypes and neighborhood configuration (second scenario)

Edge Node	Datatype		Neighbor EN
EN0	DT1	UFRJ.UbicompLab.temperature	EN1, EN4
	DT4	UFRJ.UbicompLab.smoke	
EN1	DT1	UFRJ.UbicompLab.temperature	EN0, EN2, EN3
	DT4	UFRJ.UbicompLab.smoke	
EN2	DT1	UFRJ.UbicompLab.temperature	EN1,EN3
	DT4	UFRJ.UbicompLab.smoke	
EN3	DT1	UFRJ.UbicompLab.temperature	EN1, EN2, EN4
	DT4	UFRJ.UbicompLab.smoke	
EN4	DT1	UFRJ.UbicompLab.temperature	EN0, EN3
	DT4	UFRJ.UbicompLab.smoke	

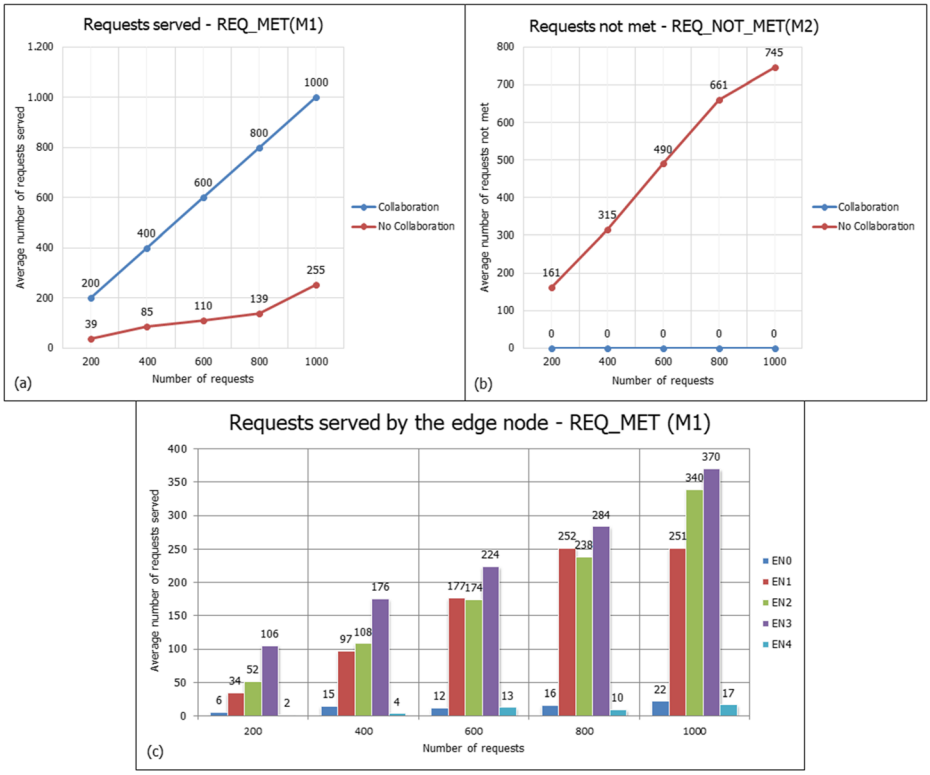


Figure 15 **a** Average number of requests met. **b** Average number of requests not met. **c** Average number of requests met and distributed through Edge Nodes

container by adding more memory. Moreover, we are considering that the physical memory available of the edge node is not enough to scale up the container, then this task will fail, and the request will be forwarded to the collaboration process. It is worth to inform that the default setting for the minimum memory available in our system is 5%.

The operation of the experiment E4 is similar to the E1 with some modifications regarding parameters used to its execution. First, the freshness was set to 5(s). Second, the requests were submitted to the same edge node EN2, thereby generating an **overloaded** of its resources. We choose the EN2 since it is the neighbor of the EN1 and EN3 that are neighbors with each other. Moreover, EN1 and EN3 are neighbors of the other ENs. Therefore, a request not met by EN2 can be met by any EN.

Figure 15a shows the requests met with the proposed resource management (RM) algorithm and collaboration process **enabled** and **disabled**. We can see in the figure that our system can serve 200 out of 200 submitted requests using the collaboration process against only 39 requests met without collaboration. Similarly, the system served 400 out of 400 requests submitted whereas only 85 were met without collaboration. This behavior is observed up to the last point in the graph, where 1000 out of 1000 requests submitted were successfully met with collaboration against only 255 requests met without collaboration, thereby demonstrating a decrease in efficiency of 74.5%. Therefore, by using the proposed collaboration process, we obtained an

[4] https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

efficiency of 100% in meeting the application requests. This performance was achieved by using the neighboring nodes to meet the requests that the source node (EN2) could not serve. Figure 15b depicts the requests not met. We can see in the first point of the graph that 161 out of 200 requests submitted were not met by the CoT system because the collaboration process has been **disabled**. In the second point, 315 out of 400 requests submitted were not met. We can see this behavior up to the last point in the graph. These values represent the number of requests not forwarded for processing on the neighboring edge nodes.

Figure 15c shows how our system distributed the requests between the other ENs when the collaboration process is enabled and the EN that received the request is unable to meet. According to the figure, we can see that for every 1000 requests submitted to the EN2, it was able to serve only 340 requests due to the exhaustion of its resources. For the 660 requests not met by the EN2, our distributed resource management with collaboration selected the neighboring edge node EN1 and EN3 to meet such requests since these ENs had the best resources at that moment. Moreover, when EN1 and/or EN3 did not have enough resources to meet a request, our algorithm selected one of its neighboring nodes to forward the request, and so on. The remaining of the distribution occurred as follows. The edge node EN1 served a total of 251 requests. The edge node EN3 met 370 requests. The edge node EN0 met 22 requests. Lastly, the edge node EN4 met 17 requests.

From the results presented in this Section, we can conclude that even though the CoT system has presented edge nodes with the exhaustion of resources, the proposed collaboration process effectively helped to engage the neighbor edge nodes to meet such requests. Despite this, the requests were served by satisfying the response time threshold desired by the application. Therefore, these results suitable provide answers to the posed questions Q1 and Q4.

6 Final remarks

In this work, we presented LW-CoEdge, a novel lightweight virtualization model and P2P collaboration process for Edge computing. Our proposal relies on a three-tier architecture using Edge computing, and it is supported by two main technologies/approaches: containerization and microservice. The containerization provides a simple mechanism for configuration, packaging, and instantiation of our components and tackling the heterogeneity of edge nodes to build the virtual nodes (VN). Moreover, it allows hiding the physical details of infrastructure thus avoiding dependence on any technology, consuming less resource, and processing time. The microservices are exploited to develop our VNs and operational support components. They are designed as lightweight components, i.e., small, highly decoupled and performing a single responsibility. Therefore, containerization and microservice fulfill the goal of enabling our lightweight virtualization model besides facilitating the distribution and managing our components on the edge nodes. Concerning the collaboration, we designed a flat P2P process to create and manage the neighborhood of edge nodes besides allowing the data sharing and the distributed resource management. The data sharing allows a VN to share its fresh data with neighboring VNs actively. Also, the data sharing provides the capability for VNs to identify the data demands of other VNs during the collaboration process, thereby reducing the data transmissions between VNs. Furthermore, our resource management distributes the decision-making at the edge of the network, i.e., each edge node is provided with the ability to engage neighboring edge nodes to allocate or provision VNs when needed, thereby allowing meeting more requests.

A prototype with all the proposed features was implemented in a real environment. This prototype was used as a Proof-of-Concept, generating the data that served as input for the experiments performed in order to assess the proposed process and algorithms. The results of the performed experiments showed that our algorithm for distributed allocation of resources and resource provisioning, improved with flat P2P collaboration and data sharing, enabled the CoT system to meet more application requests, with lower latency, while saving energy of the end devices and decreasing the overall data traffic.

Despite the advances promoted by our proposal to advance the state-of-the-art of the Edge computing paradigm, we have selected a set of open issues that need to be addressed as future work to improve our CoT system. First, we are planning new experiments to be applied in a real environment encompassing a larger number of heterogeneous edge nodes deployed in a geographically larger area. The current number of nodes used in our experiments was limited by the computational capacity available at the laboratory where the experiments were executed. Second, our solution currently allows the applications to send only a single datatype per request. Therefore, we intend to support applications requesting different types of data in a single request in the future. Third, we plan to design a new monitor system to identify the most commonly used VN containers to start them during the system boot automatically, thereby optimizing the initial response time to meet a request. The task of starting the new container requires an amount of time. During this operation, the requests are queued to be met later and are served with high response time. Therefore, with a proactive approach to start containers, we aim at further improving the response time of the system. Lastly, we intend to integrate our virtualization model with other frameworks in charge of interacting with the Things tier, for instance, the Edgex Foundry [25].

Acknowledgements This work is partially funded by FAPESP (grant 2015/24144-7). Professors Flavia C. Delicato and Paulo F. Pires are CNPq Fellows.

References

1. Aazam, M., Huh, E.N.: Fog computing: the cloud-iot/ioe middleware paradigm. IEEE Potentials. **35**(3), 40–44 (2016)
2. Aazam, M., Khan, I., Alsaffar, A.A., Huh, E.N.: Cloud of things: integrating internet of things and cloud computing and the issues involved. In: Applied Sciences and Technology (IBCAST), 2014 11th International Bhurban Conference on, pp. 414–419. IEEE (2014)
3. Alam, M.G.R., Hassan, M.M., Uddin, M.Z., Almogren, A., Fortino, G.: Autonomic computation offloading in mobile edge for IoT applications. Futur. Gener. Comput. Syst. **90**, 149–157 (2019)
4. Aloi, G., Caliciuri, G., Fortino, G., Gravina, R., Pace, P., Russo, W., Savaglio, C.: Enabling IoT interoperability through opportunistic smartphone-based mobile gateways. J. Netw. Comput. Appl. **81**, 74–84 (2017)
5. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., et al.: A view of cloud computing. Commun. ACM. **53**(4), 50–58 (2010)
6. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
7. Basili, R.B., (1992). Software Modeling and Measurement: The Goal/Question/Metric Paradigm. Technical Report. University of Maryland at College Park, College Park, MD, USA.
8. Bonomi, F.: Connected vehicles, the internet of things, and fog computing. In: The Eighth ACM International Workshop on Vehicular Inter-Networking (VANET), Las Vegas, USA, pp. 13–15 (2011)
9. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, pp. 13–16. ACM (2012)

10. Bonomi, F., Milito, R., Natarajan, P., Zhu, J.: Fog computing: a platform for internet of things and analytics. In: Big Data and Internet of Things: a Roadmap for Smart Environments, pp. 169–186. Springer International Publishing (2014)
11. Botta, A., De Donato, W., Persico, V., Pescapé, A.: On the integration of cloud computing and internet of things. In: Future Internet of Things and Cloud (FiCloud), 2014 International Conference on, pp. 23–30. IEEE (2014)
12. Bouzeghoub, M.: A framework for analysis of data freshness. In: Proceedings of the 2004 International Workshop on Information Quality in Information Systems, pp. 59–67. ACM (2004)
13. Byers, C. C., & Wetterwald, P. (2015). Fog computing distributing data and intelligence for resiliency and scale necessary for IoT: the Internet Of Things (ubiquity symposium). Ubiquity, 2015 (November), 4
14. Carrol, J.M.: Five reasons for scenario-based design. In: Proceedings of the 32nd Annual Hawaii International Conference on Systems Sciences. 1999. HICSS-32. Abstracts and CD-ROM of Full Papers. 11-pp, IEEE (1999)
15. Casadei, R., Fortino, G., Pianini, D., Russo, W., Savaglio, C., Viroli, M.: Modelling and simulation of opportunistic IoT services with aggregate computing. Futur. Gener. Comput. Syst. **91**, 252–262 (2019)
16. Catarinucci, L., De Donno, D., Mainetti, L., Palano, L., Patrono, L., Stefanizzi, M.L., Tarricone, L.: An IoT-aware architecture for smart healthcare systems. IEEE Internet Things J. **2**(6), 515–526 (2015)
17. Cavalcante, E., Pereira, J., Alves, M.P., Maia, P., Moura, R., Batista, T., et al.: On the interplay of internet of things and cloud computing: a systematic mapping study. Comput. Commun. **89**, 17–33 (2016)
18. CentOS linux: https://www.centos.org/
19. CEP: Complex Event Processing. Available in: https://en.wikipedia.org/wiki/Complex_event_processing (2017). Last accessed: 11/07/2017
20. Cisco IOx: Available in: https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html. Last accessed: 11/07/2017
21. Delicato, F.C., Pires, P.F., Pirmez, L., Batista, T.: Wireless sensor networks as a service. In: Engineering of Computer Based Systems (ECBS), 2010 17th IEEE International Conference and Workshops on, pp. 410–417. IEEE (2010)
22. Delicato, F.C., Pires, P.F., Batista, T.: The resource management challenge in IoT. In: Resource Management for Internet of Things, pp. 7–18. Springer International Publishing (2017)
23. Distefano, S., Merlino, G., Puliafito, A.: Sensing and actuation as a service: a new development for clouds. In: Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on, pp. 272–275. IEEE (2012)
24. Docker: https://www.docker.com
25. EdgeX Foundry: The Open Platform for the IoT Edge. Available in: https://www.edgexfoundry.org
26. Endo, P.T., de Almeida Palhares, A.V., Pereira, N.N., Goncalves, G.E., Sadok, D., Kelner, J., et al.: Resource allocation for distributed cloud: concepts and research challenges. IEEE Netw. **25**(4), (2011)
27. FIWARE: https://www.fiware.org
28. FIWARE IoT Agent for Ultralight 2.0 protocol: https://github.com/telefonicaid/iotagent-ul
29. FIWARE Orion Context Broker, Release 4: http://fiware-orion.readthedocs.io/en/master/index.html
30. Fortino, G., Russo, W., Savaglio, C., Shen, W., Zhou, M.: Agent-oriented cooperative smart objects: from IoT system design to implementation. IEEE Transactions on Systems, Man, and Cybernetics: Systems. **99**, 1–18 (2017)
31. Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., et al.: Edge-centric computing: vision and challenges. ACM SIGCOMM Computer Communication Review. **45**(5), 37–42 (2015)
32. Giang, N.K., Blackstock, M., Lea, R., Leung, V.C.: Developing iot applications in the fog: a distributed dataflow approach. In: Internet of Things (IOT), 2015 5th International Conference on the, pp. 155–162. IEEE (2015)
33. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (IoT): a vision, architectural elements, and future directions. Futur. Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
34. GZIP: https://www.gzip.org/
35. Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B., Koldehofe, B.: Mobile fog: a programming model for large-scale applications on the internet of things. In: Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing, pp. 15–20. ACM (2013)
36. Inaba, M., Katoh, N., Imai, H.: Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering. In: Proceedings of the Tenth Annual Symposium on Computational Geometry, pp. 332–339. ACM (1994)
37. Java: https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html
38. Johnston, W.M., Hanna, J.R., Millar, R.J.: Advances in dataflow programming languages. ACM Computing Surveys (CSUR). **36**(1), 1–34 (2004)

39. JSON: https://www.json.org/
40. Khan, I., Belqasmi, F., Glitho, R., Crespi, N., Morrow, M., Polakos, P.: Wireless sensor network virtualization: a survey. IEEE Communications Surveys & Tutorials. **18**(1), 553–576 (2016)
41. Kokash, N.: An introduction to heuristic algorithms, pp. 1–8. Department of Informatics and Telecommunications (2005)
42. Lewis, J., & Fowler, M. (2014). Microservices. Available in: http://martinfowler.com/articles/microservices.html. Last accessed: 27/09/2017
43. Luan, T.H., Gao, L., Li, Z., Xiang, Y., Wei, G., Sun, L.: Fog computing: Focusing on mobile users at the edge. arXiv preprint. arXiv, 1502.01815 (2015)
44. Maaroju, N., & Garg, D. G. (2009). Choosing the best heuristic for a NP-Problem. Masters of Engineering, Patiala, Thapar University, Faculty of Computer Science and Engineering. June 2009
45. Madria, S., Kumar, V., Dalvi, R.: Sensor cloud: a cloud of virtual sensors. IEEE Softw. **31**(2), 70–77 (2014)
46. Mahmud, R., Kotagiri, R., Buyya, R.: Fog computing: a taxonomy, survey and future directions. In: Internet of Everything, pp. 103–130. Springer, Singapore (2018)
47. Morabito, R.: Virtualization on internet of things edge devices with container technologies: a performance evaluation. IEEE Access. **5**, 8835–8850 (2017)
48. Morabito, R., Cozzolino, V., Ding, A.Y., Beijar, N., Ott, J.: Consolidate IoT edge computing with lightweight virtualization. IEEE Netw. **32**(1), 102–111 (2018)
49. Morvaj, B., Lugaric, L., Krajcar, S.: Demonstrating smart buildings and smart grid features in a smart energy city. In: Proceedings of the 2011 3rd International Youth Conference on Energetics (IYCE), pp. 1–8. IEEE (2011)
50. Munir, A., Kansakar, P., Khan, S.U.: IFCIoT: integrated fog cloud IoT architectural paradigm for future internet of things. arXiv preprint. arXiv, 1701.08474 (2017)
51. Mutlag, A.A., Ghani, M.K.A., Arunkumar, N., Mohamed, M.A., Mohd, O.: Enabling technologies for fog computing in healthcare IoT systems. Futur. Gener. Comput. Syst. **90**, 62–78 (2019)
52. Naha, R.K., Garg, S., Georgakopoulos, D., Jayaraman, P.P., Gao, L., Xiang, Y., Ranjan, R.: Fog computing: survey of trends, architectures, requirements, and research directions. IEEE access. **6**, 47980–48009 (2018)
53. Nan, Y., Li, W., Bao, W., Delicato, F.C., Pires, P.F., Zomaya, A.Y.: Cost-effective processing for delay-sensitive applications in cloud of things systems. In: Network Computing and Applications (NCA), 2016 IEEE 15th International Symposium on, pp. 162–169. IEEE (2016)
54. Nishant, K., Sharma, P., Krishna, V., Gupta, C., Singh, K.P., Rastogi, R.: Load balancing of nodes in cloud using ant colony optimization. In: Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on, pp. 3–8. IEEE (2012)
55. Pace, P., Aloi, G., Gravina, R., Caliciuri, G., Fortino, G., Liotta, A.: An edge-based architecture to support efficient applications for healthcare industry 4.0. IEEE Transactions on Industrial Informatics. **15**(1), 481–489 (2018)
56. Pahl, C., Lee, B.: Containers and clusters for edge cloud architectures–a technology review. In: Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on, pp. 379, 2015–386. IEEE (2015)
57. Peralta, G., Iglesias-Urkia, M., Barcelo, M., Gomez, R., Moran, A., Bilbao, J.: Fog computing based efficient IoT scheme for the industry 4.0. In: 2017 IEEE International Workshop of Electronics, Control, Measurement, Signals and their Application to Mechatronics (ECMSM), pp. 1–6. IEEE (2017)
58. PubNub Staff. How Fast is Realtime? Human Perception and Technology. On-line publish. February 9, 2015. Available in: https://www.pubnub.com/blog/how-fast-is-realtime-human-perception-and-technology/. Last accessed: 05/09/2019
59. Sahni, Y., Cao, J., Zhang, S., Yang, L.: Edge mesh: a new paradigm to enable distributed intelligence in internet of things. IEEE Access. **5**, 16441–16458 (2017)
60. Santos, I.L., Pirmez, L., Delicato, F.C., Khan, S.U., Zomaya, A.Y.: Olympus: the cloud of sensors. IEEE Cloud Computing. **2**(2), 48–56 (2015)
61. Santos, I.L., Pirmez, L., Delicato, F.C., Oliveira, G.M., Farias, C.M., Khan, S.U., Zomaya, A.Y.: Zeus: a resource allocation algorithm for the cloud of sensors. Futur. Gener. Comput. Syst. **92**, 564–581 (2019)
62. Satyanarayanan, M., Bahl, P., Caceres, R., Davies, N.: The case for vm-based cloudlets in mobile computing. IEEE pervasive Computing. **8**(4), (2009)
63. Sheng, X., Tang, J., Xiao, X., Xue, G.: Sensing as a service: challenges, solutions and future directions. IEEE Sensors J. **13**(10), 3733–3741 (2013)
64. Shi, H., Chen, N., Deters, R.: Combining mobile and fog computing: using coap to link mobile device clouds with fog computing. In: Data Science and Data Intensive Systems (DSDIS), 2015 IEEE International Conference on, pp. 564–571. IEEE (2015)
65. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)

66. Skarlat, O., Schulte, S., Borkowski, M., Leitner, P.: Resource provisioning for iot services in the fog. In: Service-Oriented Computing and Applications (SOCA), 2016 IEEE 9th International Conference on, pp. 32–39. IEEE (2016)
67. Spring boot: https://spring.io/projects/spring-boot
68. Taleb, T., Dutta, S., Ksentini, A., Iqbal, M., Flinck, H.: Mobile edge computing potential in making cities smarter. IEEE Commun. Mag. **55**(3), 38–43 (2017)
69. Tan, L., Wang, N.: Future internet: the internet of things. In: Advanced Computer Theory and Engineering (ICACTE), 2010 3rd International Conference on(Vol. 5, Pp. V5–376). IEEE (2010)
70. Thönes, J.: Microservices. IEEE Softw. **32**(1), 116–116 (2015)
71. VMware: https://www.vmware.com/
72. Wang, N., Varghese, B., Matthaiou, M., Nikolopoulos, D.S.: ENORM: a framework for edge node resource management. IEEE Trans. Serv. Comput. (2017)
73. Weidenhaupt, K., Pohl, K., Jarke, M., Haumer, P.: Scenarios in system development: current practice. IEEE Softw. **15**(2), 34–45 (1998)
74. Xia, C., Li, W., Chang, X., Delicato, F., Yang, T., Zomaya, A.: Edge-based energy Management for Smart Homes. In: 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), pp. 849–856. IEEE (2018)
75. Xu, J., Palanisamy, B., Ludwig, H., Wang, Q.: Zenith: Utility-aware resource allocation for edge computing. In: Edge Computing (EDGE), 2017 IEEE International Conference on, pp. 47–54. IEEE (2017)
76. Yang, L., Li, W., Ghandehari, M., Fortino, G.: People-centric cognitive internet of things for the quantitative analysis of environmental exposure. IEEE Internet Things J. **5**(4), 2353–2366 (2017)
77. Yi, S., Li, C., Li, Q.: A survey of fog computing: concepts, applications and issues. In: Proceedings of the 2015 Workshop on Mobile Big Data, vol. 2015. ACM (2015a)
78. Yi, S., Hao, Z., Qin, Z., Li, Q.: Fog computing: Platform and applications. In: Hot Topics in Web Systems and Technologies (HotWeb). 2015 Third IEEE Workshop on, pp. 73–78. IEEE (2015b)
79. Zhang, B., Mor, N., Kolb, J., Chan, D. S., Lutz, K., Allman, E., ... & Kubiatowicz, J. (2015). The Cloud Is Not Enough: Saving IoT from the Cloud. In HotCloud

Affiliations

Marcelo Pitanga Alves[1] · **Flavia C. Delicato**[1] · **Igor L. Santos**[2] · **Paulo F. Pires**[1]

✉ Marcelo Pitanga Alves
mpitanga@gmail.com

Flavia C. Delicato
fdelicato@gmail.com

Igor L. Santos
igor.santos@cefet-rj.br

Paulo F. Pires
paulo.f.pires@gmail.com

[1] Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil

[2] Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET-RJ), Rio de Janeiro, Brazil