Load Aware Provisioning of IoT Services on Fog Computing Platform

Bruno Donassolo[†‡], Ilhem Fajjari[†], Arnaud Legrand[‡] and Panayotis Mertikopoulos[‡]
[†] Orange-Labs: 44 Avenue de la République, 92320 Châtillon, France
Emails: ilhem.fajjari@orange.com, bruno.donassolo@orange.com
[‡] Univ. Grenoble Alpes, CNRS, INRIA, LIG - Grenoble, France
Emails: arnaud.legrand@imag.fr, panayotis.mertikopoulos@imag.fr

***Abstract*—To support the drastically increasing traffic generated by devices at the edge of the network, 5G players are urged to rethink their infrastructure design. Unfortunately, conventional Cloud infrastructures struggle to adapt to the huge volume of traffic. In this context, Fog computing has been developed to bridge Cloud data centers and edge devices servicing a multitude of heterogeneous devices. These nearby nodes offer analytics and data storage capabilities increasing considerably the capacity of the infrastructure. However, provisioning IoT applications on such a heterogeneous infrastructure, while meeting their stringent requirements is extremely challenging. In this paper, we study the Fog service provisioning issue in a practical manner. In this regard, we propose a novel strategy, which we call `GO-FSP`. `GO-FSP` optimizes the placement of IoT application components while coping with their strict performance requirements. To do so, we first propose an Integer Linear Programming (ILP) formulation for the IoT application provisioning problem. The latter targets to minimize the deployment cost while ensuring a load balancing between heterogeneous devices. Then, a GRASP-based approach is proposed to achieve the aforementioned objectives. Finally, we make use of the `FITOR` orchestration system to evaluate the performance of our solution under real conditions. Obtained results show that our scheme outperforms the related strategies.**

Keywords: Fog Computing, IoT, application placement, service provisioning

I. Introduction

The impressive proliferation of IoT devices has led to the explosion of traffic demand. Specifically, recent statistics highlight that 10% of traffic will be generated by these edge devices by 2020. Unfortunately, despite its agility, Cloud infrastructures can no longer withstand this prolific growth of the generated traffic. Therefore, key 5G players make every effort to envision innovative architectures which alleviate the processing burden in Cloud infrastructures.

In this perspective, one of the key novel concepts is that of Fog computing. Driven by ever-growing traffic and more stringent requirements, this new paradigm bridges Cloud data centers and edge devices to offer efficient IoT services. To do so, Fog computing relies on geographically distributed heterogeneous devices such as routers, switches, base stations, servers, etc. The so-called Fog nodes perform nearby analytics and data storage. They can be either virtualized or physical, depending on the hardware characteristics. In this large spectrum of resources, we highlight a set of specialized Fog nodes which are resource constrained and placed even closer to the end IoT devices. These resources, referred as Mist [1] nodes, are less powerful than other Fog nodes. However, being deployed side-by-side with end devices, they can ensure even lower delays for latency-sensitive applications.

The heterogeneity and the distribution of Fog nodes raise new challenges in terms of resources and IoT application lifecycle management. Specifically, how to optimize the provisioning of IoT application modules on Fog nodes to compose an application workflow, while meeting non-functional requirements in terms of quality of service (QoS), performance and low latency. Indeed, selecting optimal Fog resources becomes increasingly challenging when considering the uncertainty of the underlying Fog environment. In this perspective, orchestration is the cornerstone of Fog systems that handles the lifecycle management of multi-component IoT applications. It is worth noting that an IoT application can be modelled as a collection of lightweight, inter-dependent application components referred as micro-services. This building block is responsible for the design, on-boarding and delivering of application components that, put together, implement an end-to-end IoT service. In this context, several orchestration systems such as Kubernetes[1] and Mesos [2] are put forward in the literature. However, the diversity of Fog nodes, the dynamic character of running applications and the uncertainty of the underlying environment make them, even mature, ill-suited.

In this paper, we study the problem of Fog service orchestration. Specifically, we address the provisioning of Fog-enabled IoT applications. To achieve our objective, we envision an orchestration system called `FITOR`, which builds a realistic Fog environment while offering efficient orchestration mechanisms. Our framework provides an effective representation of the Fog infrastructure in terms of topology, resources usage, circulating flows, network latency, bandwidth, etc. In this perspective, `FITOR` leverages this accurate view to implement powerful Fog service provisioning strategies while abstracting the heterogeneity of the underlying Fog infrastructure.

In addition, we propose an efficient Fog service provisioning strategy, called `GO-FSP` which harnesses the flexibility of `FITOR` to optimize the placement of the IoT application components while considering their requirements in terms of resources usage and QoS. To achieve our objective, we formulate Fog service provisioning as an Integer Linear Problem (ILP). Since the problem has been proved NP-complete [2] and the ILP does not scale for medium and large scale instances of the problem, we put forward, `GO-FSP`, a GRASP-based [3] (Greedy Randomized Adaptive Search Procedures) approach to address scalability issues. `GO-FSP` bears two optimization objectives to fulfil QoS requirements of IoT applications. Indeed, it places IoT application components while jointly minimizing the provisioning cost and ensuring a satisfactory load share between Fog nodes. Our approach rapidly converges to an optimized solution while handling scalability constraints. Based on extensive experimentations, we assess the performance of our proposal compared to the most relevant related strategies. The results obtained highlight the strength of `GO-FSP` in terms of i) applications throughput, ii) end-to-end latency, iii) resource usage and iv) cost.

The remainder of this paper is organized as follows. In Section II, we describe the related work dealing with IoT service provisioning problematic. In Section III we give an overview of the `FITOR` architecture. In Section IV, we formulate the IoT service provisioning problem. Then, our proposal `GO-FSP` is described in Section V. The performance evaluation based on extensive experimentations is detailed in Section VI.

II. Related Work

Although Fog computing paradigm has drawn significant research interest, the orchestration of Fog services remains an

[1]Kubernetes. Available at: https://kubernetes.io/.
[2]Mesos. Available at: http://mesos.apache.org/.

978-1-5386-8088-9/19/$31.00 ©2019 IEEE

open issue. Only rare research work has been carried out to deal with this problematic in a practical manner. It is straightforward to see that, in this context, additional constraints need to be considered. In fact, addressing jointly the unique properties of IoT applications and the uncertainty of the Fog environment is extremely challenging. Hereafter, we summarize the most relevant related work that helped us to have an insight into the orchestration problematic in Fog infrastructures.

In [4], the authors propose a Fog orchestrator based on a Planning-Execution-Optimization control loop. Their solution makes use of a genetic-based algorithm to provision applications while enhancing the quality of service (QoS). In [5], the authors propound Enorm, a framework for Fog computing. Enorm is a 3-tier architecture composed of Cloud, edge and end devices. Such an environment is divided into regions and managed by the Cloud. In this context, the tasks are offloaded to the edge nodes when necessary. In [6], the authors adopt a similar approach which decomposes the environment into geographic regions hosting IoT applications. The latter are controlled by an SDN controller which is responsible for the orchestration of IoT applications while collecting infrastructure metrics. Inspired by ETSI NFV MANO reference architecture, the authors in [7] put forward a service orchestration architecture for Fog computing. The architecture relies on two main components: i) a Fog Orchestration Agent which manages the containerized services running locally in the Fog node, and ii) a centralized Fog orchestrator which manages the Fog nodes. Moreover, a relevant work in [8] makes use of Cloud Foundry PaaS to support the orchestration of IoT applications. In this context, some extensions are proposed. They aim at i) developing and deploying the application components, ii) controlling the application during its execution and iii) ensuring a given QoS level.

Regarding the programming models, [9] propounds a Foglet programming infrastructure for Fog environments. Together with the Foglet application, the developer can describe a set of characteristics, such as location, capacity and QoS, that will be used by the runtime system to manage the application execution. In [10], the authors propose a distributed dataflow programming model to describe the applications as directed graphs. Application components can run either in the Cloud or in the edge based on their Perception-action cycles.

We recall that in this paper, we study the problem of Fog service orchestration. At first sight, the latter seems to be similar to the provisioning problem of applications in Cloud infrastructures which has been deeply analyzed in the literature [11] [12]. However, although these studies can give insights into the subject, they cannot be directly applied in the Fog environment. This is mainly due to the noticeable differences between Cloud and Fog environments. In the former, the resources are homogeneous, powerful and centralized while in the latter, they are heterogeneous, constrained and geographically distributed. Notwithstanding, Fog applications raise new challenges related to their management due to their unique characteristics such as location-awareness, critical delay, mobility, volatility and distributed resources.

With regard to the provisioning of IoT applications in Fog Computing, some research work has been proposed. In [13], the authors deal with the Fog service placement problem (FSPP) while using Fog colonies to place IoT applications. FSPP models the problem as an Integer Linear Programming (ILP), whose objective consists in maximizing the Fog utilization while taking into account all constraints specified by the user. Both the infrastructure and the applications considered are described in terms of CPU, RAM, storage and network delay between elements. This work is later extended in [14], where a genetic algorithm is proposed to solve the optimization problem. In [15], the provisioning problem is also modelled as an ILP. However, the objective function consists in minimizing the cost. The model accepts as input a service level agreement (SLA) and a monetary penalty if the SLA is not fulfilled. To solve the problem, two heuristic-based approaches are proposed. The first strategy minimizes the SLA violation while the second minimizes the cost. The authors in [16] consider a different optimization objective function which aims to minimize application's average response time. In [17], the authors propound a heuristic that aims to maximize the number of satisfied IoT requests while respecting targeted QoS levels. In order to maximize the number of accepted requests, the proposed greedy algorithm sorts the demands in an ascending order according to the scarcest resources. Finally, in [18], the MCAPP (Multi-component Application Placement Problem) is formulated as a MILP (Mixed Integer Linear Program) problem whose objective is to minimize the total execution cost of an application. The MCAPP problem is then solved by MCAPP-Iterative Matching based heuristic.

In our work, similarly to [5] [8], we address the Fog service orchestration in a practical manner. Contrary to [4] [6], we implement an experimental platform of the proposed `FITOR` architecture to prove the feasibility of our solution. Unlike [5], we aim to provision multi-component IoT applications in a distributed Fog infrastructure. Moreover, similarly to [10], we describe IoT applications using dataflow programming model. However, we take into consideration QoS related requirements of applications during the provisioning such as in [9]. It worth noting that both [9] and [10] don't address the optimization of IoT application provisioning.
The proposed approaches in [13]–[19] deal with Fog service provisioning while considering one single objective function. This reinforces the analysis provided in [20], which states that only few research papers consider the multi-objective optimization problem in Fog computing. In this context, our work aims to solve the Fog service provisioning problem while taking into consideration two conflicting objectives: minimize the provisioning cost and increase the acceptance rate. To achieve our objective, we design and implement an efficient provisioning approach which iteratively optimizes the provisioning cost of allocating IoT application components while keeping a fair load-balancing of Fog resources.

III. Fog-IoT Orchestrator Architecture

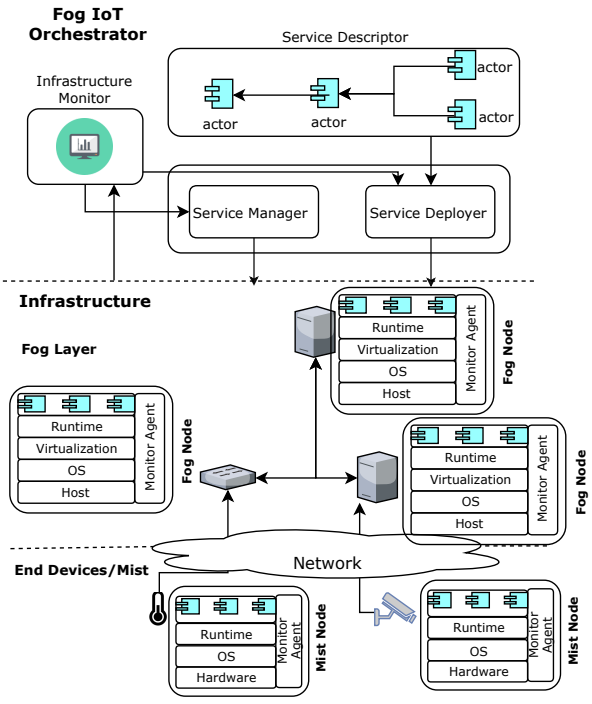


Fig. 1: Fog-IoT Orchestrator Architecture

Fig. 1 gives insights into our proposed FITOR architecture designed to involve mainly two main parts: i) Fog Infrastructure and ii) Fog IoT Orchestrator. Together, they handle the lifecycle management of Fog-enabled IoT applications. In this context, an IoT application is composed of a set of micro-services, each running in a container. The micro-service is implemented according to two main models: actor-based and dataflow. Hence, an actor is characterized by a private state and communicates thanks to the exchange of tokens between well-defined ports.

The Fog Infrastructure is composed of heterogeneous equipment and the modules needed to provide the hardware abstraction. The Fog layer can be subdivided into two main sub-layers: i) High Fog layer, providing more powerful computational and storage resources and ii) Lightweight Fog Layer (Mist), more specialized and constrained resources that are located closer to the end devices. In the bottom of the architecture we have the end devices, including sensors and actuators that interact with the environment. The heterogeneity of physical devices is handled by the Fog Node. The Fog Nodes can be hosted by a large variety of devices, such as servers, routers, switches or even and end device. The virtualization is carried out using containers (e.g. docker) that run the micro-services. On top of the virtualization layer, the runtime is responsible for the execution of actors. A runtime controls the actor scheduling and the data transport abstraction. Besides, it is characterized by its capabilities (e.g. access to the temperature sensor) and its performance metrics (e.g. CPU/RAM usage). These metrics are then collected, aggregated and exported by the Monitor Agent.

The Fog IoT Orchestrator is responsible for the provisioning of IoT applications in the underlying Fog infrastructure. To do so, it handles a **Service Descriptor** to enable a dynamic composition of IoT application modules. This deployment template specifies the IoT application, its components and requirements. It is worth noting that the aforementioned components correspond to the actors and their connections, while requirements are specified to ensure a guaranteed QoS. Indeed, CPU/RAM affinity and capacity can be specified for the actors, while latency and bandwidth can be defined for the connections. **The Service Deployer** carries the provisioning problem out. To do so, it makes use of the service descriptor and the collected infrastructure related metrics, to find the best placement of the application components while meeting their correspondent requirements. Once the provisioning of the Fog service is performed, the **Service Manager** takes in charge the scaling of services, their updating and/or upgrading and their termination. To do so, it uses the monitored key performance indicators of the service made available by the **Infrastructure monitor**. The latter is responsible for sketching out the telemetry information related to the Fog infrastructure by collecting several metrics from Fog nodes and their links. Among the collected metrics, we can cite the CPU, RAM and disk usage for Fog nodes, bandwidth and latency for network links. Further details about FITOR are given in [19].

IV. Problem Formulation

In this section, we give insights into the Fog-enabled service provisioning problem. We first define both Fog infrastructure and application models and then, we detail our problem formulation.

A. Fog System Model

The Fog landscape $\mathcal{P}$ is modelled as an undirected graph denoted by $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_P)$, where $V_{\mathcal{P}}$ represents the set of Fog nodes belonging to the different layers and $E_{\mathcal{P}}$ represents the direct communications between them. It is worth noting that $V_{\mathcal{P}} = (F_{\mathcal{P}} \cup S_{\mathcal{P}})$, where $F_{\mathcal{P}}$ corresponds to the set of Fog/Mist nodes, and $S_{\mathcal{P}}$ includes the sensors and actuators in the end device layer. Each node $v \in V_{\mathcal{P}}$ is characterized by its i) available CPU capacity $W(v)$ in MIPS, ii) available memory $M(v)$ in MB, iii) geographic location $G(v)$ and its iii) category $K(v)$[3]. Specifically, two types of Fog resources are considered: edge sensor/actuator (end device) and Mist/Fog node. Besides, we introduce two cost parameters for each Fog node, v: a unit cost of processing $C_W(v)$ and a unit cost of memory $C_M(v)$. Similarly, each physical link $l \in E_{\mathcal{P}}$ is characterized by i) its residual bandwidth $B(l)$, ii) its communication delay $L(l)$ and iii) its communication cost per unit bandwidth $C_B(l)$.

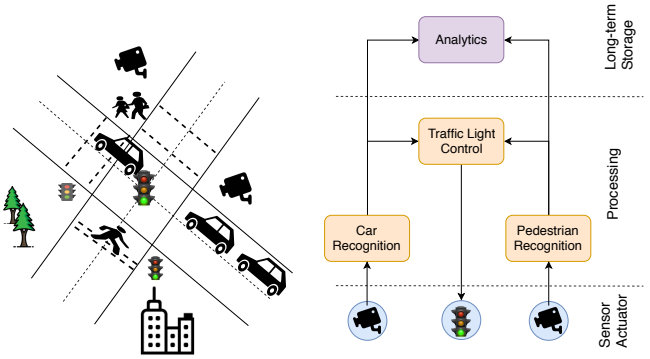


Fig. 2: Smart Traffic Light Application

Similarly, as depicted in Fig. 2, an application $\mathcal{A}$ is composed of a set of inter-dependent application modules (i.e., micro-services). Fig. 2 presents an example of an application with 3 levels: i) Data source that senses the environment and sends the collected data to be processed, ii) Application Logic which implements the business model of the application and may have the stringent requirements in terms of latency and processing, and iii) Storage which stocks the sensed data for further analysis. Formally, it is modelled as a directed acyclic graph (DAG), $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$, where $V_{\mathcal{A}}$ and $E_{\mathcal{A}}$ correspond to the services and their logical links, respectively. Specifically, each service $a \in V_{\mathcal{A}}$ requires an amount of processing power $W(a)$ and a memory $M(a)$, a geographic location $G(a)$ and a specific category $K(a)$ (e.g sensing, processing, storage, etc.). Following the dataflow model, $T_{rate}(a)$ is defined as the number of tokens per second sent by service a and $T_{size}(a)$ as the token size. Likewise, each link $e \in E_{\mathcal{A}}$ is characterized by a bandwidth demand $B(e)$ and a tolerable communication delay $L(e)$.

B. Fog Service Provisioning Problem

The objective of Fog-enabled IoT application provisioning problem consists in optimizing the placement of application components (i.e., micro-services) on distributed Fog nodes while meeting their non-functional requirements. Mist resources are constrained because of their inherent physical structure. Hence, we aim at favoring the usage of higher level Fog nodes since they offer better performance. In doing so, latency-sensitive applications are not obstructed and can be easily provisioned on Mist nodes when needed. However, deploying applications' components far from their data sources may deteriorate the network performance. Consequently, a trade-off between the provisioning cost and applications QoS fulfillment should be assured.

Hereafter, we formulate the provisioning problem of an IoT application $\mathcal{A}$, denoted by the graph $G_{\mathcal{A}}$, in the Fog infrastructure $\mathcal{P}$, modelled by the graph $G_{\mathcal{P}}$. To do so, we introduce:

- α_{av} is a binary variable indicating whether the service, $a \in V_{\mathcal{A}}$, is assigned to the physical node, $v \in V_{\mathcal{P}}$, or not.

[3]The category specifies the hardware properties of the Fog resource (e.g. storage node, network node, sensor, etc.)

- N_{vx} denotes the set of admissible physical paths from v to x, $(v, x) \in V_{\mathcal{P}}^2$.
- $\mathcal{N}$ denotes the set of all admissible paths. Formally, $\mathcal{N} = \bigcup_{\{v,x\} \in V_{\mathcal{P}}^2} N_{vx}$.
- β_{en} is a binary variable indicating whether a logical link $e \in E_{\mathcal{A}}$ is hosted in the physical path $n \in \mathcal{N}$.
- $(a_s(e), a_d(e)) \in V_{\mathcal{A}}^2$ denotes the starting (source) and terminating (destination) component of the logical link $e \in E_{\mathcal{A}}$.
- δ_{ln} is a binary coefficient determining whether the physical link $l \in E_{\mathcal{P}}$ belongs to the path $n \in \mathcal{N}$ or not.
- $B(e) = T_{rate}(a_s(e)) \times T_{size}(a_d(e))$ corresponds to the exchanged data rate in link e, from $a_s(e)$ to $a_d(e)$. We recall that T_{rate} is the number of tokens sent per second by $a_s(e)$ and T_{size} is the size of each token.

The provisioning of services is constrained so that for each IoT application $\mathcal{A}$, a service must be hosted in one Fog node. Formally,

$$\sum_{v \in V_{\mathcal{P}}} \alpha_{av} = 1, \quad \forall a \in V_{\mathcal{A}} \tag{IV.1}$$

A service $a \in V_{\mathcal{A}}$ can be hosted in the physical node $v \in V_{\mathcal{P}}$, if i) the available residual resources (i.e. $W(v)$ and $M(v)$) are at least equal to those required (i.e. $W(a)$, $M(a)$) and ii) a has the same category and geographical location as v. Formally,

$$\forall v \in V_{\mathcal{P}} \begin{cases} \sum_{a \in V_{\mathcal{A}}} W(a)\alpha_{av} & \leq W(v) \\ \sum_{a \in V_{\mathcal{A}}} M(a)\alpha_{av} & \leq M(v) \end{cases} \tag{IV.2}$$

$$(K(v) - K(a))\alpha_{av} = 0, \forall v \in V_{\mathcal{P}}, \forall a \in V_{\mathcal{A}} \tag{IV.3}$$

$$(G(v) - G(a))\alpha_{av} = 0, \forall v \in V_{\mathcal{P}}, \forall a \in V_{\mathcal{A}} \tag{IV.4}$$

We assume that a logical link $e \in E_{\mathcal{A}}$ between a service $a_s(e)$ and a service $a_d(e)$ is hosted in a path $n \in \mathcal{N}$ between v and x. Formally,

$$\sum_{n \in \mathcal{N}} \beta_{en} = 1, \quad \forall e \in E_{\mathcal{A}} \tag{IV.5}$$

A logical link $e \in E_{\mathcal{A}}$ must be instantiated in a single path $n \in N_{vx}$. Such as $a_s(e) \in V_{\mathcal{A}}$ is hosted in Fog node $v \in V_{\mathcal{P}}$ and $a_d(e) \in V_{\mathcal{A}}$ is hosted in physical node $x \in V_{\mathcal{P}}$. Formally,

$$\forall e \in E_{\mathcal{A}}, \quad n \in N_{vx} \begin{cases} \beta_{en} \leq \alpha_{a_s(e)v} \\ \beta_{en} \leq \alpha_{a_d(e)x} \end{cases} \tag{IV.6}$$

Each physical link $l \in E_{\mathcal{P}}$ is characterized by the consumed bandwidth $B_{used}(l)$ corresponding to the IoT application $\mathcal{A}$. Formally,

$$B_{used}(l) = \sum_{e \in E_{\mathcal{A}}} B(e) \sum_{n \in \mathcal{N}} \delta_{ln}\beta_{en}, \forall l \in E_{\mathcal{P}} \tag{IV.7}$$

Besides, each physical link $l \in E_{\mathcal{P}}$ cannot host more than its capacity. Formally,

$$B_{used}(l) \leq B(l), \forall l \in E_{\mathcal{P}} \tag{IV.8}$$

Each path $n \in \mathcal{N}$ is characterized by an end-to-end delay, $L(n)$. The latter corresponds to the sum of delays of its forming $l \in E_{\mathcal{P}}$. Formally,

$$L(n) = \sum_{l \in E_{\mathcal{P}}} \delta_{ln} L(l), \forall n \in \mathcal{N} \tag{IV.9}$$

Finally, a logical link $e \in E_{\mathcal{A}}$ must be hosted in a path n, ensuring an end-to-end delay lower than that required by itself.

$$\sum_{n \in \mathcal{N}} L(n)\beta_{en} \leq L(e), \quad \forall e \in E_{\mathcal{A}} \tag{IV.10}$$

Our decision is guided by two main objectives, and the solution is found by solving the problem as a multi-objective problem. **Firstly**, we aim to fulfil applications requirements while minimizing the cost of resources in $\mathcal{P}$. To do so, we define our first objective function as follows,

$$\min_{\forall v \in V_{\mathcal{P}}, \forall l \in E_{\mathcal{P}}} (C_W^{tot} + C_M^{tot} + C_B^{tot}) \tag{IV.11}$$

The C_W^{tot} corresponds to the processing cost of the application' components in the infrastructure. C_M^{tot} expresses the cost of the memory required by the components. Finally, C_B^{tot} represents the total communication cost for data transfer between applications components. We recall that we define the costs associated to the Fog infrastructure as i) a cost for processing $C_W(v)$, ii) a cost for memory $C_M(v)$, and iii) a cost for data transfer $C_B(l)$. Formally,

$$C_W^{tot} = \sum_{v \in V_{\mathcal{P}}} \sum_{a \in V_{\mathcal{A}}} C_W(v) W(a) \alpha_{av} \tag{IV.12}$$

$$C_M^{tot} = \sum_{v \in V_{\mathcal{P}}} \sum_{a \in V_{\mathcal{A}}} C_M(v) M(a) \alpha_{av} \tag{IV.13}$$

$$C_B^{tot} = \sum_{l \in E_{\mathcal{P}}} C_B(l) B_{used}(l) \tag{IV.14}$$

Secondly, we aim to maximize the number of provisioned IoT applications while meeting 100% of their demand. To to so, it is crucial to perform a load-aware application provisioning. The latter should balance the load of Fog nodes while taking into account their properties in terms of processing and memory usage. In this perspective, we formulate our second optimization objective function as follows:

$$\max \min_{v \in V_{\mathcal{P}}} \left(\frac{W(v)}{W_{max}(v)} + \frac{M(v)}{M_{max}(v)} \right), \tag{IV.15}$$

where W_{max} and $M_{max}(v)$ correspond to the CPU and memory capacity of node v, respectively. It straightforward to see that, Eq. IV.15 aims at maximizing the minimal residual resources of Fog nodes while considering their capabilities. In doing so, an equitable load sharing will be achieved.

V. Proposal: GO-FSP

The Fog-Enabled IoT application provisioning problem corresponds to an advanced formulation of composable service placement in computer networks problem. As the latter has been proved NP-complete [2], the proposed model in section IV can only be solved, in an efficient manner, for small size instances. It is straightforward to see that the Fog service provisioning problem is very hard to solve, due to scalability constraints. In fact, the dimension of the solution space would heavily increase following: i) the number of IoT applications and ii) the size of the Fog infrastructure. To get rid of this complexity challenge, we propose a new strategy which makes

use of Greedy Randomized Adaptive Search Procedures [3] to optimize the Fog service provisioning.

Our **GRASP-based Optimized Fog Service Provisioning** solution, called GO-FSP, initially generates N best initial solutions, $\{S_i\}_{1\leq i\leq N}$. Then, for each S_i, our strategy iteratively constructs new solutions by performing an efficient local search procedure. The key idea behind our strategy is to generate new solutions which simultaneously enhance the objective functions proposed in Eq. IV.15 and Eq. IV.11. The process will be repeated until all S_i are explored or no new improving solutions are found.

GO-FSP proceeds in three main stages: i) decomposition of the Fog service, ii) generation of initial solutions and iii) local search. Algorithm 1 illustrates the pseudo-code of our proposal GO-FSP. In the following, we will detail each stage.

Algorithm 1: GO-FSP pseudo-algorithm

```
Input: G_A, G_P, S_{1..n}, N, K, ε
Output: S_best
Function LocalSearch(C_a, S̃):
    RCL ← ∅
    for v ∈ V_P do
        if C(v,a) ≤ C(v*,a) × (1+ε) then
            RCL ← RCL ∪ v
    r ← FindBetterSolution (C_a, RCL)
    if r != S̃(a) then
        S̃(a) ← r
        return (true)
    return (false)
S_best ← ∅, Optimized ← true
for (i = 1 to N) and (Optimized = true) do
    Ã ← A
    Stop ← false, Optimize ← false
    for (j = 1 to K) and (Stop = false) do
        Stop ← true
        while Ã != ∅ do
            Select a ∈ V_Ã having the highest number
              of orphan links
            C_a ← a and all its in/out orphan links
            if LocalSearch (C_a, S_i) then
                Stop ← false
            Ã ← Ã / C_a
    if Improved(S_i, S_best) then
        S_best ← S_i
        Optmized = true
```

A. Fog Service Decomposition

The Fog service $\mathcal{A}$, is split up into a set of k components according to $|V_{\mathcal{A}}|$ and $|E_{\mathcal{A}}|$, known as solution components and denoted by $\{\mathcal{C}_i\}_{1\leq i\leq k}$. The Fog service decomposition is performed as follows. First, the sensor/actuator nodes and their outgoing/ingoing links are selected. Then, these solution components are retrieved from the graph. It is straightforward to see that the remaining $\mathcal{A}$'s topology, called $\tilde{\mathcal{A}}$, will contain several nodes bereft of their attached links. We refer to these links as "orphan links". $\mathcal{C}_i$ encompasses i) a central node a_i, and ii) its orphan attached links. The sequence of $\{\mathcal{C}_i\}_{1\leq i\leq k}$ is iteratively built in decreasing order of orphan links' number. To do so, the Fog service with the highest number of orphan links is selected. Then, $\mathcal{C}_i$ is constituted using only the Fog service and all its attached orphan links. Afterwards, $\mathcal{C}_i$ is subtracted from $\tilde{\mathcal{A}}$ (i.e., $\tilde{\mathcal{A}} \leftarrow \tilde{\mathcal{A}} \setminus \mathcal{C}_i$). The process is recursively repeated to generate the remaining solution components until $\tilde{\mathcal{A}}$ becomes empty (i.e., $\tilde{\mathcal{A}} = \emptyset$).

B. Generation of Initial Solutions

This stage consists in generating N provisioning solutions $\{S_i\}_{1\leq i\leq N}$. These S_i are built using our heuristic based provisioning approach, O-FSP [19]. The rational behind the aforementioned strategy is to greedily select N best solutions that minimize the provisioning cost. To do so, O-FSP incrementally constructs N optimized solutions while generating during each iteration, N best partial solutions within a fixed neighbourhood.

C. Local search

Starting from an initial S_i, GO-FSP, incrementally constructs an improved solution, $\tilde{S}_i$. To do so, it iteratively explores the ordered set of solution components that has been provided as the outcome of the decomposition stage. During each iteration, a Restricted Candidate List (RCL) is generated for the ongoing solution component, $\mathcal{C}_i$. RCL is composed of nodes within ε distance of the best ranked Fog node v^* in terms of $C(v^*, a_i)$. Note that $C(v, a)$ corresponds to the provisioning cost of v for hosting a. The aforementioned cost can be formulated as follows:

$$C(v,a) = (C_W(v) \times W(a) + C_M(v) \times M(a)) \qquad \text{(V.16)}$$

In this perspective, RCL can be defined as $\{v \in V_{\mathcal{P}} \mid C(v,a_i) \leq C(v^*,a_i) \times (1+\varepsilon)\}$. It is worth noting that the Fog nodes belonging to RCL are potential provisioning candidates since they fulfil the requirements of $\mathcal{C}_i$ in terms of CPU, memory, network latency and bandwidth.

Afterwards, thanks to FindBetterSolution, the least loaded node will be selected to host a_i. Formally,

$$\max_{v\in RCL}\left(\frac{W(v)}{W_{max}(v)} + \frac{M(v)}{M_{max}(v)}\right). \qquad \text{(V.17)}$$

In doing so, we minimize the variance of Fog nodes' load, to achieve an equitable load sharing among them, which amounts to optimize the second objective function formulated in IV.15.

VI. Performance Evaluation

In this section, we evaluate the performance of the proposed Fog service provisioning strategy GO-FSP by performing a series of detailed experimentations. Hereafter, we will describe the technical details of FITOR experimental platform and define the performance metrics considered to evaluate our strategy. Then, we analyze the results and discuss the effectiveness of our proposal GO-FSP compared with three main heuristics: i) O-FSP favouring nodes and links that minimize the total provisioning cost while respecting the requirements of the application, ii) Best-fit selecting nodes and links with the smallest residual resources, and iii) Uniform uniformly distributing service components $a \in V_{\mathcal{A}}$ in available nodes $v \in V_{\mathcal{P}}$ in the infrastructure.

A. Experimental environment

It is worth pointing out that GO-FSP is implemented as a provisioning strategy of the "Service Deployer" functional block. FITOR leans on **Grid5000** [21] and **FIT/IoT-LAB** [22] platforms to implement its Fog infrastructure. FIT/IoT-LAB is an open platform implemented to carry out IoT experiments which is composed of more than 2000 heterogeneous sensor nodes. Grid5000 focuses on parallel and distributed computing. It provides a high amount of powerful resources, grouped in homogeneous clusters. Additionally, FITOR employs a customized version of **Calvin** [23] to orchestrate the IoT services in the Fog. Calvin is a project led by Ericsson

which implements a framework for the development of IoT applications. The customized version includes extensions in Calvin's "Service Descriptor" to support dynamic requirements in terms of IT (CPU, RAM) and network (bandwidth, latency). This customized Calvin version enabled us to optimize the provisioning of Fog services while considering the stringent requirements of IoT applications. More technical details about FITOR are given in [19].

B. Environment setup

Our infrastructure $\mathcal{P}$ relies on elements from Grid5000 and FIT/IoT-LAB platforms. The Fog layer is composed of 20 servers from Grid5000 which are part of the *parapide* cluster. The latter are characterized by 2 CPUs Intel Xeon X5570, with 4 cores per CPU and 24 GB of RAM. Their CPU cost, C_W, and memory cost, C_M, are arbitrarily fixed to 0.1. Besides, the runtimes are hosted by Docker-based containers. Hence, to emulate a heterogeneous environment, we define four types of containers:

- Controller: runs the FITOR's node that is responsible for the deployment and control of IoT applications. This node is characterized by a memory capacity of 4GB and can use 100% of the host CPU.
- Fog100: is a more powerful Fog node with 2GB of RAM and 100% of available CPU.
- Fog60: is a middle Fog node with 2GB of RAM. The latter can use only 60% of the CPU.
- Fog30: is a limited capacity Fog node which can use only 30% CPU but has the same 2GB RAM size.

The Mist layer is composed of 50 A8 nodes. The A8 nodes are characterized by their ARM A8 microprocessor, 600 Mhz, and 256MB of RAM. These nodes run FITOR's processes and may execute application components. Besides, C_W and C_M are set to 0.9. The cost of links C_B is fixed to 0.1. This higher cost, compared to the Fog layer, points out the fact that these resources are less powerful, less available and closer to the end devices and so, must be cautiously used.

IoT applications are implemented as depicted in Fig 2. They employ three types of components: i) trigger service which periodically sends tokens of a fixed size, T_{size}, equals to 1024 byte at a given rate, T_{rate}, taking values in $[1, 10]$ tokens per second. These tokens represent the measurements collected from the end devices, ii) processing service which emulates the performed application treatment, as for example the car recognition process. So, it consumes a certain amount of million instructions (MI) per token, and iii) storage service which stores the received tokens in memory for analytics, if necessary.

We use a very simple scenario where each application arrives one after the other, every 2 minutes and is immediately placed by the orchestrator based on the application requirements and the current state of the platform. For each application, we set the number of components according to a discrete uniform distribution. In this respect, we fix the number of trigger services to 1, while we vary the number of processing services and storage services in $[1, 4]$ and $[1, 2]$, respectively. The trigger service requires a memory $M(a)$ equals to 100 bytes and an amount of CPU $W(a)$ proportional to its T_{rate}, in the range $[300, 1000]$ MIPS. For the processing service, we consider that the processing of one token varies in $[100, 1500]$ MI. Consequently, the CPU request is proportional to the processing effort per token, W_{token}, i.e., $W(a) = T_{rate} \times W_{token}$. On the other hand, $M(a)$ is fixed to 100Kb. Finally, we assume that the storage service requests a $W(a)$ equals to 500 MIPS and a $M(a)$ proportional to the token size and rate, such as $M(a) = T_{rate} \times T_{size}$.

Evaluation of this ramp-up for the four strategies are averaged over 10 different workloads and presented with 95% level confidence.

C. Performance metrics

To evaluate the performance of O-FSP compared with the classical approaches, we consider the following metrics:

- $\mathbb{A}$: is the acceptance rate of IoT applications.
- $\mathbb{T}$: is the total number of processed tokens per second within the infrastructure. This metric represents the applications' throughput.
- $\overline{\mathbb{L}}$: is the average latency (in seconds) for applications. It measures the latency between the end devices (represented by trigger services) and the top application layer (represented by storage services).
- $\overline{\mathbb{W}}$: is the average CPU utilization (expressed in percentage) of physical nodes in Grid5000 and FIT/IoT-LAB.
- $\mathbb{C}_{tot}$: is the total provisioning cost related to the consumed resources, as measured by the monitoring tools.

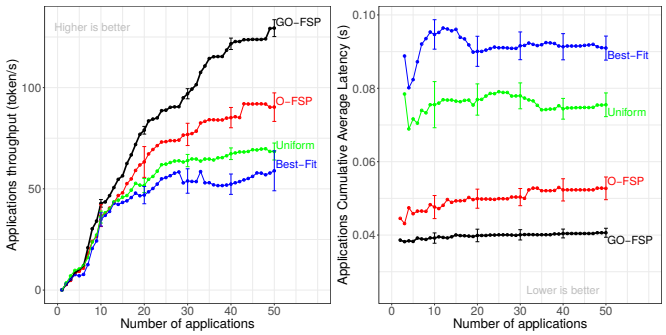
D. Evaluation results

We evaluate in Table I, the rate of accepted IoT applications. We notice that O-FSP ensures a high acceptance rate compared to Uniform and Best-fit. Indeed, GO-FSP accepts 10% more of the second strategy, Best-fit. This is due to the fact that our strategy aims at optimizing the load balancing between Fog nodes. In doing so, resources bottleneck is considerably relieved and consequently more applications can be executed on the infrastructure.

Provisioning Approach	Acceptance Rate (%)
GO-FSP	65.5 ± 0.8
O-FSP	$56.8.8 \pm 2.4$
Best-fit	58.3 ± 3.7
Uniform	55.1 ± 1.6

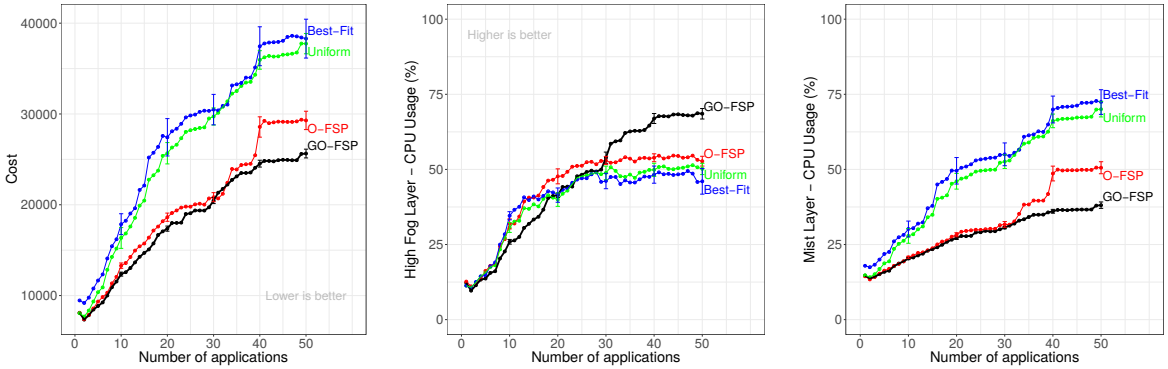
TABLE I: Acceptance rate - $\mathbb{A}$

Fig. 3 depicts the performance results in regard to the users' perspective. Specifically, the Fig. 3 (a) presents the rate of processed tokens while the number of provisioned applications increases. This metric expresses the applications' throughput by counting the number of tokens that are fully processed and delivered to the storage services. We note that GO-FSP achieves 30% higher throughput than other strategies thanks to its higher acceptance rate while meeting the applications' requirements. The performance gap gets wider when the number of applications increases. This is due to the fact that at the beginning, most resources are available and thus, all provisioning strategies are capable of placing the applications. In Fig 3 (b), we evaluate $\overline{\mathbb{L}}$, the network latency experimented by the applications. It is straightforward to see that GO-FSP achieves the lowest network delay. Indeed, by the end of experimentations, our proposal decreases the end-to-end latency of applications by respectively 18%, 48%, 59% compared to O-FSP, Uniform and Best-fit. Such an



(a) $\mathbb{T}$ - Applications throughput (b) $\overline{\mathbb{L}}$ - Applications average latency

Fig. 3: Application performance results



(a) $\mathbb{C}_{tot}$ - Total cost (b) $\overline{\mathbb{W}}$ - High Fog layer (c) $\overline{\mathbb{W}}$ - Mist layer

Fig. 4: Infrastructure performance results

achievement is ensured thanks to the capability of GO−FSP to aggregate application components while jointly optimizing the provisioning cost and load balance.

In order to gauge the efficiency of GO−FSP in terms of resources usage, we evaluate in Fig. 4, the infrastructure performances. Firstly, Fig. 4 (a) depicts the provisioning cost, $\mathbb{C}_{tot}$. We note that GO−FSP decreases $\mathbb{C}_{tot}$ by 15% compared to O−FSP strategy. This can be explained by the fact that our proposal favours cheaper resources and hence places application components in the High Fog Layer whenever it is pertinent. Secondly, the CPU utilization, for both High Fog and Mist layers, is evaluated in Fig. 4 (b) and (c). It is straightforward to see that GO−FSP optimizes the usage of the resources available in the High Fog Layer. The gap is approximately equal to 12.5% compared to the second best strategy, O−FSP. Such a result highlights the utility of the load balance which corroborates the results obtained in Fig. 3. Finally, we notice that the CPU utilization in the High Fog layer remains less than 75% due to the heterogeneity of applications profiles.

VII. Conclusion

In this paper, we addressed the provisioning problem of IoT applications in a Fog computing platform. To do so, we propose an load aware provisioning strategy named GO−FSP. This novel approach adopts the GRASP metodology to optimize the provisioning of Fog services while meeting their non-functional requirements. GO−FSP iteratively improves the initial solutions of the problem, considering a multi-objective criteria: provisioning cost and infrastructure load balance. By making use of the experimental environment available in FITOR, we conducted extensive experiments to measure the effectiveness of our proposal. The obtained results prove that GO−FSP achieves better performances compared with the related strategies in terms of: i) acceptance rate, ii) provisioning cost, iii) resource usage and iv) application latency.

Acknowledgments: Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.

References

[1] M. Iorga *et al.*, "Fog computing conceptual model," National Institute of Standards and Technology (NIST), Gaithersburg, MD, United States, Tech. Rep. SP 500-325, March 2018.

[2] X. Huang, S. Ganapathy, and T. Wolf, "Evaluating algorithms for composable service placement in computer networks," in *2009 IEEE International Conference on Communications*, June 2009, pp. 1–6.

[3] T. A. Feo and M. Resende, "Greedy randomized adaptive search procedures," vol. 6, pp. 109–133, 03 1995.

[4] Z. Wen *et al.*, "Fog orchestration for internet of things services," *IEEE Internet Computing*, vol. 21, no. 2, pp. 16–24, Mar 2017.

[5] N. Wang *et al.*, "Enorm: A framework for edge node resource management," *IEEE Transactions on Services Computing*, no. 99, 2017.

[6] S. Tomovic *et al.*, "Software-defined fog network architecture for iot," *Wirel. Pers. Commun.*, vol. 92, no. 1, pp. 181–196, Jan. 2017.

[7] M. S. de Brito *et al.*, "A service orchestration architecture for fog-enabled infrastructures," in *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, May 2017, pp. 127–132.

[8] S. Yangui *et al.*, "A platform as-a-service for hybrid cloud/fog environments," *2016 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)*, pp. 1–7, 2016.

[9] E. Saurez *et al.*, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, DEBS*, A. Gal *et al.*, Eds. ACM, 2016, pp. 258–269.

[10] N. K. Giang *et al.*, "Developing iot applications in the fog: A distributed dataflow approach," in *5th International Conference on the Internet of Things, IOT 2015, Seoul, South Korea, 26-28 October, 2015*. IEEE, 2015, pp. 155–162. [Online]. Available: http://dx.doi.org/10.1109/IOT.2015.7356560

[11] I. Fajjari, N. Aitsaadi, and G. Pujolle, "Cloud networking: An overview of virtual network embedding strategies," in *Global Information Infrastructure Symposium - GIIS 2013*, Oct 2013, pp. 1–7.

[12] X. Minxian, T. Wenhong, and B. Rajkumar, "A survey on load balancing algorithms for virtual machines placement in cloud computing," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12.

[13] O. Skarlat *et al.*, "Towards qos-aware fog service placement," in *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, May 2017, pp. 89–96.

[14] ——, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, pp. 427–443, Dec 2017.

[15] A. Yousefpour *et al.*, "Qos-aware dynamic fog service provisioning," *CoRR*, vol. abs/1802.00800, 2018.

[16] Y. Xia *et al.*, "Combining hardware nodes and software components ordering-based heuristics for optimizing the placement of distributed iot applications in the fog," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, ser. SAC '18. New York, NY, USA: ACM, 2018, pp. 751–760.

[17] H. Hong *et al.*, "Supporting internet-of-things analytics in a fog computing platform," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Dec 2017, pp. 138–145.

[18] T. Bahreini and D. Grosu, "Efficient placement of multi-component applications in edge computing systems," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, ser. SEC '17. New York, NY, USA: ACM, 2017, pp. 5:1–5:11.

[19] B. Donassolo *et al.*, "Fog Based Framework for IoT Service Provisioning," in *16th IEEE Annual Consumer Communications & Networking Conference, CCNC 2019, Las Vegas, NV, USA, January 11-14, 2019*, 2019, pp. 1–6.

[20] C. Mouradian *et al.*, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *IEEE Communications Surveys Tutorials*, vol. 20, no. 1, pp. 416–464, Firstquarter 2018.

[21] D. Balouek *et al.*, "Adding virtualization capabilities to the Grid'5000 testbed," in *Cloud Computing and Services Science*, ser. Communications in Computer and Information Science. Springer International Publishing, 2013, vol. 367, pp. 3–20.

[22] C. Adjih *et al.*, "Fit iot-lab: A large scale open experimental iot testbed," in *IEEE World Forum on Internet of Things*, Dec 2015, pp. 459–464.

[23] P. Persson and O. Angelsmark, "Calvin – merging cloud and iot," *Procedia Computer Science*, vol. 52, pp. 210 – 217, 2015, the 6th International Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015).