Fog Based Framework for IoT Service Provisioning

Bruno Donassolo, Ilhem Fajjari, Arnaud Legrand, Panayotis Mertikopoulos

Fog Based Framework for IoT Service Provisioning

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 this day, the Internet of Things (IoT) continues its explosive growth. Nevertheless, with the exceptional evolution of traffic demand, existing infrastructures are struggling to resist. In this context, Fog computing is shaping the future of IoT applications. It offers nearby computational, networking and storage resources to respond to the stringent requirements of these applications. However, despite its several advantages, Fog computing raises new challenges which slow its adoption down. Hence, there is a lack of practical solutions to enable the exploitation of this novel concept. To deal with this shortcoming, we propose `FITOR`, an orchestration system for IoT applications in the Fog environment. This solution builds a realistic Fog environment while offering efficient orchestration mechanisms. In order to optimize the provisioning of Fog-Enabled IoT applications, `FITOR` relies on `O-FSP`, an optimized fog service provisioning strategy which aims to minimize the provisioning cost of IoT applications, while meeting their requirements. Based on extensive experiments, the results obtained show that `O-FSP` optimizes the placement of IoT applications and outperforms the related strategies in terms of i) provisioning cost ii) resource usage and iii) acceptance rate.**

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

I. Introduction

The Internet of Things (IoT) is an emerging concept that promises to revolutionize our daily lives and the way we interact with our surrounding. It refers to the interconnection of heterogeneous end devices, especially everyday life objects, that are addressable and controllable via Internet. The number of such devices is expected to reach 75 billion(s) by 2025,[1] an explosion in size which will be inevitably the catalyst for the transformation of Cloud infrastructures. Being established too far away, traditional data centers struggle to meet stringent bandwidth and latency requirements of IoT systems. This is why the deployment of a new generation of infrastructure is crucial to deal with the huge amount of data transmitted by sensors and connected devices.

In this context, Fog Computing [1] is shaping the future of IoT solutions. To deal with the prolific growth of the generated traffic, the Cloud infrastructure is expanded to handle the processing in the surrounds of end devices. This augmented architecture provides nearby resources, performing analytics tasks and data storage. The so-called Fog nodes can be whether physical or virtualized. They are deployed between end devices and centralized services hosted by the Cloud. Note that a subset of Fog nodes are more specialized, dedicated and hence can be placed even closer to the end devices than the more powerful Fog nodes. They are called Mist nodes [2]. In doing so, a decentralized processing will be supported while taking advantage of the Cloud utilities and virtualization technology. It is straightforward to see that, the establishment of such a decentralized processing will ensure enhanced network performance, lower operational costs, alleviated network congestion and improved survivability.

However, such a design raises new challenges in terms of resource management and application orchestration. In this context, the orchestration is a keystone of Fog PaaS (Platform as a Service): It corresponds to designing, creating, and delivering one or more service components that, put together, offer an end-to-end service. Nevertheless, the heterogeneity, the dynamic nature and the large-scale deployment of the Fog environment make existing orchestration solutions, such as Kubernetes[2] and Docker Swarm[3], even mature, ill-adapted.

Recently, several researches [3] [4] [5] have proposed conceptual Fog frameworks for IoT service orchestration. However, most of them do not address the problematic in a practical and concrete manner. In this paper, we propose a Fog computing based framework, referred to as `FITOR`: **F**og-**I**o**T** **OR**chestrator which integrates end devices and Fog nodes in the Cloud ecosystem to create a Fog environment. `FITOR` provides various functionalities, which deal with the provisioning of IoT applications in a Fog infrastructure while meeting non-functional requirements in terms of network performances (e.g., latency, throughput) and IT resources (e.g., CPU, memory).

The contribution of this paper is twofold: first, we design and implement `FITOR`, an orchestration framework for the automation of the deployment, the scalability management, and migration of micro-service based IoT applications. Second, we propose a provisioning solution for IoT applications that optimizes the placement and the composition of IoT components, while dealing with the heterogeneity of the underlying Fog infrastructure.

In order to optimize the placement of micro-service based IoT components and their composition, we formulate the problem as an Integer Linear Problem. Since this problem is computationally intractable, the optimal solution could only be generated in small-sized instances of IoT applications and Fog infrastructures. We, therefore, propose an **O**ptimized **F**og **S**ervice **P**rovisioning strategy named `O-FSP` which adopts a divide and conquer approach. Its main objective is to minimize the cost of Fog-enabled IoT applications' provisioning while responding to their requirements in terms of resources and network performance. To do so, `O-FSP` exploits the gathered information about the current state of the sensors, actuators, Fog nodes, and the condition of IT and network resources to select the best placement of micro-services on Fog resources.

The performance of `O-FSP` is validated by robust experiments within the proposed `FITOR` platform and compared with several classical approaches: `Uniform`, `Best-Fit` and `Min-latency`. The results obtained highlight the strength of our strategy in terms of i) acceptance rate of IoT applications, ii) provisioning cost, and iii) CPU usage.

The remainder of this paper is organized as follows. In Section II, we will summarize the related work dealing with this problematic. In Section III we describe `FITOR`, a Fog based framework for IoT applications orchestration. In Section IV, we will formulate the IoT service provisioning problem. Then, our proposal `O-FSP` will be described in Section V. Performance evaluation based on experimentation will be detailed in Section VI.

II. Related Work

Several Fog architectures have been put forward in the literature. In this respect, in [3], the authors propose a prototype framework for Fog computing, named Enorm. The latter centralizes the management of applications and edge nodes in the Cloud, while offloading the workload to the edge nodes closer to the end user. In addition, an auto-scaling mechanism is implemented to scale up/down the resources

[1] Statista. Available at: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] Kubernetes. Available at: https://kubernetes.io/.

[3] Docker Swarm. Available at: https://docs.docker.com/engine/swarm/.

based on the network latency and the tasks' execution time. In [4], Inspired by ETSI NFV MANO reference architecture, the authors develop a service orchestration system for Fog computing. The architecture relies on two main components: i) a Fog Orchestration Agent (FOA) which runs in the Fog node and locally manages the services, and ii) a Fog orchestrator (FO) which makes use of a centralized view to manage the services and Fog nodes. In [6], a set of extensions to the Cloud Foundry architecture are proposed to cope with the Fog environment. These extensions include new modules to: i) develop applications, ii) deploy the components either in the Cloud or Fog using containers, and iii) manage the application during its execution. [7] proposes an SDN-based orchestration solution for Fog environments. To do so, the environment is split into several Fog regions managed by an SDN controller. The latter is responsible for keeping an updated view of the Fog nodes, collecting information about their capabilities (CPU, RAM, network, etc.), and orchestrating IoT applications while taking into consideration the business policies defined by applications. [8] proposes a Foglet programming infrastructure for Fog environments. Foglet provides a programming interface to be used by developers and a runtime system to manage the application execution.

Although the Fog computing concept has been relatively well investigated, the provisioning of IoT applications in the Fog environment remains an open issue. This is due to the distributed nature of Fog nodes and the uncertainty induced by multiple factors, such as end device volatility, peak resource demands and utilization gaps. It is worth noting that service provisioning in Cloud computing is mature [9] [10] and may offer interesting insights. However, there are key differences between Cloud services and Fog services that prevent the use of these methods in Fog context. Indeed, the type and size of Cloud resources are very different from Fog resources. In this respect, Cloud resources are usually homogeneous and centralized in data centers, while their counterparts in Fog computing are heterogeneous and distributed in a large area. In addition, Cloud applications are less challenging than IoT services running in the Fog. The latter are generally latency-sensitive. They require location-awareness, mobility support and a distributed coordination between geo-distributed end devices.

To the best of our knowledge, very few works have been carried out to specifically deal with the Fog service provisioning problem. In [11], the authors put forward the concept of Fog colonies to address the Fog service placement problem (FSPP), while taking into account Quality of Service constraints. In [12], two heuristics are proposed to efficiently deal with the FSPP while reducing the cost. In [13], the authors formulate an optimization problem to deal with the placement of IoT applications over Fog resources, while considering the heterogeneity of the resources and applications. Then, a greedy first fit heuristic and a genetic algorithm are proposed to solve the proposed optimization problem. In [14], the authors formulate the FSPP problem with an objective function that aims to minimize applications' average response time. The problem is addressed using two placement algorithms and two combinable heuristics to accelerate the placement decision making process.

A key underlying limitation of the aforementioned Fog service provisioning solutions [11]–[14], which serve as the main motivation of our work, is that none of them have been implemented nor do they tackle the issue of orchestrating heterogeneous Fog resources in a practical and concrete manner. Our work aims at complementing this, by offering a general centralized framework for holistic Fog resource orchestration. Our solution provides high level abstractions of the interactions between software components and end devices, which enable flexible deployment of management approaches. To gauge the efficiency of our FITOR architecture, we design and implement an efficient provisioning approach which exploits the gathered statistics, to optimize the allocation of IoT application components while considering their requirements.

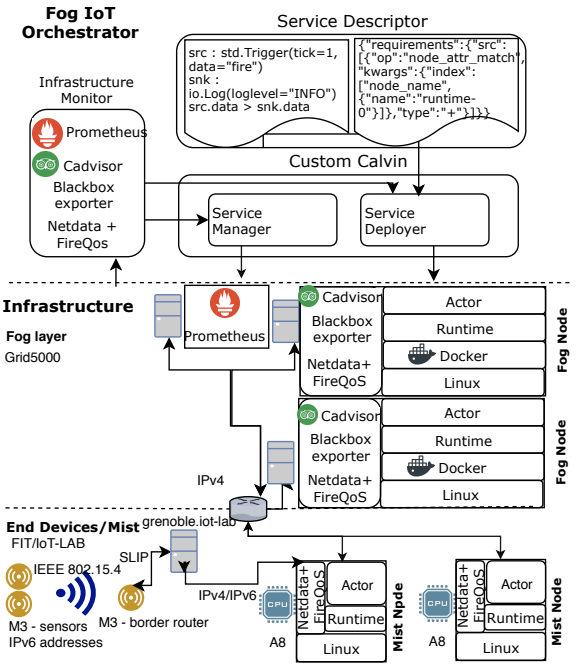


Fig. 1: Fog-IoT Orchestrator Architecture

III. Fog-IoT Orchestrator Architecture

Fig. 1 provides an overview of the FITOR architecture. Our proposed solution is device-aware, and hence, handles the heterogeneity of the Fog environment. Application components can be easily deployed on end devices and Fog nodes. It is worth noting that FITOR works across two main layers: 1) **Fog layer**: can be subdivided into two sublayers: i) High Fog layer composed of distributed Fog nodes providing nearby computational and storage resources and ii) Lightweight Fog Layer (Mist) encompassing more specialized, dedicated nodes that provide low computational resources. The so-called Mist nodes are placed closer to the end devices, and 2) **End devices**: includes the sensors, which collect information about the environment, and the actuators which have an effect on it.

In our context, an IoT application deployed in a Fog environment is composed of a set of micro-services which are containerized and running on the nodes of the infrastructure. We adopt an actor-based model to develop our IoT applications, where each micro-service is modelled as a set of actors and communicates, with other micro-services, through flows. An actor is characterized by a private internal state and a set of communication ports through which tokens are transmitted.

FITOR supports various orchestration mechanisms to deal with the stringent requirements of such applications. The global architecture for the proposed orchestration framework is depicted in Fig. 1 showing its components and their roles:

1) The Service Descriptor: This component aims to describe the IoT application, its building components and its requirements. Indeed, the developer needs to describe the actors, their requirements in terms of both location and computational effort, and how data should circulate between them. It is worth noting that, in order to ensure a guaranteed QoS, network related requirements could be specified during the description of links between the actors. Specifically, CPU/RAM affinity and capacity can be specified for the actors, while latency and bandwidth can be defined for the links.

2) The Service Deployer: Once the description is submitted, this component handles the mapping between the application components (i.e., actors, communications) and the nodes hosting the latter. Its main objective consists in optimizing the placement of actors and their links while considering their location, computational and network requirements.

3) The Service Manager: A running application is exposed to the uncertainty of the Fog environment. In addition, its components may change in response to the variation of data sources or internal transformation. To deal with this dynamicity, the applications containers are continually monitored in order to be managed while considering, in real-time, the environment evolution. To do so, scale out/in actions can be triggered to allocate or de-allocate resources. Besides, migration actions could be triggered when necessary.

4) The Infrastructure Monitor: This is responsible for sketching out the telemetry information by extracting several resource metrics from the Fog nodes and links. To do so, it makes use of various probes to get real-time information about both physical resources and their running containers. In our context, we consider two main categories of metrics to observe: i) Host-related metrics, concerning all information about the host and container in which the application is running. This includes: CPU, RAM, disk, etc. ii) Network related metrics, corresponding to the end-to-end latency and bandwidth, which are crucial and thus, mandatory to be collected during the application execution. It is straightforward to see that two views of resources are provided: i) reserved resources, which concerns the allocated capacities in terms of compute, storage and RAM, ii) the real resource capacities consumed by containers.

5) The Fog Node: This can be hosted by a server, a network equipment, or even an end device. High Fog nodes are capable of running containerized (e.g., docker) micro-services while Mist nodes are used as a bare-metal equipment. The so-called runtimes are responsible for the execution of actors. They may handle, for example, the actor scheduling and the data transport message parsing. It is worth noting that one runtime may concurrently run multiple actor instances belonging to different applications. A runtime is characterized by a set of capabilities (e.g., access to a camera, access to a disk, etc.) and performances metrics (e.g., CPU, memory) which are monitored by a set of tools. These tools collect, aggregate, process and export information about all components in the Fog node, including the physical host, running containers and network bandwidth and latency.

IV. Problem statement

A. Infrastructure and Application Models

We model our Fog infrastructure $\mathcal{P}$ as an undirected graph denoted by $G_{\mathcal{P}} = (V_{\mathcal{P}}, E_P)$, where $V_{\mathcal{P}}$ and $E_{\mathcal{P}}$ are respectively the sets of physical equipment and their connected links. Note that $V_{\mathcal{P}} = (F_{\mathcal{P}} \cup S_{\mathcal{P}})$, where $F_{\mathcal{P}}$ corresponds to the set of Fog nodes, while $S_{\mathcal{P}}$ encompasses the sensors and actuators in the end device layer.

Each physical node $v \in V_{\mathcal{P}}$ is characterized by its i) residual processing power $W(v)$, ii) residual memory $M(v)$, and iii) kind $K(v)$ and iv) geographic location $G(v)$. [4] We consider two types of resources: edge sensor/actuator (end device) and Mist/Fog node. Likewise, each physical link $l \in E_{\mathcal{P}}$ is characterized by its bandwidth capacity $B(l)$ and its latency $L(l)$.

Similarly, a Fog-Enabled IoT application $\mathcal{A}$ consists of multiple interconnected components (i.e., micro-services). It is modelled as a directed acyclic graph (DAG) $G_{\mathcal{A}} = (V_{\mathcal{A}}, E_{\mathcal{A}})$, where $V_{\mathcal{A}}$ and $E_{\mathcal{A}}$ are respectively the sets of services and their logical links. Within the IoT application $\mathcal{A}$, each service $a \in V_{\mathcal{A}}$ is associated with the required processing power $W(a)$ and memory $M(a)$, its kind $K(a)$, and geographic location $G(a)$. We also define $T_{rate}(a)$ and $T_{size}(a)$ as the number of tokens per second sent by service a and its token size, respectively. We recall that the kind specifies the requirement of a service to be executed on a specific category of resource. Finally, each link $e \in E_{\mathcal{A}}$ is characterized by a data rate $B(e)$, in terms of bandwidth, and $L(e)$, in terms of latency.

[4] It is worth noting that kind $K(v)$ specifies the hardware properties of the physical resource.

B. Problem formulation

In this section, we formulate the provisioning problem of a Fog-enabled IoT application $\mathcal{A}$, modelled by the graph $G_{\mathcal{A}}$, in the Fog infrastructure $\mathcal{P}$, denoted by the graph $G_{\mathcal{P}}$. First, we take into consideration the fact that all the physical nodes and link capacities (i.e. processing, memory and bandwidth) in $\mathcal{P}$ are limited. In fact, $\mathcal{P}$ is able to host a limited number of $\mathcal{A}$ while responding to the requested QoS. Consequently, an optimized $G_{\mathcal{A}}$ provisioning in $G_{\mathcal{P}}$ is crucial in order to maximize the acceptance rate while minimizing the cost of resources.

In what follows, we will use the following notation:

- 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.
- N_{vx} denotes the set of admissible physical paths from v to x, $(v, x) \in V_{\mathcal{P}}^2$. Note that k^2 denotes the family of all ordered pairs of elements within set k (i.e. $k = \{(v, x) \in V_{\mathcal{P}}^2 : v \neq x\}$).
- $\mathcal{N}$ 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 physical 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 kind 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 embedded in a physical 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 embedded in a single path $n \in N_{vx}$. Such as $a_s(e) \in V_{\mathcal{A}}$ is hosted in physical 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 used bandwidth $B_{used}(l)$ by the IoT application $\mathcal{A}$. Formally,

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

Moreover, 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 physical 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 be hosted in a path n, ensuring an end-to-end delay lower than that required by itself.

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

Our objective is to generate, for each application $\mathcal{A}$, the best possible provisioning solution while minimizing the placement cost in terms of allocated resources within $\mathcal{P}$. For this reason, we define our 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} represents the processing cost of the application' components in the infrastructure. C_M^{tot} is related to the cost of the memory required by the components. Finally, C_B^{tot} corresponds the total communication cost for data transfer between applications components. Also, we define the costs associated to the physical 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,

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

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

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

V. Proposal: O-FSP

In this section, we will detail our proposal named **Optimized Fog Service Provisioning** (O-FSP) to resolve the formulated problem in the previous section. Our strategy is a greedy approach which aims to incrementally construct an optimized Fog service provisioning solution. To achieve its objective, O-FSP proceeds as follows. First, the problem is split into a set of solution components that are sorted: solving a solution component corresponds to building a small part of the final solution. Then, each solution component is greedily placed while considering its requirements. Finally, the process is repeated until all solution components are provisioned. It is worth noting that O-FSP rejects a Fog service $\mathcal{A}$ as soon as it fails to find a placement to one of its component. O-FSP is summarized in pseudo-code form in algorithm 1. In the following, we will detail each stage.

A. Fog service decomposition stage

The Fog-enabled IoT application (i.e., Fog service) $\mathcal{A}$ is subdivided into a set of k components according to $|V_{\mathcal{A}}|$ and $|E_{\mathcal{A}}|$. The aforementioned subsets are called solution components and are denoted by $\{\mathcal{C}_i\}_{1 \leq i \leq k}$. In order to do this, O-FSP selects first the subset of sensor/actuator nodes and their corresponding incoming/outgoing links. The latter are, then retrieved from the graph. The remaining $\mathcal{A}$'s topology, devoid of the aforementioned solution components, is called

Algorithm 1: O-FSP pseudo-algorithm

```
Input: G_A, G_P, N_max
Output: S_best
S_best ← ∅
{S_i}_{1≤i≤N_max} ← ∅
{C_i}_{1≤i≤k} ← FogServiceDecomposition (G_A)
Ã ← A
/* Find N_max placements of end devices */
for (a ∈ V_A and K(a) = 0) do
    {C_a^i}_{1≤i≤N_max} ← FindProvisioning (C_a)
    for i = 1 to N_max do
        S_i ← S_i ∪ {C_a^i}
    Ã ← Ã / C_a
Stop ← false
while (Ã != ∅) and (Stop = false) do
    Select a ∈ V_Ã having the highest number of
     orphan links
    C_a ← a and all its in/out orphan links
    {C_a^i}_{1≤i≤N_max} ← FindProvisioning (C_a)
    if ({C_a^i} != ∅) then
        for i = 1 to N_max do
            S_i ← S_i ∪ {C_a^i}
        Ã ← Ã / C_a
    else
        S_best ← ∅
        Stop ← true
S_best ← SelectBestSolution ({S_i}_{1≤i≤N_max})
```

$\tilde{\mathcal{A}}$. It is straightforward to see that $\tilde{\mathcal{A}}$ may hold several nodes lacking their incoming links. We refer to these links as orphan links. $\mathcal{C}_i$ encompasses i) a central node v_i, and ii) its orphan incoming/outgoing links. $\{\mathcal{C}_i\}_{1 \leq i \leq k}$ is iteratively built on the basis of the number of their orphan links. Indeed, the nodes with the highest number of orphans links are selected first. The process is repeated until $\tilde{\mathcal{A}}$ becomes empty (i.e., $\tilde{\mathcal{A}} = \emptyset$).

B. Solution component's provisioning stage

During the first iteration, O-FSP finds the N_{max} best placements of end device solution components $\{\mathcal{C}_i\}_{1 \leq i \leq k_1} \mid k_1 \leq k$. Then, O-FSP incrementally constructs the Fog service provisioning solution. Hence, during each iteration, the N_{max} best (partial) placements, $\{S_i\}_{1 \leq i \leq N_{max}}$, maximizing the objective function defined by equation (IV.11) are generated (FindProvisioning in algorithm 1). To do so, N_{max} physical nodes are selected for hosting the considered $\mathcal{C}_i$. The latter has sufficient resources (i.e. CPU, memory), and it is connected to one $\{\mathcal{C}_j\}_{1 \leq j \leq i-1}$'s placement solution through a shortest path ensuring the required bandwidth and latency. We recall that N_{max} solutions $\{\mathcal{C}_j^i\}_{1 \leq i \leq N_{max}}$ have already been generated for the placement of $\mathcal{C}_j$ during the previous iterations. In doing so, we generate the best possible solution while avoiding the exploration of all possible combinations that would make the resolution impractical. Finally, the best solution S_{best}, which corresponds to the placement that maximizes the objective function, is selected (SelectBestSolution in algorithm 1).

VI. Performance Evaluation

In this section, we will assess the feasibility of the FITOR framework and evaluate the performance of the proposed optimized Fog Service Provisioning strategy O-FSP. To achieve this, we first describe the technical details of the FITOR prototype. We then define the performance metrics to evaluate our proposal, and finally, we analyze the results and discuss the

effectiveness of O−FSP compared with three main heuristics: i) Uniform which uniformly distributes service components $a \in V_{\mathcal{A}}$ in available node $v \in V_{\mathcal{P}}$ in the infrastructure, while respecting the requirements of the application, ii) Best−fit which favors the physical nodes and physical links with the smallest residual resources, and iii) Min−latency which minimizes the sum of delays between nodes hosting the applications components.

As depicted in Fig. 1, a prototype of FITOR is implemented based on a customized version of **Calvin** [15]. Calvin is a project led by Ericsson which proposes a framework for the development of IoT applications. First, we have extended Calvin's application descriptor to support dynamic metrics, such as available CPU and RAM, network latency and bandwidth. In doing so, we are able to consider the stringent requirements of IoT applications. Then, a custom implementation of "Service Deployer" was developed to optimize the placement of Fog service components while considering the specified requirements. To ensure a holistic monitoring, we make use of **Prometheus**[5] to collect metrics about our Fog platform behavior during tests. Prometheus relies on monitoring tools, such as netdata, cadvisor and blackbox exporter to generate the metrics.

To experiment our orchestration solution, we make use of both FIT/IoT-LAB [16] and Grid5000 [17] infrastructures. FIT/IoT-LAB is a large scale open platform to perform IoT experiments. It provides more than 2000 heterogeneous sensor nodes spread in 6 different sites in France. In contrast, Grid5000 focus is on parallel and distributed computing, such as Cloud and Big Data. Grid5000 contains a large amount of powerful resources, grouped in homogeneous clusters. Similar to the FIT/IoT-LAB platform, nodes are also spread in different locations in France. Both platforms are highly configurable and accessible remotely, leveraging experiments in different areas.

Our main objective is to set up a realistic Fog environment, enabling the deployment of IoT applications and hence, their orchestration. To do so, we use M3 and A8 boards to set up the end device layer and the Mist sublayer, encompassing the sensors and Mist nodes. Each M3 board is composed of an ARM micro-controller, 64KB of RAM, various sensors (e.g., ambient sensor light, atmospheric pressure and temperature, etc.) and an IEEE 802.15.4 radio interface. On the other hand, A8 nodes are more potent. Thanks to its ARM A8 microprocessor and 256MB of RAM, it is capable of running user's applications and hence, can be considered as a Mist node. We then make use of some Grid5000 servers to emulate more powerful Fog nodes. To connect both infrastructures, we create L3 VPNs between the A8 nodes and Grid5000. Also, we use public IPv6 addresses to connect the M3 sensors in the network, as detailed in the tutorials available at FIT-IoT platform. Each node of our Fog infrastructure hosts:

- **Docker** engine to cope with hardware and software heterogeneity. Docker containers are used to create an image which encapsulates the software necessary in our setup, namely Calvin, Blackbox exporter and Netdata.
- **Cadvisor** to ensure the monitoring of Docker containers performance. Some of the collected metrics are CPU and RAM utilization.
- **Blackbox exporter** to measure the latency between the node and all other nodes of the infrastructure if needed.
- **Netdata** and **FireQoS** to monitor the bandwidth utilization without the support of network equipment. In our setup, we use these tools to measure the bandwidth utilization of each Calvins flow. Note that the flow is characterized by a TCP connection between two nodes. Using FireQoS, we configure the Traffic Control module in the Linux kernel to identify each flow and to keep statistics about them. Then, Netdata collects the information and sends it to our infrastructure monitor.

[5]Prometheus. Available at: https://prometheus.io/.

A. Environment setup

Our Fog 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 *genepi* cluster. They are characterized by 2 CPUs Intel Xeon E5420, with 4 cores per CPU and 8 GB of RAM. Their CPU cost C_W and memory cost C_M are arbitrarily set to 0.1. It is worth noting that the runtimes are hosted by containers.

The Mist layer is composed of 50 A8 nodes. 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.

On the other hand, IoT applications are characterized by 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 transport the collected measurements related to the end devices' surrounding environment, ii) processing service which emulates the performed application treatment. It consumes a certain amount of million instructions (MI) per token, and iii) storage service which stores the received tokens in memory for further processing (if necessary).

The arrival rate of applications is fixed to 1 application every 2 minutes. 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 results are always presented with confidence intervals corresponding to a confidence level of 95%. Tiny confidence intervals are not shown in the following figures.

B. Performance metrics

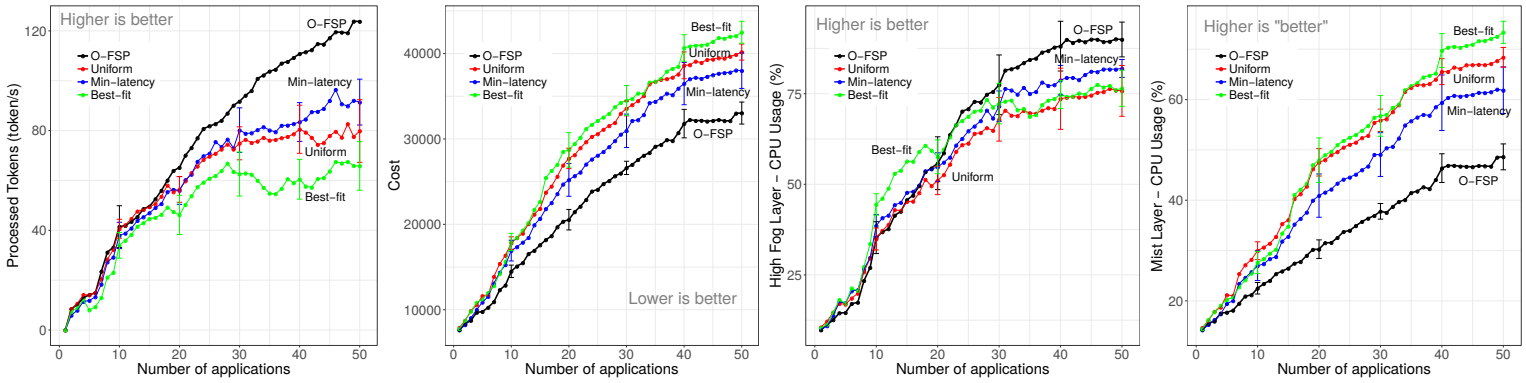
To evaluate the performance of O−FSP compared with the classical approaches, we consider the following metrics:

- $\mathbb{A}$: is the acceptance rate of IoT applications.
- $\mathbb{T}$: is the total number of processed tokens per second within the infrastructure.
- $\mathbb{W}$: 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.

C. Evaluation results

First, we evaluate the proposed strategy O−FSP regarding its acceptance rate and in comparison with the alternative approaches, Best−fit, Uniform and Min−latency. Table I compares the percentage of accepted IoT applications. We notice that O−FSP achieves 77.8% and hence, outperforms the classical strategies. This is due to the fact that the proposed approach aims to minimize the provisioning cost. In doing so, higher residual resources, specifically network bandwidth, are maintained. It is worth noting that Best−fit achieves good results since it favours the selection of less powerful nodes which will lead to a higher number of available powerful nodes.

Fig. 2 (a) depicts the rate of processed tokens when increasing the number of provisioned applications. It is straightforward to see that our scheme O−FSP outperforms the classical



(a) $\mathbb{T}$ - Processed tokens rate (b) $\mathbb{C}_{tot}$ - Total cost (c) $\overline{\mathbb{W}}$ - High fog layer (d) $\overline{\mathbb{W}}$ - Mist layer

Fig. 2: Performance results

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

Provisioning Approach	Acceptance Rate (%)
O-FSP	77.8 ± 5.5
Min-latency	65.6 ± 1.9
Best-fit	69.4 ± 3.4
Uniform	65.2 ± 2.9

strategies thanks to its capability to accept IoT applications while meeting their requirements. It is worth noting that the gap between the different approaches is insignificant for low number of provisioned applications. This is due to the fact that initially, all provisioning strategies are capable of placing the applications. However, the gap gets wider as soon as the number of applications increases. Such a result proves that `O-FSP` efficiently places the application components which corroborates the results related to the acceptance rate.

Fig. 2 (b) illustrates the provisioning cost of accepted IoT applications. It notably shows that `O-FSP` achieves a lower cost compared with the classical strategies and does so throughout the experiment. In fact, at the end of the experiment, the provisioning cost of `O-FSP` is $\approx 13\%$ lower than the one of the second strategy, `Min-latency` ($33,033 \pm 1,289$ vs $37,998 \pm 2,078$). Such results are predictable since the proposed approach favours high fog physical nodes as long as the application requirements are satisfied. It worth noting that end devices and mist nodes have limited capacity. Consequently, they incur a high cost compared with fog nodes which are characterized by higher capacities.

In order to gauge the efficiency of `O-FSP` in terms of resource consumption, we evaluate the CPU usage of both end device/mist and fog nodes. Fig. 2 (c) and Fig. 2 (d) prove that our proposed strategy favours the fog nodes whenever they meet the application requirements. In fact, it achieves 90% of CPU usage which is 10% higher than the second method `Min-latency`. In doing so, end devices and mist nodes are kept available and used only when it is necessary. We recall that the latter are less powerful and more expensive than Commercial-Off-The-Shelf (COTS) servers. It is worth noting that all strategies reach, at the end of the experiment, a high level of CPU usage on both end device/mist and fog layers. This is due to the system saturation.

VII. Conclusion

In this paper, we tackled the problem of Fog-Enabled IoT application orchestration. To do so, we proposed `FITOR`, a new orchestration solution for IoT applications in the Fog environment. To deal with the service provisioning issue, we put forward a novel strategy, `O-FSP`, which optimizes the placement of IoT application components while meeting their non-functional requirements. In order to gauge the effectiveness of our solution, we implemented `FITOR` and `O-FSP` which make use of an extended version of Calvin, Grid5000 and FIT/IoT-LAB platforms. Extensive experiments show that the `O-FSP` strategy makes the provisioning more effective and outperforms classical strategies in terms of: i) acceptance rate, ii) provisioning cost, and iii) resource usage.

References

[1] F. Bonomi *et al.*, "Fog computing and its role in the internet of things," in *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. New York, NY, USA: ACM, 2012, pp. 13–16.

[2] 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.

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

[4] 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.

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

[6] 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.

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

[8] 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, USA*, A. Gal *et al.*, Eds. ACM, 2016, pp. 258–269.

[9] 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.

[10] 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.

[11] 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.

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

[13] O. Skarlat *et al.*, "Optimized iot service placement in the fog," *Service Oriented Computing and Applications*, vol. 11, pp. 427–443, Dec 2017.

[14] 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.

[15] 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).

[16] 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.

[17] 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.