Architecture for Virtualization and Collaboration in Edge Computing: an implementation based on FIWARE building blocks

Marcelo P. Alves[1], Flavia C. Delicato[2], Igor L. Santos[3], Paulo F. Pires[2]

[1]Programa de Pós-graduação em Informática (PPGI) – Universidade Federal do Rio de Janeiro (UFRJ) – Rio de Janeiro – Brazil

[2]Instituto de Computação – Universidade Federal Fluminense (UFF) – Niterói – RJ – Brazil

[3]Programa de Pós-graduação em Engenharia de Produção e Sistemas (PPPRO) – CEFET-RJ – Rio de Janeiro – RJ – Brazil.

{mpitanga,fdelicato,paulo.f.pires}@gmail.com, igor.santos@cefet-rj.br

Abstract. *Edge Computing is a novel paradigm that allows moving the computation closer to the end-users and/or data sources. In this paper, we present a three-tier architecture for virtualization and collaboration of Virtual Nodes that leverages the Edge tier to meet those emerging IoT applications that demand requirements such as low latency, geo-localization, and energy efficiency. Besides the Edge tier, our implementation is based on the mix of lightweight virtualization and microservices using the building blocks from the FIWARE platform to interact with the physical environment. Furthermore, we presented two experiments to assess our architecture under severe and realistic conditions, regarding the network latency and fault-tolerance.*

1. Introduction

Despite the recent advances that have made IoT a reality, IoT devices typically have limited capability of resources and present some drawbacks regarding reliability, performance, security, and privacy. Thus, as the number of interconnected devices grows in a huge pace, several challenges arise regarding the infrastructure for supporting IoT applications. In this context, virtualized infrastructures such as Cloud Computing (Armbrust et al., 2010), Cloud of Things (Cavalcante et al., 2016), Cloud of Sensors (Santos et al., 2015), Fog/Edge Computing (Bonomi et al., 2014), and most recently Edge Mesh (Sahni et al., 2017) have emerged as solutions for supporting IoT applications regarding the storing, processing, and distribution of sensing data.

In our previous work (Alves et al., 2019), we advanced the state-of-the-art in Edge computing proposing LW-CoEdge, a lightweight virtualization model and collaboration process for edge computing. LW-CoEdge encompasses a virtualization model for Edge computing based on a novel concept of Virtual Node (VN). LW-CoEdge is based on the mix of two technology assets: (i) lightweight virtualization approach (Morabito et al., 2018) and (ii) microservices (Thönes, 2015). These two technologies are adopted to design the software that concretizes our virtualization model as lightweight components thereby facilitating their packaging, distribution and managing on the edge nodes. Moreover, we leveraged our lightweight virtualization model with a flat P2P

collaboration process to allow both the data sharing and the distributed resource management at the edge of the network. The data sharing allows the VN to share fresh data with its neighboring VN. The VN can be located at the same edge node or at the neighbor edge node. This approach improves energy and bandwidth consumption by avoiding neighboring virtual nodes performing redundant access to the sensor device to obtain the same data with a valid freshness (Santos et al., 2019). We also proposed heuristic-based algorithms enhanced with the flat P2P collaboration to perform the distributed resource management (resource allocation and resource provisioning) of virtual nodes (VN) and the data sharing between VNs. LW-CoEdge principles and design were validated and an initial evaluation showed that it succeeded in meeting its goals.

Here in this paper, the goal is to present the implementation of LW-CoEdge based on building blocks from the FIWARE platform. FIWARE is widely used by the academy and industry to build IoT applications. It provides building blocks, called Generic Enablers (GEs) (FIWARE GE, 2019), which are components of general purpose and available to be used at the Cloud and at the Edge tiers. In addition, GEs use the FIWARE NGSI protocol, a simple yet powerful open and RESTful API that facilitates dealing with heterogeneity and interoperability issues. Therefore, FIWARE GEs reduce the effort to design and integrate our lightweight components to interact with the physical environment (IoT devices). Besides describing the implemented architecture in detail, in this paper we present new experiments to assess LW-CoEdge under more severe and realistic conditions than it was performed in our previous work (Alves et al., 2019), regarding the network and execution context. **Network latency** is a critical factor in defining the success of a collaborative approach for CoT systems in order to meet application requests, particularly those related to time critical applications. Thus, we assessed our proposal in terms of the impact that a network with high communication latency values causes in our distributed resource management with the collaboration process enabled. In addition, we assessed LW-CoEdge regarding **fault tolerance**. Fault tolerance was achieved through the capability of collaboration/data sharing among the edge nodes. Thus, our CoT system can deliver a continuous service to applications despite fails in some edge nodes and/or end devices.

The remainder of this paper is organized as follows. Section 2 presents related work. Section 3 presents our implementation architecture for the virtualization model and collaboration process. Sections 4 and 5 describe the evaluation methodology and results. Section 6 presents final remarks and future work.

2. Related work

Madria et al. (2014) proposed a centralized virtualization model, which encompasses Virtual Sensors and provides sensing as a service for the users (SaaS). Unlike Madria et al. (2014), we implement a decentralized virtualization model tailored to meet requirements of emergent IoT applications such as low latency and location-awareness.

Santos et al. (2019) extended their original design of Olympus (Santos et al., 2015) 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. Our model provides predefined types of VNs representing each datatype provided to serve the application requests. This is important to favor the collaboration process among virtual nodes.

Shen et al. (2019) propose an information-centric collaborative Fog (ICCF) platform. In their work, a new in-network self-learning algorithm to run on Fog nodes is presented. By using a Fog-to-Fog connection, the Fog nodes communicate with each other to collect only the data required from their neighbor nodes. Moreover, the authors present a novel naming schema for sensor data content by utilizing ICCF characteristics and aiming to reduce the data communication load and delay that occur in IoT networks. Our proposal has a point of convergence with this work concerning the collaboration between fog/edge nodes for data sharing. However, it differs in one aspect. We provide a virtualization model with fully distributed resource management (allocation and provisioning) of virtual nodes (VN) and data sharing at the edge of the network. In our data sharing mechanism, the VN only connects with its neighbor VNs that provide the same data semantics. Unlike ICCF where a node requests data to its neighbors, our data sharing approach is active, i.e., the VNs are actively sending fresh data to their neighbors.

Wang et al. (2017) 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.

3. LW-CoEdge architecture

In this section, we describe the software components that encompass the proposed architecture to concretize the concepts behind the LW-CoEdge model (Alves et al., 2019).

In LW-CoEdge, the Virtual Node (VN) is the core computational unit of our virtualization model to provide sensing data or perform actuation in response to the application requests. We leverage the Edge tier for the provision of VNs closer to the data source (*end devices*). Moreover, LW-CoEdge includes a 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, 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.

Figure 1 illustrates the LWCoEdge components, the third-party components, their services, and relationships, as well as the tier in which they are deployed considering the Cloud, Edge, and Sensor tier of our 3-tier architecture. Our architecture builds on two technology assets: lightweight virtualization and microservices. Lightweight virtualization (Morabito et al., 2018) allows packaging and deploying our components in containers. Furthermore, we use the microservices concepts (Thönes, 2015) to develop the components of the proposed architecture. Hence, they become lightweight-

components since they implement a unique responsibility and are appropriate to be packaged in lightweight-images. Such a mix of technology assets is suitable to run on resource-constrained devices.

As third-party components, we adopted FIWARE (FIWARE GE, 2019). Following the FIWARE programing model, all interactions with GEs and between GEs use the FIWARE NGSI protocol. In LW-CoEdge, we used the GE Device Management (FIWARE GE, 2019) and GE IoT Agent (FIWARE GE, 2019) to perform tasks related to the physical environment, such as getting sensing data, performing actuation, and managing and communicating with the end devices. In addition, the GE Orion Broker (FIWARE GE, 2019) and the GE Short Time Historic (FIWARE GE, 2019) from FIWARE Data/Context Management are used to enable the data persistence.

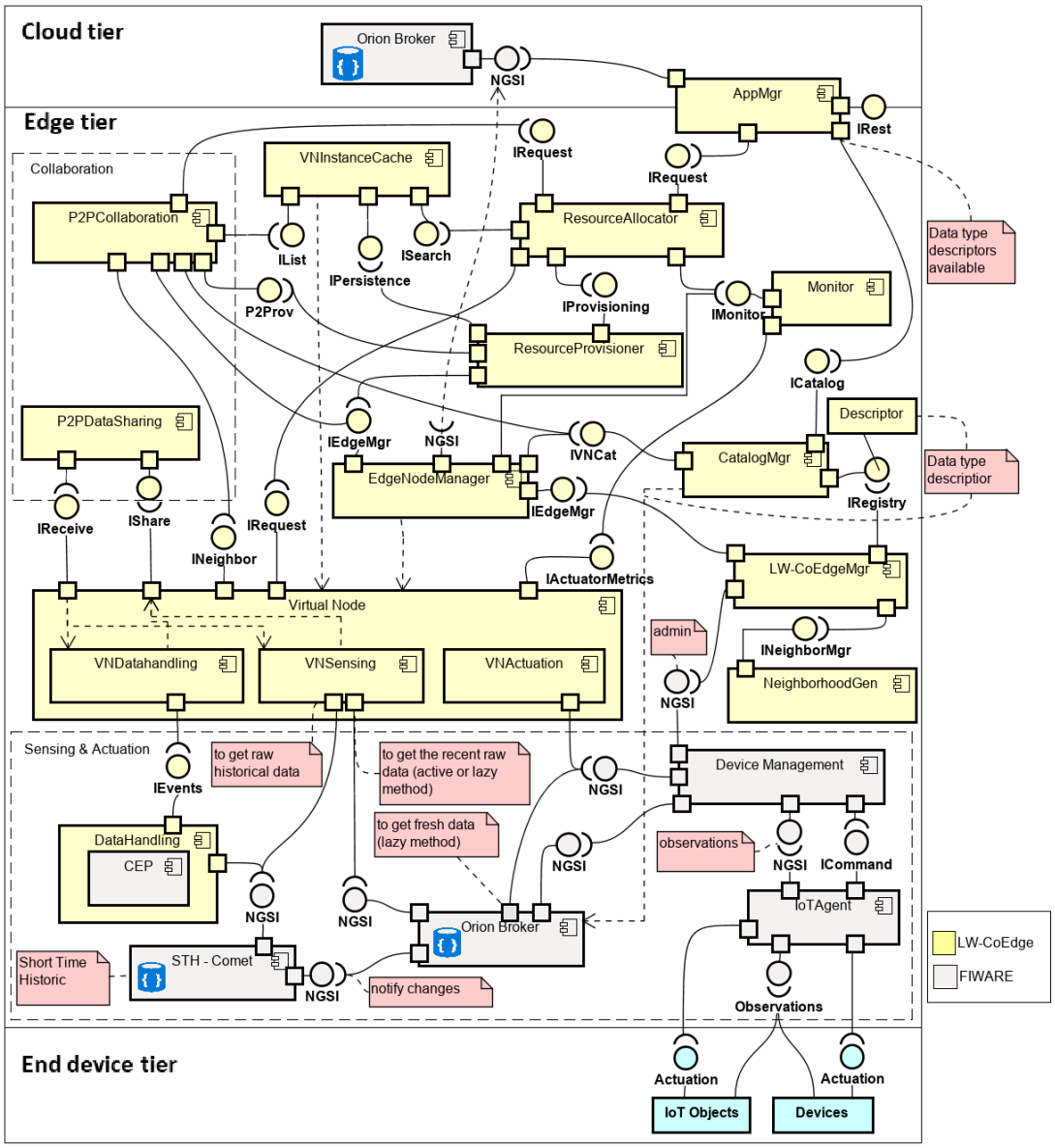

Figure 1. Architectural model

During the boot of the edge device, except the virtual node container, all the other components of the architecture are initialized. During the boot, the **EdgeNodeManager**

component loads the basic settings of the edge device and their neighbors and makes these settings available to other components through some APIs. Moreover, it also sends these settings to the Orion Broker at the Cloud tier to be used by **AppMgr**. Furthermore, the GE IoTAgent begins to receive observations (or raw sensing data) from the End device tier and sends them to the GE Orion Broker at the Edge tier.

Following the strategy behind our virtualization model, at the time an application request arrives in the system via an entry-point (at the Cloud or Edge tier), the resource allocator component allocates a VN to meet such a request. As the VN receives a new request to process, it can either interact with FIWARE GEs to get fresh data (raw or processed), whenever the last data served is not valid (above a predefined data freshness threshold) or perform actuation on the physical environment. Moreover, whenever the VN has fresh data, it invokes the **P2PDataSharing** to share the new data with its neighbor VNs. At the end, the VN sends the processing result to the request issuer.

It is important to notice that, since in our architecture the components are microservices packaged in lightweight images, we assume that each edge node already has the container images necessary to run the VN to meet the respective datatype. In this way, we avoid incurring in any network overhead and save bandwidth by not transferring container images between edge nodes when the resource provisioning needs to provision a new VN. In the next section, we describe the behavior of each building block, their services, and the relationship among them.

3.1. The Building Blocks, their Services, and Relationships

In our architecture, the cloud is used to host and run GE Orion Broker and AppMgr. **GE Orion Broker** (FIWARE GE, 2019) is the reference implementation of the Publish/Subscribe Context Broker in charge of all data management in LW-CoEdge. Its data persistence mechanism uses the MongoDB database (https://www.mongodb.com/). MongoDB is an open-source document-oriented and distributed database (NoSQL) built for the cloud era, and ideal for storing distinct data structures from several sources. This GE provides an interface containing services to manage the lifecycle and distribute context information (e.g., temperature, presence). **AppMgr** is the component in charge of providing management services, discovering of datatypes and service to application submit their requests. It presents both a cloud and an edge version. The AppMgr cloud version has two essential functions. The first allows discovering and getting the list of the available datatype descriptors in the global repository. The second allows the applications to submit requests to the CoT system.

Regarding the Edge tier, we can divide it into main groups of components, which are: Data Management, Resource Management, Collaboration, Catalog Management, Virtual Node, Device Management and Infrastructure of communication. We also provide the components EdgeNodeManager, Monitor, NeighborhoodGen, VNInstanceCache and AppMgr edge version. Data Management group encompasses the GE Orion Broker and FIWARE Short Time Historic (GE STH-Comet). GE Orion Broker at the edge network has two goals. The first goal is to serve as the repository for storing datatype descriptors. The second goal is to store the raw sensing data from sensors. GE STH-Comet (FIWARE GE, 2019) component enables raw historical data in our architecture at the edge network. It provides a set of services through NGSI interface to save, retrieve and remove historical

context data and aggregated time series information regarding the evolution in time of context data registered in the GE Orion Broker.

EdgeNodeManager oversees the provision of essential services to support the execution of other components such as ResourceProvisioner, P2PCollaboration, and LW-CoEdgeMgr through the IEdgeMgr interface. These services are responsible for performing the following tasks: (i) managing the virtual node lifecycle (container's start and stop), (ii) configuring the edge node settings and its neighbors, (iii) scaling (up and down) containers, and (iv) providing the minimum resources required to run each VN component. The settings regarding the neighborhood are generated using the **NeighborhoodGen** component. Moreover, EdgeNodeManager interacts with the **CatalogMgr** using the IVNCat interface to get the details about the datatype to start the container. Also, it interacts with the GE Orion Broker at the cloud using the NGSI interface to register the edge node settings (e.g., hostname, IP, location, etc.).

AppMgr component deployed in the edge tier (edge version) has two essential functions. The first function allows discovering and getting the list of the available datatype descriptors in the local repository. The second function allows the applications to submit their requests to the LW-CoEdge through the edge node.

Collaboration group encompasses two components namely P2PCollaboration and P2PDataSharing. **P2PCollaboration** is the component in charge of implementing the algorithm (Alves et al., 2019, pg. 27) to enable the collaboration between edge nodes. It provides the P2Prov interface composed of two services. The first service is responsible for receiving the request forwarded by the resource provisioning component and then performs two steps: (i) finding a neighbor Edge Node with the best resources for running the VN, and (ii) sending the request to be served by the neighbor node. The second service is used to register new VNs to perform the data sharing by invoking the appropriate VN through INeighbor. Moreover, the **P2PCollaboration** component interacts with EdgeNodeManager through the IEdgeMgr interface to get the neighboring nodes as well as the needed resources to instantiate and run a VN container. **P2PCollaboration** uses these set of information to choose the best neighboring node to provide a VN. The **P2PDataSharing** component implements the algorithm (Alves et al., 2019, pg. 29) to execute the data sharing among VNs. This service is provided through the IShare interface and a VN invokes it to share fresh data with its neighbors.

Resource Management group encompasses two components namely ResourceAllocator and ResourceProvisioner. **ResourceAllocator (RA)** is the component in charge of implementing the algorithm (Alves et al., 2019, pg. 24) responsible for providing an instance of the Virtual Node (VN) to meet each application request submitted by the AppMgr component (both at cloud and edge). Initially, RA interacts with the **VNInstanceCache** component to find a VN matching the application request using the *search* method from the ISearch interface. Then, if there is an instance of the VN, and the container has available resources, RA forwards the request to process in it. When it is not possible to find a VN, the RA invokes the **ResourceProvisioner**.

ResourceProvisioner (RP) is the component in charge of implementing the algorithm (Alves et al., 2019, pg. 25) for the provision of a new VN instance using the provisioning service through the IProvisioning interface. This service executes essential functions described as follows. The VN container deployment is the first function to execute. To fulfill deploy, RP interacts with **EdgeNodeManager** through the IEdgeMgr

interface to verify if the edge node has enough resources for provisioning a new VN, and then performs the deployment. RP then registers the new VN instance in the pool using the *register* method provided by the **VNInstanceCache** component through the IPersistence interface. In the end, RP interacts with **P2PCollaboration** through the P2Prov interface to register the new VN for collaboration (data sharing). Whenever the RP component is unable to provision a new VN instance, it forwards the application request to the **P2PCollaboration** using a service available in P2Prov. The service finds a neighboring edge node capable of hosting this VN. Furthermore, RP interacts with **EdgeNodeManager** to use the scale-up service when the edge node has enough resources to increase the performance of a running container to meet more requests.

Virtual Node group encompasses the three types of VN designed to handle the application requests, namely **VNSensing**, **VNDatahandling** and **VNActuation**. These three types of VNs expose two interfaces: INeighbor and IRequest. The INeighbor is used by **P2PCollaboration** to register new VN instances for collaboration. The IRequest interface is used by the **ResourceAllocatior** component to forward the application request for processing. Meanwhile, the VN of the type VNSensing and VNDatahandling interacts with the **P2PDataSharing** component through the IShare interface to send their fresh data to the neighboring VNs. Moreover, they expose the IReceive interface to receive the fresh data shared. Also, the VN operations are supported by engaging the FIWARE GEs through NGSI interface and LW-CoEdge components. The interaction among these components is described as follows.

The VN type were defined keeping in mind the requirements and operating models of most IoT applications. In this sense, we understood that the generic goals of these applications will always be to obtain sensing data from the physical world (current or historical), to receive notifications about events that occur in the physical world, or to act on it. **VNSensing** interacts with the GEs to get recent or historical raw data from the database maintained at the Edge tier or fresh data directly from the sensors (IoT devices). The VN invokes the GE Orion Broker to get recent raw data or fresh data. Concerning the historical data, it is obtained by invoking the GE STH-Comet. **VNActuation** interacts with GE Device Management to send actuation commands to the physical device. **VNDatahandling** interacts with LW-CoEdge **DataHandling** (DH) component using the IEvents interface to provide value-added information. DH component abstracts the complexity of the information processing executing queries through a Complex Event Processing (CEP) engine (FIWARE GE, 2019).

Device management and communication infrastructure group provides the components GE Device Management, GE IoT Agent, and **LW-CoEdgeMgr**. They are responsible for all interaction with the physical environment. GE Device Management (DM) (FIWARE GE, 2019) is the component in charge of mediating the interaction of LW-CoEdge components (Virtual Nodes and LW-CoEdgeMgr) and other GEs with the physical environment through the GE IoT Agent component. GE IoT Agent (FIWARE GE, 2019) is the component in charge of enabling the communication between LW-CoEdge components and physical environment, handling both standard and proprietary protocols. **LW-CoEdgeMgr** component is used by the Service Provider Manager (SPM) to manage the physical devices, interacting with the DM component through the administration services available in NGSI interface. Thus, the SPM submits payloads of configurations to both create the services and register the end devices. Such

configurations allow devices to establish communication for sending sensing data (or observations) to the system or performing actuation.

Catalog Management group encompasses only the **CatalogMgr**. It is used to manage and provide a datatype descriptor from the descriptor repository through three services interfaces: IVNCat, IRegistry, and ICatalog. ICatalog contains the service used to provide the list (either full or by specific geolocation) of descriptors. IVNCat contains the service to get details about the datatype. Finally, IRegistry provides the services for registering, updating, or removing a datatype descriptor from the repository.

In this section, we described the LW-CoEdge architecture and the relationship between its building blocks. Furthermore, to enable the operation between LW-CoEdge and FIWARE, it is necessary to establish the relationship between their models. The complete description of the models and relationship is found in https://bit.ly/3855aD4.

4. Evaluation

We adopted the Goal Question Metric (GQM) methodology (Basili, 1992) for planning our evaluation. The goals, questions, and the derived metrics used in our work are presented in Table 1. Goal **G1** is to analyze LW-CoEdge, for the purpose of evaluating its distributed resource management with the collaboration process enabled, with respect to the impact caused by a network with high communication latency values, from the viewpoint of the end-user. Goal **G2** is to analyze LW-CoEdge, for the purpose of evaluating its distributed resource management algorithms, concerning the fault tolerance in relation to the physical end devices to meet application requests. From these two goals, we derive the relevant questions described in Table 1. Table 2 shows a set of metrics that are collected from our CoT system. 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 even if the application RTTh is not met.

Table 1. Questions of GQM

Question (Q)		Goal (G)
Q1	Does collaboration between virtual nodes help meeting requests within the application-defined response time threshold?	G1
Q2	Does collaboration between virtual nodes help meeting requests even if there is an end device failure?	G2

Table 2. 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.	
M3	TTS	Total time spent to meet a given request	
M4	R_ENDDEV	Requests met using data obtained from end devices	Q2
M5	R_MEM_CACHE	Requests met using data from Virtual Node memory cache	
M6	R_TEMP_DB	Requests met using data obtained from the local database	

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.

Total time spent (TTS) is the sum of the time periods (a) since receiving a request to process at the same edge node, (b) to perform the collaboration (receiving the request, finding a neighbor node, and forwarding the request to the neighbor node) if necessary, and (c) communication to send the request to the neighbor node. The request is adequately met when the datatype required is provided in the system, and the value of TTS to achieve it is less than or equal to the application response time threshold.

To answer question Q2, we assess the number of requests met using data of each data-source. Thus, we selected the following metrics. R_ENDDEV (M4) records the number of requests served using sensing data read directly from the end device. R_MEM_CACHE (M5) records the number of requests served using sensing data obtained from the VN cache. Lastly, R_TEMP_DB (M6) records the number of requests served using sensing data collected from the local temporary database.

4.1. Proof-of-Concept Implementation

To achieve the defined evaluation goals, we used a Proof-of-Concept (PoC) that concretizes the components of our three-tier architecture. This PoC was used in our previous work (Alves et al., 2019, pg. 34), where further detail about it can be found. In this section, we describe the initial setting values used in this current work to start the environment before running the experiments. Initially, we need to set the hardware and deploy all software necessary to run our CoT system. In Table 3, we summarized the most significant hardware and software settings.

Table 3. Summary of parameters used in LW-CoEdge

Parameter	Value
Virtual Machine software	VMware
Operational system in VM	CentOS 7 64-bit
Edge node (EN) hardware	2.3 GB RAM, 10 GB HD space and 2 CPUs model i7-8550U of 1.80Ghz
Local network link capacity (Edge tier)	10/100Mbps
Container-based solution	Docker
Java virtual machine	Java VM 64-bits version “11.0.3 2019-04-16 LTS”
End device virtualization	FIWARE figway components (https://github.com/telefonicaid/fiware-figway)
Number of Edge nodes (EN)	5
Number of End devices	4
Delay for querying local database (FIWARE)	(0,972±0,019s)
Virtual Node output data size	72 Bytes
Application request size	140 Bytes (average)
Sensing data size – (SDsize) – FIWARE	122 bytes ± 0.1191 KB
Time to feed the local database	30(seconds)
Maximum freshness	5 (seconds)
Communication latency between ENs	Depends on the experiment
The average power consumption of devices	20w

Next, with the edge nodes VMware configured, the following configuration files should be deployed: (a) NeededResources, (b) Edgenode, (c) Catalog of datatype descriptors, and (d) LW-CoEdge components ports. Item (b) should be changed for each edge node mainly regarding the hostname, IP, and its neighborhood. Item (c) should be

changed depending on the sensors connected on the edge node. Examples of these file configuration are available in https://bit.ly/2LryLwV.

To execute the system, the components that encompass the architecture should be packaged into Docker images and deployed for every Edge node. The script files to generate the images are available in: https://bit.ly/2rVYzdN. Regarding the third-party components (FIWARE) used, they are available in the FIWARE Catalogue (FIWARE GE, 2019). More detail in https://bit.ly/2DRtwSU. In this current work, we considered the same smart city/smart building application and the same desktop computer described in our previous work (Alves et al., 2019, pg. 36) to run the new experiments.

5. Experiments and Results

In this section, we describe the scenario of experiments, along with the analysis of the achieved results. For our hypothetical scenario, we considered an area of 800x400 meters representing a distribution center of a retail company at the Rio de Janeiro city where the sensors (temperature and smoke) of the end device tier are deployed. At this location, the company stores cartons that are highly flammable. Due to the climate conditions of the city, the distribution center faces high temperatures, which affect the equipment used for order processing. Thus, the company deployed an application to monitor the environment temperature and regulate the cooling system automatically. Moreover, the system is used to detect any signal of smoke that represents 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 sensors are displayed on monitors scattered in the distribution center and are accessible from mobile devices. Table 4 describes the configuration concerning the datatypes and the neighborhood considered in the scenario. For instance, the edge node EN0 was configured to provide the datatypes UFRJ.UbicompLab.temperature (DT1) and UFRJ.UbicompLab.smoke (DT4) and its neighbors are the edge nodes EN1 and EN4, and so on.

Table 4. Summary of Edge Nodes, datatypes and neighborhood configuration

Edge Node	Datatype		Neighbor EN
EN0	DT1	DT4	EN1, EN4
EN1	DT1	DT4	EN0, EN2, EN3
EN2	DT1	DT4	EN1, EN3
EN3	DT1	DT4	EN1, EN2, EN4
EN4	DT1	DT4	EN0, EN3

In this scenario, experiments E1 and E2 were created to evaluate the behavior of our algorithms under more rigorous operating conditions than the experiments performed in our previous work (Alves et al., pg. 37). These experiments help to answer the defined GQM questions beforementioned. To do so, we modified relevant system settings in the edge nodes concerning the datatypes, and neighborhood. Unlike the scenario evaluated in our previous work, in this new configuration the edge nodes are homogeneous regarding the provided datatypes, allowing more effective demonstration of the collaboration behavior (e.g., load balancing) when edge nodes are overloaded. Moreover, each result of the experiments was achieved after ten rounds of execution.

5.1. Experiment E1

We designed this experiment to assess the impact that a network with high communication latency causes in our distributed resource management.

Table 5 presents the measured real latency of the network obtained by executing the ping command between two edge nodes. Therefore, the experiment was run for each configured value of latency, in order to measure the response time spent to meet the application requests and to analyze how the latency affects the computation time of our collaboration process. Moreover, in each execution of the experiment, a total of 30000 application requests were produced and these requests were distributed among the edge nodes randomly. To trigger the collaboration quickly, we will simulate the resource exhaustion of the VN using the parameter of memory available (heap memory) for processing. For this, this parameter was set to 12% for the edge nodes EN1 and EN3. We chose such edge nodes because of their neighborhood, thereby allowing a higher number of requests to be forwarded for collaboration. Therefore, whenever the available memory is less than or equal to 12% the collaboration is executed.

In Figure 2, each label of the X-axis represents a communication latency configured in the network when the experiment was run. For instance, the number 20 denotes a latency value of about 20ms; the number 30 represents a latency of around 30ms, and so on. Figure 2(a) illustrates the average total time spent (TTS) to meet all the requests submitted by the applications. As we can see in the figure, as the latency of the network increases, the time to fulfill a request also increases, as expected. This behavior is due to the time spent by the collaboration process to find a neighbor node to forward a request (Figure 2(b)). Moreover, we also can observe in Figure 2(a) that times oscillate either up or down due to the number of requests handled by collaboration in each execution, as we can see in Figure 2(c).

Table 5. Measured latency of the network

Point of the figure (X-axis)	Real measured latency (ms)	Point of the figure (X-axis)	Real measured latency (ms)
20	23.46	80	83.80
30	32.96	90	93.81
40	43.84	100	105.28
50	54.65	110	111.79
60	63.50	120	123.40
70	73.50	130	133.76

Let's observe Figure 2(b) that shows only the requests that were handled by the collaboration process. For a latency of around 20ms, the measured TTS to meet the application requests is 165ms, for a latency of around 30ms the TTS is 216ms, and so on. This value increases for each execution of the experiment. For instance, when the request is received by EN1 and it has not enough resources to meet it, the collaboration is invoked, then, it seeks the best edge node within the neighborhood of EN1 to forward the request to (in this experiment, three neighbor nodes). In this process of finding the best neighbor node, the collaboration spent an average time of 70.38ms for a real latency of 23.46ms (Table 5). Besides, this time is increased with the computational time that the VN uses to process the request (valid for all values in the figure). Therefore, we can infer that two factors, when combined, can affect the total computing times to meet a request. The first is network latency. The second is the number of nodes to be queried within a neighborhood. Therefore, the greater the neighborhood of the edge node, the latency value of the communication network between the edge nodes should be as small as possible.

Furthermore, the negative effect of the high latency on the collaboration can be visualized when we compare the values measured of TTS of Figure 2(b) for each latency

configured. For instance, for the latency of around 20ms, the TTS measured is 165ms, whereas for the latency of around 40ms the TTS is 226ms. By observing the Figure 2(c), for these latency values the number of requests met is 6157 and 6053 respectively. Therefore, the experiment performed with 40ms latency showed an average increase of 37% in the TTS value to meet fewer requests.

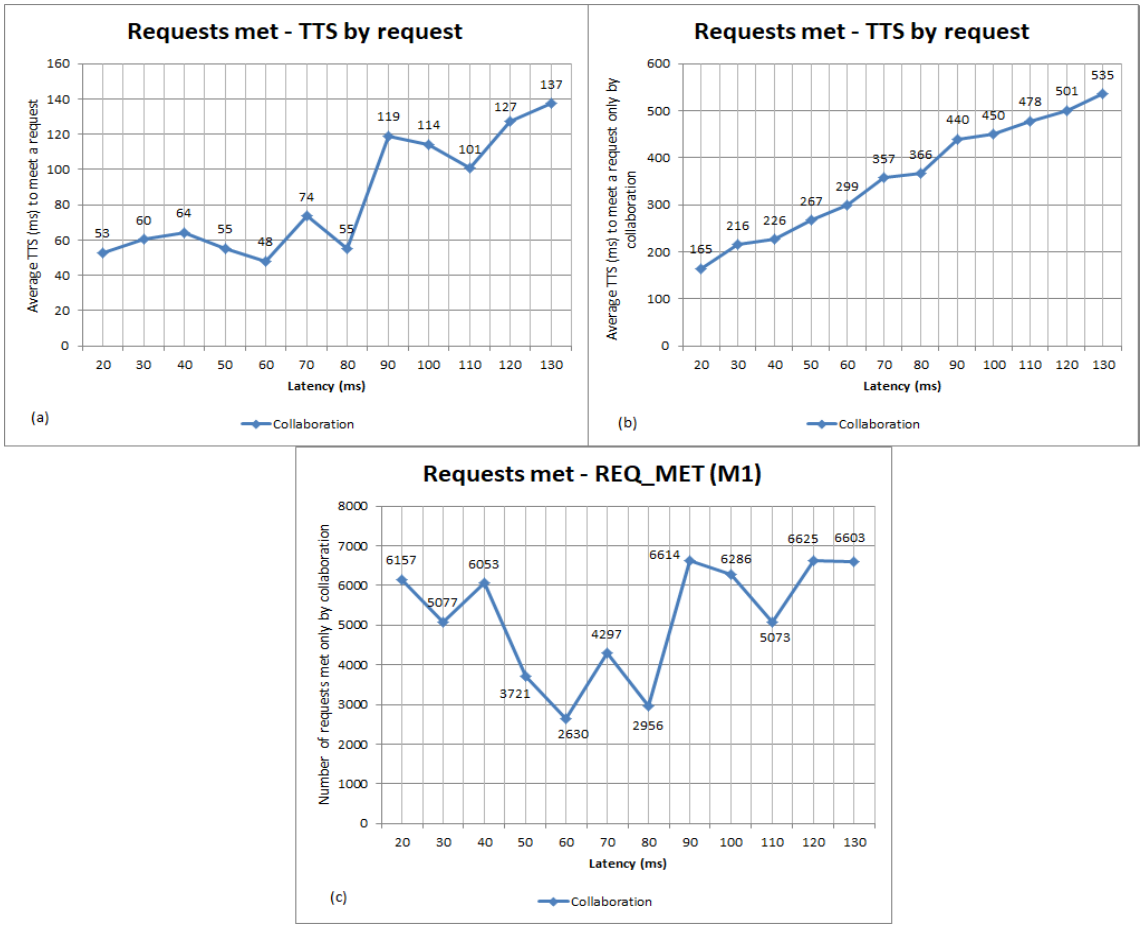

Figure 2. (a) Average total time spent (TTS) for all the requests met. (b) A view of the average TTS only of the requests met by the collaboration process. (c) Number of requests met using only by the collaboration

Concerning the GQM defined to assess our work, the results of the experiment E1 showed that the goal G1, in relation to the collaboration and network latency values, was properly achieved, even in adverse conditions of the computational infrastructure with edge nodes overloaded and network with high values of latency.

5.2. Experiment E2

A fault-tolerant system continues to operate correctly despite the failure of some of its components. We designed this experiment to assess the ability of our collaborative approach to meet application requests, even in the presence of connectivity failures between edge nodes and end devices.The operation of the experiment E2 is similar to the E1 with modifications regarding some parameters used to run it. First, the latency between ENs was set to 20ms. Second, a total of 15070 requests to the datatype of Temperature were generated and submitted to the edge nodes randomly. In Figure 3, each label of the X-axis represents an edge node configured to meet application requests. For each edge

node, we show three-bar values representing the source of the sensing data used to serve the application. The first bar is the value obtained directly from the end device, the second bar is the value obtained from the VN cache, and the last one is the value obtained from the temporary local database.

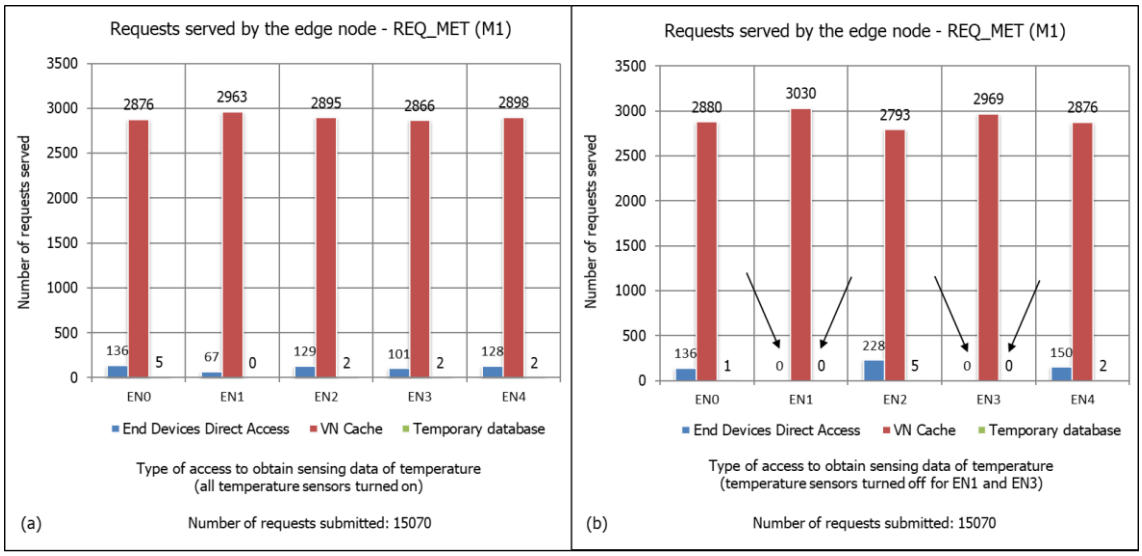

Figure 3. (a) Requests served using all temperature sensors turned on. (b) Requests served with the temperature sensors turned off for edge devices EN1 and EN2.

The execution of the experiment E2 was divided into two parts. Figure 3(a), shows the requests met during the first execution with all end devices enabled. We can see in the first point of the figure (representing the EN0) that our system can serve 136 requests using data obtained directly from end devices, a total of 2963 requests using data from the VN cache, and only 5 requests using data obtained from the temporary local database. This behavior is observed up to the last point in the figure. In turn, Figure 3(b) shows the second execution, in which we disconnected the end device that provides the datatype of temperature to the edge device EN1 and EN3.

Let's observe the point 2 (EN1) and point 4 (EN3) in both figures. In Figure 3(a), the application requests are met using all available data sources. Figure 3(b) also shows that all the application requests submitted are successfully served. However, edge nodes EN1 and EN3 met requests using only the updated sensing data of temperature from the cache once the end device was disconnected. This operation was possible thanks to the collaboration algorithm implementing the data sharing between the edge nodes. Finally, concerning the collaboration and fault-tolerance, the results of the experiment E2 showed that the goal G2 was achieved.

6. Conclusion

In this paper, we introduce the implementation of a novel architecture that takes advantage of the collaboration features of edge nodes to provide high quality services for running IoT applications. Our architecture is based on a lightweight device virtualization model and a flat collaboration model between edge nodes. In addition to describing the concrete components of the proposed architecture, we present the results of an experimental evaluation with a well-defined focus: we aimed to prove that the system operates well even under fault conditions and high latency networks. The results of the performed experiments showed that our system can meet the requests in adverse

conditions of the network infrastructure (high latency). Moreover, even in the presence of disconnections between edge nodes and end devices, either permanently or temporarily, the edge node is able to provide updated sensing data from the internal cache thanks to our data sharing mechanism implemented. Thus, our novel architectural model can handle time critical applications and is fault tolerant. Therefore, the results of the performed experiments were promising and validate the feasibility of our proposal.

Acknowledgements

This work was partially funded by FAPESP (grant number 2015 / 24144-7). Flavia C. Delicato and Paulo F. Pires are CNPq fellows.

References

Alves, M. P. et al. (2019) “LW-CoEdge: a lightweight virtualization model and collaboration process for edge computing”. In: World Wide Web, 1-49.

Armbrust, M. et al. (2010) “A view of cloud computing”. In: Communications of the ACM, 53(4), 50-58.

Basili, V. R. (1992) “Software modeling and measurement: the Goal/Question/Metric paradigm”.

Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. (2014) “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.

Cavalcante, E. et al. (2016) “On the interplay of Internet of Things and Cloud Computing: A systematic mapping study”. In: Computer Communications,89, 17-33.

FIWARE GE (2019) “Generic Enablers”. Available in: https://catalogue-server.fiware.org/. Last accessed: 07/07/2019.

Madria, S., Kumar, V., and Dalvi, R. (2014) “Sensor cloud: A cloud of virtual sensors”. In: IEEE software, 31(2), 70-77, 2014.

Morabito, R. et al. (2018) “Consolidate IoT Edge Computing with Lightweight Virtualization”. In: IEEE Network, 32(1), 102-111.

Sahni, Y. et al. (2017) “Edge Mesh: A New Paradigm to Enable Distributed Intelligence in Internet of Things”. In: IEEE Access, *5*, 16441-16458.

Santos, I. L., Pirmez, L., Delicato, F. C., Khan, S. U. and Zomaya, A. Y. (2015) “Olympus: The cloud of sensors”. In: IEEE Cloud Computing, 2(2), 48-56, 2015.

Santos, I. L. et al. (2019) “Zeus: A resource allocation algorithm for the cloud of sensors”. In: Future Generation Computer Systems, 92, 564-581.

Shen, Z. et al. (2019) “ICCF: An Information-Centric Collaborative Fog Platform for Building Energy Management Systems”. In: IEEE Access.

Thönes, J. (2015) “Microservices”. In: IEEE Software 32.1: 116-116.

Wang, N. et al. (2017) “ENORM: A framework for edge node resource management”. In: IEEE Transactions on Services Computing.