This is the peer reviewed version of the paper, which has been published in final form at https://doi.org/10.1007/978-3-030-12388-8-27.

Latency-Aware distributed resource provisioning for deploying IoT applications at the edge of the network

Cosmin Avasalcai
Distributed Systems Group
TU Wien, Vienna, Austria
c.avasalcai@dsg.tuwien.ac.at

Schahram Dustdar
Distributed Systems Group
TU Wien, Vienna, Austria
dustdar@dsg.tuwien.ac.at

***Abstract*—With the increased success of Internet of Things (IoT), the conventional centralized cloud computing is encountering severe challenges (e.g., high latency, non-adaptive machine type of communication), that proved insufficient to meet the stringent requirements of IoT applications. Besides requiring fast response time, increased security and privacy, they lack computational resources at the edge of the network. Motivated to solve these challenges, new technologies are driving a trend that distributes the computational resources and shifts the function of centralized cloud computing to the edge. Several edge computing technologies, edge and fog paradigms, originating from different backgrounds have been emerging to overweight these challenges. However, to fully utilize these limited devices, we need advanced resource management techniques. In this paper, we present a novel distributed resource allocation algorithm with the purpose of enabling seamless integration and deployment of different applications in an IoT infrastructure. The algorithm decides: *(i) the mapping* of an IoT application at the edge of the network; *(ii) dynamic migration* of parts of the application, such that Service Level Agreement (SLA) is satisfied. Furthermore, we analyze and discuss our approach and the potential to minimize the latency of different IoT applications.**

***Keywords*—Resource Management; Edge Computing; Fog Computing; Internet of Things;**

I. Introduction

Over the past decades, cloud computing has been applied in most of the industries due to its high cost-efficiency and flexibility achieved through consolidation, in which computing, storage, and network management functions work in a centralized manner. Since the number of connected devices in the IoT has increased dramatically in the last couple of years generating more and more data, the existing centralized cloud computing architecture is encountering severe challenges. For instance, transferring all this data to the cloud introduces congestion in the network and extra delays in the application. Hence, deploying a real-time IoT application, which requires fast response times and increased security, entirely to the cloud is not an effective strategy anymore. As a result, the need for processing applications closer to the edge of the network has become a necessity [1].

Shifting the storage and computational power from the cloud closer to the edge provides a series of benefits, such as smaller end-to-end (e2e) delay, better scalability of the network, enhancing privacy and independence of the cloud [2]. Applications such as self-driving cars [3] and aid for people with disabilities [4] benefits from having a minimum e2e delay and fast response time. For example, when care services are extended from hospital to private homes with a focus on remote monitoring of people suffering from chronic diseases or old people, in case of emergencies, the faster the response time is, the immediate help can be provided (e.g., the command sent to the sensors should be granted immediately, stabilizing the patient, until a nurse will arrive). Moreover, there are applications where security and privacy have an important role like ambient assisted living [5]. Finally, since most of the computation is performed at the edge, only a small portion of generated data is sent to the cloud for further processing. This greatly impacts the network overhead and offers correct functionality even when the connection to the cloud is lost.

The underlying premise of new paradigms like fog [6] and edge [7] computing is to deploy distributed heterogeneous computational resources closer to the IoT sensors. We differentiate the two by the resource capabilities and location. For example, edge devices are placed closer to the source of data and have limited resources (e.g., limited computational power, limited energy). On the other hand, fog devices have more computational resources, being located between edge devices and cloud. However, deploying applications in such diverse IoT infrastructure where mobile devices can come and go without prior notice is impossible, if there is no support of novel resource management techniques.

Since the cloud has almost unlimited but far away resources, the *resource management* comes as a solution to the constraints imposed by the IoT devices. Deploying IoT applications at the edge comes with a set of challenges. The devices in the network are heterogeneous, with limited resources (i.e., limited computational power and energy supply) and are subject to different environments. Moreover, the nodes can be mobile introducing even more uncertainty in the system, since nodes can enter or leave the network without any prior announcement. In edge computing, the diversity found in such networks and resource management play a significant role in assigning and distributing tasks (that are generated by local devices), to the remote cloud (the center of the network) or local servers/devices (the edge of the network). A common approach for resource management in edge computing is to assign tasks to the remote cloud or local servers according to several factors such as energy, bandwidth consumption, having

as final scope the minimization of the latency.

In this paper, we introduce a decentralized resource management algorithm with the purpose of deploying IoT applications at the edge of the network such that e2e delay is minimized. The algorithm tackles resource allocation [8] from two different perspective, where to place an application and when to migrate. This ensures that the system is able to dynamically adapt to the uncertainties of the IoT architecture and provide a better availability of applications deployed at the edge of the network. Furthermore, to aid the algorithm in deciding what tasks to migrate, we propose a task ranking system that offers a classification of the tasks on a node based on some criteria (e.g., battery level, trustiness and communication latency among others). This is important, especially for being able to handle variations in resource demand while assuring a good quality of service (QoS) [9].

The remainder of the paper is structured as follows: In Section II we discuss the related work on resource management techniques. Section III defines the IoT application and architecture considered in this paper. In Section IV we describe the implementation details of our proposed algorithm. Section V present the challenges of such a framework and discuss the evaluation we intend to make. Finally, Section VI concludes the paper and provides an outlook on future work.

II. Related work

With the advent of new paradigms, like edge and fog computing and the tremendous impact IoT devices is having on people lives, researcher propose new solutions to take advantages of all available resources found closer to the IoT sensors.

In the context of using the available computational resources, the authors in [10] presents a mapping algorithm that deploys a pre-partition application at the edge of the network, reducing the cloud communication and minimizing the latency. Another deployment algorithm composed of two stages, (i) partitions the IoT application in multiple tasks and annotates them with location information, (ii) place the obtained tasks on multiple edge nodes based on their location, is described in [11].

In [12], the authors suggest a cooperative fog platform using a distributed communication model. The purpose of the platform is to achieve a better collaboration between multiple static and mobile fog devices. Moreover, to improve the service efficiency of IoT applications, an allocation algorithm is applied by selecting hosts based on system characteristics. A similar approach is presented in [13] where an optimization service placement algorithm is developed to share fog resources. However, the algorithm will always try to map an application to the deployment node or to a neighbor device. In case of failure, the application is mapped to the cloud.

Another important topic in resource allocation that helps adapt to the increased uncertainty is resource migration. In [14] the authors present an algorithm to dynamically migrate virtual machines (VMs) and find the best communication paths based on predictions of user movements. Another solution to service migration is presented in [15], where the authors propose an edge-enabled publish-subscribe middleware to continuously monitor the QoS and transparently migrate clients to another host in close proximity. For our self-adaption solution, we propose a similar approach that finds the latency of nearby neighbors devices and solves the problem of mobile nodes unpredictability. However, in comparison, our approach supports more diverse IoT applications.

Multiple other solutions that tackle the challenges of deploying IoT applications at the edge have been proposed. The authors in [16] define algorithms to find the best location to distribute applications in a cloud-assisted vehicular network architecture. Others have proposed adaptive workload management mechanisms [17] or resource estimation techniques [18] to provide services closer to the edge similar to those in the cloud. Furthermore, the authors in [19] introduce a solution to minimize the response time of video analytics applications close to the edge.

With respect to aforementioned research papers, most of the researchers work on emphasizing the placement of edge devices, whilst load distribution is considered to be a hot topic. In contrast, our proposal provides a more comprehensive resource management solution that is independent of IoT applications and environment. By combining initial placement of the application at runtime, with dynamic migration and neighborhood selection, the edge devices will become more intelligent.

III. IoT infrastructure

The proposed solution is a decentralized algorithm designed to ensure that deployed IoT applications meet their SLA, i.e., the maximum E2E delay for an application to function as intended. The framework is divided into three different modules deployed on each computational device in an IoT architecture.

A. IoT architecture

For the architecture, we consider that every edge, fog device, and cloud are connected in a peer-to-peer manner. This changes the pyramidal approach where there are four layers (i.e., sensor, edge, fog, and cloud) into a more flatter approach where devices have a direct connection to different cloud providers and to all nearby nodes (see Figure 1).

B. IoT application model

Due to the limited computational power of edge devices, we model an IoT application as a Directed Acyclic Graph (DAG) where vertices represent different tasks (i.e., a set of instruction that performs a specific computation) and edges show the dependencies between them. Each task is characterized by its computational requirements (i.e., CPU utilization, RAM, storage) and the address of the dependent tasks. An example of such an IoT application is presented in Figure 2.

Moreover, to deploy the application in such a heterogeneous IoT network, the tasks are placed in individual docker containers. A docker container represents a lightweight, standalone, executable package that contains everything needed to run the specifically added task [20]. Furthermore, since tasks are isolated from the environment, a better security and a fast and easy deployment are ensured.

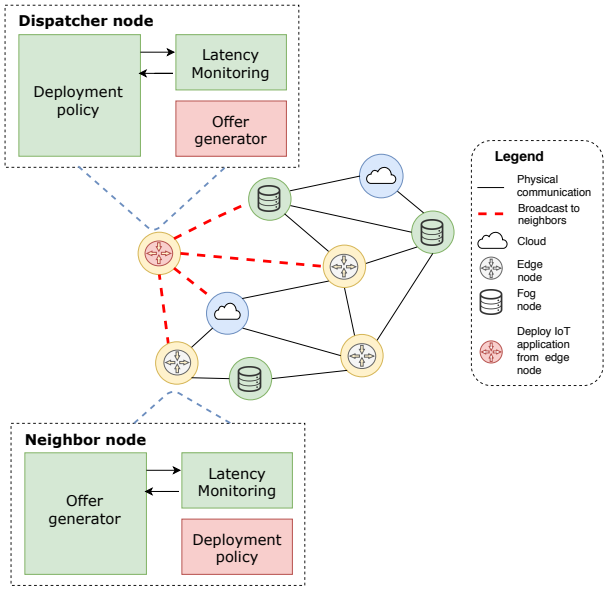

Fig. 1: Our system in an IoT architecture

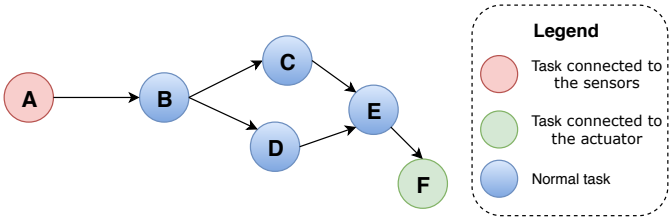

Fig. 2: IoT application model

In this paper, we assume that the application is divided into multiple dockers by the developer and sent directly to the deployment edge node once everything is prepared.

IV. Resource Management Solution

We design an algorithm to dynamically distribute and adapt the application such that the e2e delay is minimized. In our approach, the algorithm is deployed on every edge and fog node in the IoT architecture, being divided into three modules each performing a different functionality as depicted in Figure 1. Moreover, the algorithm is developed to work on two different states: *(i) deployment state* represents the place from where an application is deployed (i.e., the dispatcher node) and a *(ii) resource sharing state* where offers are generated to receive tasks for further processing (i.e., the neighbor node). In the deployment state, the deployment policy is aided by the latency module to find the best placement for the IoT applications, while the offer generator module is not working. In contrast, in the resource sharing state the offer generator module switch places with the deployment policy to choose the tasks and compute the offers.

The functionality of our algorithm is inspired from a real-world private auction house rules. Once an item has arrived at the auction house and is ready to be sold, a list of people is created, who are invited to the event. These persons are not randomly chosen since they have to fulfill some criteria and prove that own the necessary "resources" to pay for the items. Once the item placed for auction, the interested buyers can place offers in a limited period of time. In the end, the buyer who offered the most is the winner of the object.

In our case, the IoT devices found in close proximity to the dispatcher are eligible to participate. However, since each IoT application is different, the latency monitoring module creates a personalized list of invitations from all the neighbors, based on the application's characteristics. When the participants are elected and the list created, a message is sent to present the application model and start a timer in which offers can be submitted. During this period of time, the participating nodes send offers for one or more tasks depending on their available resources. Once the time has expired, all the received offers are collected and the winners are announced. Finally, each winner receives the desired tasks.

A. Latency monitoring module

The primary function of the latency monitoring module is to filter the nearby IoT devices and create, at runtime, a list of participants that meet the IoT application requirements. Since each IoT application is different, the generated list of participants is unique. The difference comes from the filter threshold used by the module. In this case, the threshold is dependent on the application maximum e2e delay and has the purpose of ensuring that the communication latency from the dispatcher to the participants does not exceed a certain percentage. The equation used to compute this threshold is presented in Eq. 1:

$$T_{max} = e2e_{delay} \times p \tag{1}$$

where $e2e_{delay}$ is the maximum e2e delay of the application and p represents the percentage taken from the $e2e_{delay}$.

Furthermore, the threshold provides an upper bound of the accepted latency of each participant. Thus, the algorithm is in charge of monitoring the latency to the neighbor nodes, by measuring the time taken to send a message and until the time it arrived back, similar to [15]. Hence, this technique is perfect to monitor mobile devices and deal with their uncertainty. For example, if the latency of such a mobile device increases, then the mobile device has left the area. The entire functionality of this module is presented in Figure 3.

The second functionality of this approach is to keep track of the nodes that joined the network and provide a possibility to easily enter into the neighborhood of an edge device. Consequently, each node has the knowledge of all the computational resources in its vicinity. Since the module knows exactly how old (i.e., the period of time since the node joined the network) a node is, a better decision can be made when inviting participants. A list of participants is presented in Figure 1.

B. Offer generator module

This module is used only when a device is in a resource sharing state and has the purpose to create and send offers to the dispatcher. Each offer reflects the resources that a node desire to share and represents the Worst-Case Response

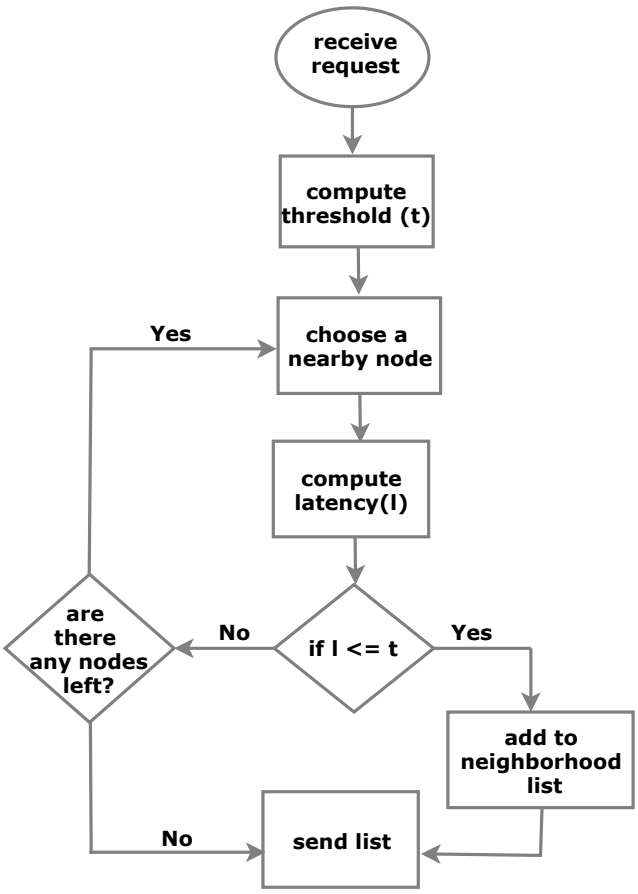

Fig. 3: Latency monitoring flowchart

Time (WCRT) of the selected tasks. Moreover, the offers are compared to a computed threshold to filter them and send only the ones that could improve the overall e2e delay. This strategy lowers the network communication and at the same time helps the neighbor node to take better decisions regarding its resources.

The threshold is calculated based on two parameters, the number of selected tasks and a percentage of the e2e delay. Through this approach, we ensure that only very good offers are sent to the dispatcher. The mathematical formula is presented in Eq. 2:

$$T_{max} = e2e_{delay} \times p \times n \tag{2}$$

where n represents the number of selected tasks and p represents the percentage taken from the $e2e_{delay}$.

The algorithm starts executing when an invitation message is received. At the same time, a counter, used to know how much time it takes to generate the current offer, is started by the algorithm. Next, based on the available resources, the module selects a number of tasks for which an offer is prepared. However, multiple tasks can be mapped on the same device only if they share a direct dependency. Therefore, the possible combinations of tasks are limited which makes the algorithm more efficient. Once the number of tasks is decided, a message containing an offer, the selected tasks, and a latency table is created. However, before sending the message to the dispatcher, the algorithm performs the last check, by comparing the counter with the duration received from the dispatcher. In this way, it verifies if the node still received offers or the predefined period ended. The overview of this module is presented in Figure 4.

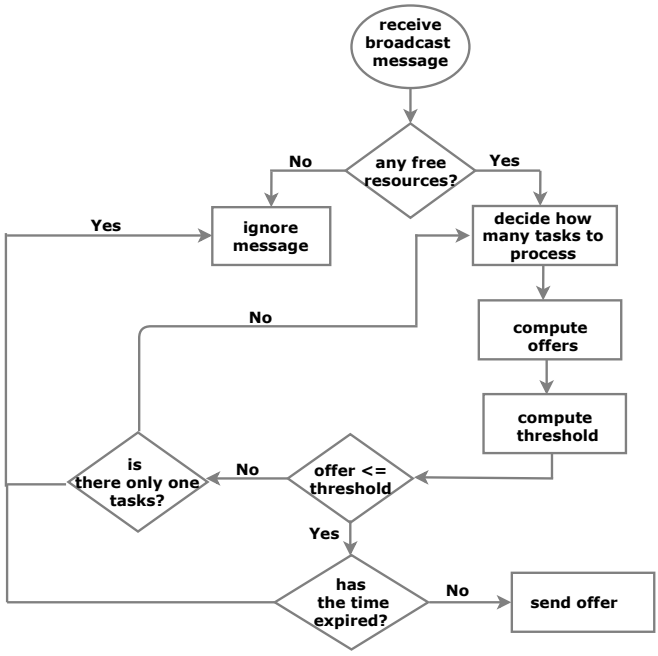

Fig. 4: Offer generator flowchart

Besides generating offers, this module creates a latency table, consisting of neighbor IoT devices and their respective latency (computed with respect to the current node). The devices considered in this table represents the results of an intersection between the received list of participants from the dispatcher and its own neighborhood. This approach aids the deployment strategy to find the e2e delay since it furnishes the latency communication between two different participants nodes (i.e., a knowledge that is not available from the latency monitoring module).

C. Deployment policy module

The deployment policy module is active only when a node is in the deployment state. To enter this state, a client request has to arrive to deploy an application. The main objective is to place the IoT application at the edge of the network such that the e2e delay is minimized. Once the deployment finishes, the module continues to monitor the winner devices to ensure a correct functionality. In case of failure, the module dynamically adapts by performing task migration. A task migration can only be executed by the dispatcher of that particular applications. Hence, if a node has to free some shared resources, the application's dispatcher is notified.

The flowchart of our deployment algorithm is outlined in Figure 5. First, the client sends an application to be deployed on an edge device. Once the application is received, the algorithm starts processing it. At the same time, a request for a list of devices, eligible to execute some of the tasks, is sent

to the latency monitoring module. While waiting for this list, the deployment module checks if there are any computational demanding tasks to be mapped to the cloud.

Next, a check of locally available resources is completed. Since the network communication latency impacts drastically the e2e delay, It is important to deploy the tasks on the smallest number of devices. Hence, as a rule, at least the first and last task (i.e., task A and task F in Figure 2) must be mapped locally because these tasks are connected directly to the IoT sensors and actuators, which are located nearer to the dispatcher. In the case where no resources are available, the module has to decide which of the current mapped tasks to migrate. This decision is made with the help of the task ranking system.

The task ranking system offers the possibility of task classification. The order is established based on a number of factors like, the communication latency to and from the current task, the resources used, how many tasks are grouped locally, and the trustiness of the associated dispatcher. The rank is computed, only once, when the edge device receives the tasks that it won.

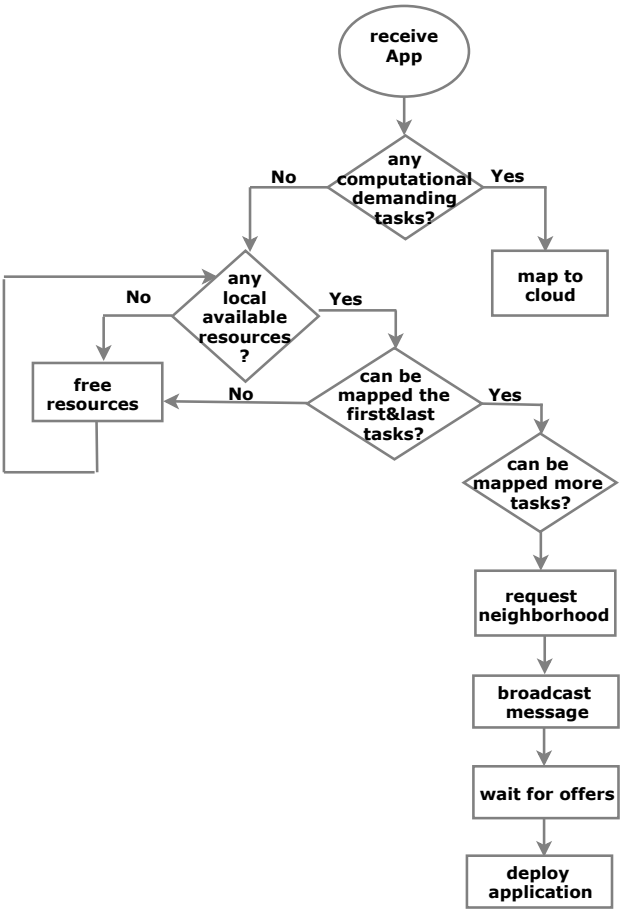

Fig. 5: Deployment policy flowchart

The algorithm creates a message once the tasks were locally mapped and the list of participants was retrieved from the latency module. The message includes the remaining tasks and their dependencies, the list of participants, the duration in which offers are received and the IoT application SLA. After the time expires, all received offers are divided into different lists. In each list, the algorithm places all the offers received for the same task/group of tasks. Based on all the offers, the deployment strategy is found by taking the minimum of each list and combining the results such that the e2e delay is minimized. The objective function, used to determine the best placement, is presented in Eq. 3:

$$App_{latency} = T_{map} + B_{latency} + WCRT \tag{3}$$

where T_{map} represents the time to find the best mapping for the IoT application, $B_{latency}$ is the sum of the communication latency between the winner nodes and $WCRT$ represents the sum of the received offers. It is imperative to state that T_{map} impacts the e2e delay only once during the initial deployment. In any other case, the T_{map} is equal to 0.

Since the offers are filtered twice: *firstly*: by the latency monitoring module when the list of participants nodes are created; *secondly* by the offer generator module when the offer is compared with the threshold, almost any combination of the nodes that offered the minimum values yields an e2e delay that meets the SLA. In doing so, the search space of this module is minimized resulting in a very efficient deployment algorithm. Moreover, knowing the communication latency between two nodes is not difficult anymore, since the module can easily find it from the offer messages.

Finally, to deploy the application a message is created carrying the docker containers of each task and the addresses of their dependent nodes, providing a direct communication link between the winner nodes.

V. Discussion

In this section, we present multiple challenges that must be investigated when developing algorithms for resource management at the edge of the network. These challenges target critical aspect of the algorithm such as network overhead and migration rules. As a result, these will help us to determine the limitations of such a system, understand, and overcome them. Next, the challenges and their implications, which we must consider when designing such a system are discussed.

A. Communication network challenges

To truly enable the full computational resources available at the edge of the network, mobile devices (i.e., laptops, smartphones or cars) have to be allowed. Besides the legal and security aspects of sharing resources, a careful attention must be accorded to the unpredictability introduced into the system. In our paper, we present a module for monitoring the latency in the network, a solution that can detect when a mobile device is leaving the area. However, the network overhead, introduced when multiple devices are connected in the same neighborhood is considered an issue. Hence, multiple evaluations must be performed to be able to discover new techniques for monitoring the latency at the edge of the network.

Furthermore, as in the case of latency monitoring, when a deployment is performed by the algorithm, multiple dockers are sent through the network. Even though we use dockers,

deploying multiple applications at the same location introduces massive traffic. As a result, performing multiple evaluations to find this limitation of our algorithm is utmost important. With this knowledge, we can further extend and improve our solution even more.

B. Proposed solution challenges

Our proposed solution is divided, as described in section IV, into three main modules each having a set of challenges that must be overcome using multiple tests.

1) Latency monitoring module: Being able to construct a personalized list of resources for each IoT application provides better opportunities to obtain individual deployment strategies. For example, if an IoT application has a smaller SLA, the algorithm considers only IoT devices that are in close proximity to the location of the dispatcher. In contrast, for a bigger SLA, more devices must be considered for sharing the resources. Therefore, it is important to define parameters that describe each IoT application in a novel way. However, besides SLA, other parameters must be investigated to determine if they influence the overall efficiency of an IoT application.

Another important parameter of this module is the threshold. It impacts directly the computational time and the results obtained by the deployment policy module. Comprehending the optimal value of the percentage used to compute the threshold, not only yields a better list of participants but aids the deployment algorithm in finding the best mapping faster and more efficient. However, to find the near-optimal percentage we must perform multiple evaluations. Such tests help us to understand better the impact of this percentage in the overall deployment strategy.

2) Offer generator module: Similarly to latency module, we must find the optimal percentage when computing the threshold. The threshold has a significant meaning in the overall functionality of our algorithm by determining if the offers are sent or not. Another possible problem that we must avoid is related to the migration of tasks. When performing tasks migration is crucial to guarantee a good decision-making algorithm. The algorithm ensures that tasks are migrated only when it is absolutely necessary. Therefore, we have introduced the ranking system notion in deciding what tasks to migrate although we still have to evaluate and ensure that the migration is not happening all the time. Since the decision can dramatically influence the latency of the application, performing constant migrations of tasks introduces huge penalties in the overall e2e delay.

3) Deployment policy module: In this case, determining the time needed to find a good mapping solution counts the most. Since this impacts the first e2e delay, we have to perform multiple tests to identify the balance between finding the best solution but missing the first SLA or accepting the first found solution and meeting the requirements. Since almost all solutions can meet the requirements, we have to decide what is a good time to let the algorithm search. Once we understand better the behavior, maybe a solution is to have a different time, based on the characteristics of each IoT application.

Besides the individual challenges of each module, an interesting problem that we desire to explain is related to the actual resources management. Currently, once an IoT application is mapped to a location, the resources are used continuously. This method is not efficient when an application is running for a specific amount of time. For example, a specific IoT application may only run during the night, being idle in the rest of the day. Therefore, such practice introduces a big waste of resources throughout the period when the application is idle. Consequently, one solution is to reuse the resources, by deploying other IoT application during the daylight. The proposed behavior requires new developments, which consider the migration of idle applications to other locations (where their functionality is needed) and free the local resources. However, once the night has come, the application has to be redeployed on the same location. This approach implies to track the timestamp of the first location (i.e., from where the application was moved) and guaranteeing that the resources are free once the application returns. In conclusion, the aforementioned approach, in combination with the proposed resource management algorithm, allow to fully employ the resource at the edge.

VI. Conclusion and future work

In conclusion, the stringent requirements of IoT applications have created new deployment challenges. In an effort to solve these challenges, new paradigms have been introduced to add more computational resources at the edge of the network. However, to truly take advantage of these resources, a distributed resource management technique is required. In this paper, we introduce a distributed latency-aware resource provisioning algorithm with the purpose of deploying IoT applications such that their SLA is satisfied. Furthermore, the algorithm can adapt to individual needs and environments of different applications. We believe that by applying the proposed algorithm in real IoT infrastructures (e.g., smart city) more application can be deployed at the edge of the network.

For future work, we plan to evaluate the proposed system considering simulations to test on a very large network and small prototypes consisting of multiple nodes where we can deploy and test our algorithm.

Acknowledgment

The research leading to these results has received funding from the European Unions Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764785, FORA–Fog Computing for Robotics and Industrial Automation. This work also has been partially supported and funded by the Austrian Research Promotion Agency (FFG) via the Austrian Competence Center for Digital Production (CDP) under the contract number 854187.

References

[1] A. Y. et al., "Edge computing technologies for internet of things: a primer," *Digital Communications and Networks*, vol. 4, pp. 77–86, 2018.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan 2017.

[3] M. Gerla, E. K. Lee, G. Pau, and U. Lee, "Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds," in *2014 IEEE World Forum on Internet of Things (WF-IoT)*, March 2014, pp. 241–246.

[4] M. C. Domingo, "An overview of the internet of things for people with disabilities," *Journal of Network and Computer Applications*, vol. 35, no. 2, pp. 584 – 596, 2012, simulation and Testbeds. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804511002025

[5] O. Fratu, C. Pena, R. Craciunescu, and S. Halunga, "Fog computing system for monitoring mild dementia and copd patients - romanian case study," in *2015 12th International Conference on Telecommunication in Modern Satellite, Cable and Broadcasting Services (TELSIKS)*, Oct 2015, pp. 123–128.

[6] F. B. et al., "Fog computing and its role in the internet of things," *1st ACM Mobile Cloud Computing Workshop*, pp. 13–15, 2012.

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

[8] K. Toczé and S. Nadjm-Tehrani, "A taxonomy for management and optimization of multiple resources in edge computing," *CoRR*, vol. abs/1801.05610, 2018. [Online]. Available: http://arxiv.org/abs/1801.05610

[9] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, Oct 2016.

[10] M. M. Shurman and M. K. Aljarah, "Collaborative execution of distributed mobile and iot applications running at the edge," in *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, Nov 2017, pp. 1–5.

[11] R. Jain and S. Tata, "Cloud to edge: Distributed deployment of process-aware iot applications," in *2017 IEEE International Conference on Edge Computing (EDGE)*, June 2017, pp. 182–189.

[12] A. Kapsalis, P. Kasnesis, I. S. Venieris, D. I. Kaklamani, and C. Z. Patrikakis, "A cooperative fog approach for effective workload balancing," *IEEE Cloud Computing*, vol. 4, no. 2, pp. 36–45, March 2017.

[13] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, no. 4, pp. 427–443, Dec 2017. [Online]. Available: https://doi.org/10.1007/s11761-017-0219-8

[14] J. Plachy, Z. Becvar, and E. C. Strinati, "Dynamic resource allocation exploiting mobility prediction in mobile edge computing," in *2016 IEEE 27th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Sept 2016, pp. 1–6.

[15] T. Rausch, S. Nastic, and S. Dustdar, "Emma: Distributed qos-aware mqtt middleware for edge computing applications," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*, April 2018, pp. 191–197.

[16] H. Meng, K. Zheng, P. Chatzimisios, H. Zhao, and L. Ma, "A utility-based resource allocation scheme in cloud-assisted vehicular network architecture," in *2015 IEEE International Conference on Communication Workshop (ICCW)*, June 2015, pp. 1833–1838.

[17] K. Habak, E. W. Zegura, M. Ammar, and K. A. Harras, "Workload management for dynamic mobile device clusters in edge femtoclouds," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 6:1–6:14. [Online]. Available: http://doi.acm.org/10.1145/3132211.3134455

[18] M. Aazam and E. N. Huh, "Dynamic resource provisioning through fog micro datacenter," in *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, March 2015, pp. 105–110.

[19] S. Yi, Z. Hao, Q. Zhang, Q. Zhang, W. Shi, and Q. Li, "Lavea: Latency-aware video analytics on edge computing platform," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, June 2017, pp. 2573–2574.

[20] D. Merkel, "Docker: Lightweight linux containers for consistent development and deployment," *Linux J.*, vol. 2014, no. 239, Mar. 2014. [Online]. Available: http://dl.acm.org/citation.cfm?id=2600239.2600241