How to best deploy your Fog applications, probably*

Antonio Brogi, Stefano Forti, Ahmad Ibrahim
Department of Computer Science, *University of Pisa, Italy*
{brogi, stefano.forti, ahmad}@di.unipi.it

***Abstract*—Deploying composite applications to Fog nodes in a QoS- and context-aware manner is challenging due to the heterogeneity and scale of Fog infrastructures. Application components must be provided with the software and hardware capabilities they need. Communication links that support interactions between components must meet certain QoS (latency and bandwidth). On the other hand, different Fog and Cloud nodes provide different software and hardware capabilities, and actual communication links support different QoS over time.**

In this paper we present a prototype (FogTorchΠ) capable of determining deployments of composite applications to Fog infrastructures, which fulfil software, hardware and QoS requirements. FogTorchΠ exploits Monte Carlo simulations to take into account possible variations of the QoS of communication links. It classifies eligible deployments both in terms of QoS-assurance and of Fog resource consumption. We illustrate the utility of FogTorchΠ over a motivating example where we compare different possible deployments for a smart agriculture application.

I. Introduction

Due to the volume, variety and velocity of data generated by IoT sensors and actuators, the Cloud cannot fully support IoT applications that must meet compelling latency or bandwidth constraints [1]. Indeed, the considerable increase in the amount of data produced at the edge of the network was not accompanied by a comparable increase of available bandwidth from/to the Cloud [2]. Furthermore, Cloud connection latencies are not adequate to host real-time tasks such as life-saving connected devices, augmented reality, or gaming [3].

The need to provide processing power, storage and networking capabilities to run IoT applications closer to sensors and actuators has been highlighted by various authors, such as [4], [5]. Fog computing [6] aims at selectively pushing computation closer to where data is produced, by exploiting a geographically distributed multitude of heterogeneous devices (e.g., gateways, micro-datacentres, embedded servers, personal devices) at the edge of the Internet. On one hand, this permits low-latency responses to (and analysis of) sensed events, on the other hand it relaxes the need for (high) bandwidth availability from/to the Cloud. Fog computing is hence expected to fruitfully extend the Cloud+IoT scenario, enabling Quality-of-Service- (QoS) and context-aware application deployments [5].

While various solutions for applications deployment have been proposed for the Cloud [7], Fog computing calls for new methods to guarantee non-functional constraints of deployments spanning from Cloud datacentres to resource-constrained Fog devices. Indeed, contrarily to the Cloud, Fog nodes often feature limited resources, what makes single node deployments not always feasible [5]. Thus, tools for distributed deployment of components, spanning multiple nodes, have to ensure that time-sensitive or bandwidth greedy applications work properly, having their (non-functional) needs reliably fulfilled. The distribution of application components between the edge and the Cloud should take into account both processing (e.g., CPU, RAM, storage) and QoS (e.g., latency, bandwidth) constraints, and it should not be tuned manually [2]. Metrics to estimate and compare the *QoS-assurance* and the *consumption of Fog resources* of alternative eligible deployments [5] can ease the selection of deployments that are more likely to meet all processing and QoS constraints while minimising the consumption of Fog resources when needed.

In this paper we present FogTorchΠ, a prototype based on a model [8] of the Cloud+Fog+IoT scenario to support application deployment in the Fog. FogTorchΠ permits to express processing capabilities and average QoS attributes (i.e., latency and bandwidth) of a Fog infrastructure, along with processing and QoS requirements of an application, and it determines deployments of the application over the Fog infrastructure that meet all such requirements.

FogTorchΠ models the QoS of communication links by using probability distributions (based on historical data), describing variations in featured latency or bandwidth over time, depending on network conditions. For instance, wireless technologies perform worse in case of bad weather, whilst Internet access technologies suffer from QoS degradation (higher latency, lower bandwidth) when the network gets congested. To deal with input probability distributions and to estimate the *QoS-assurance* of different deployments, FogTorchΠ exploits the Monte Carlo method [9]. By repeatedly sampling probability distributions, FogTorchΠ simulates different runtime behaviours of communication links and aggregates results for deployments generated over a large number runs.

For each deployment FogTorchΠ outputs QoS assurance and resource consumption over Fog nodes. Those metrics permit to compare possible deployments, and to evaluate the impact of possible changes in the Fog infrastructure/applications (what-if analysis). As highlighted in [5], the availability of tools to simulate Cloud+Fog+IoT scenarios, without implementing testbeds, is still too scarce. As we will show, tools like Fog-

*Accepted in 1st IEEE International Conference on Fog and Edge Computing (ICFEC 2017), May 14th 2017, Madrid, Spain

TorchΠ may help IT experts in deciding where to deploy each component of an application to get the best QoS assurance, as well as in identifying beforehand existing infrastructure criticalities.

The rest of this paper is organised as follows. Section II describes a motivating example from smart agriculture, which illustrates the complexity of deploying IoT applications in the Fog. Section III reviews the formal model we proposed for Fog infrastructure and applications, and presents FogTorchΠ. Section IV shows and discusses the results of applying FogTorchΠ to the motivating example of Section II. Related work and concluding remarks are in Sections V and VI, respectively.

II. Motivating Example

Consider a smart agriculture application that enables remote monitoring and irrigation of crops. The application consists of three components (Fig. 1):

- `ThingsController` interacts with IoT sensors and actuators[1]. The Things exploited by this application are a moisture sensor (checking feasible conditions for irrigation), a fire sensor (detecting fire hazards), an electronic water valve (irrigating the crops), and a video camera (streaming live footage from the fields).
- `DataStorage` stores all the information collected from the Things for future use. If streaming from video camera is enabled, a microservice (disabled by default) is used to manage the communication with the camera. Furthermore, an embedded machine learning engine periodically communicates with the `ThingsController` component to optimise crop irrigation rules.
- `Dashboard` permits to visualise, monitor and control data from the sensors, along with historical data and machine learning engine rules.

Each component has certain software and hardware requirements to function properly. Software requirements include needed operating system, programming languages support, RDBMS, etc. Hardware requirements include number of CPU cores, amount of RAM and storage each component needs [10]. Hardware requirements are classified as consumable (e.g., RAM or storage) and non-consumable (e.g., CPU cores). Available consumable resources decrease as components are deployed and exploit them.

Software components interact with other components as well as with Things through communication links. Each communication link must meet a desired QoS (latency, upload and download bandwidth) to fully enable the interactions. Fig. 1 shows all software, hardware and QoS constraints for the smart agriculture application. For instance, the arrow from `ThingsController` to `Dashboard` indicates that the average latency, upload bandwidth and download bandwidth are 140 ms, 0.4 Mbps, and 1.1 Mbps, respectively[2]. Software and hardware requirements are indicated in the grey box associated to each component.

[1]In the following we will refer to IoT sensors and actuators also as Things.

[2]Symmetrically upload and download bandwidth from `DashBoard` to `ThingsController` are 1.1 Mbps and 0.4 Mbps, respectively.

To support this smart agriculture application, the IT consultants hired by the farm consider an infrastructure (Fig. 2) that consists of three Fog nodes connected to the Things and to two Cloud datacentres. Two Fog nodes (`fog1` and `fog2`) are owned by the farm, while `fog3` is a shared Fog node of a farms consortium.

In terms of hardware capabilities, all Fog nodes have 32 GB storage, 2GB RAM and 2 CPU cores[3]. On the other hand, Cloud nodes are assumed to have infinite hardware capabilities[4]. Fog and Cloud nodes offer a combination of different operating systems, programming languages and database support. The infrastructure provides all Things required by the application (i.e., moisture, water, fire and video camera). In terms of communication links, `fog1` is connected to the Internet via satellite access, `fog2` through mobile technologies, while `fog3` connects via a Fiber-to-the-Cabinet (FttC) access. The QoS profiles used in the example make use of real QoS data provided by different vendors[5]. Such data is represented as probability distributions[6] for upload/download bandwidths in Fig. 2 to model the variability of QoS of connections. For instance, the arrow from `fog1` to the Cloud indicates that average latency is 70 ms, and that the link can feature a 7 Mbps or 14 Mbps satellite Internet connection. In the case of 7 Mbps, when connection is up (98% of the cases), download and upload bandwidth are 6 Mbps and 0.75 Mbps, respectively. Furthermore, all Fog nodes are connected to each other through WLAN.

A first natural question is where to deploy (each component of) the smart agriculture application to the Fog infrastructure, so that all requirements (hardware, software and QoS) are satisfied:

> **Q1(a)** — *Which are the eligible deployments that minimise resource consumption (of Fog nodes) and "comply most" with the desired QoS constraints?*

It is easy to see that, even in this simple scenario, solving the (NP-hard [8]) problem of finding an eligible deployment that meets all constraints is hard to solve by hand.

Suppose that the farm now wants to perform what-if analyses to verify whether it is possible to avoid using some of the nodes, or to reduce their resource consumption, while maintaining the same QoS-assurance. For instance:

> **Q1(b)** — *Is it possible to avoid using (and paying for) the Fog node of the consortium (`fog3`)?*
>
> **Q1(c)** — *Is it possible to reduce the resource consumption of (our) Fog nodes `fog1` and `fog2`?*

Suppose now that the farm decides to activate the currently unused videocamera (Fig. 1). This increases the hardware requirements and changes the desired QoS for links with respect to the originally deployed application, also based on

[3]The hardware specs are those of the Dell Edge Gateway http://www.dell.com/us/business/p/edge-gateway.

[4]To reflect that Cloud customers can purchase more processing power, RAM and storage as needed, *as if* they were unbounded.

[5]Satellite access:https://www.eolo.it/, Mobile access:https://www.agcom.it, FttC access:http://www.vodafone.it.

[6]To simplify the example, we only consider Bernoulli distributed values.

the video stream quality. Suppose Standard Definition (SD) video requires 0.7 Mbps upload bandwidth and 16 GB storage for 24 hours recordings, and that an associated microservice is enabled on `DataStorage`. Then:

> **Q2** — *Are there eligible deployments after the application upgrade with SD video streaming? What is their compliance with the desired QoS?*

Suppose that supporting High Definition (HD) videostreaming requires more resources, namely 3.5 Mbps uplink and 32 GB storage for 24 hours recordings. It may also be the case that the farm wants to record more than 24 hours, e.g. one week, in order to increase the machine learning accuracy or for surveillance. Then:

> **Q3(a)** — *Are there eligible deployments which support HD video streaming with the available infrastructure? Do they comply with the desired QoS?*
>
> **Q3(b)** — *If the current infrastructure does not support HD video streaming adequately, is there an upgrade for the access technology at either* `fog1` *(from 7 to 14 Mbps) or* `fog2` *(from 3G to 4G) that permits HD streaming?*
>
> **Q3(c)** — *Is it possible to support the recording of one week of streamed HD video (1 TB of data for* `DataStorage`*)?*

In Section IV we will use our prototype FogTorchΠ to answer all the above questions.

III. FogTorch: Model and Prototype

Our approach is based on the formal model and algorithms presented in [8].

A. The Model

We first formalise the concept of QoS profile as follows.

Definition 1. *The set Q of* QoS profiles *is a set of pairs $\langle \ell, b \rangle$ where ℓ and b denote respectively the average latency and bandwidth featured by (or required for) a communication link. The bandwidth is a pair $(b_{\downarrow}, b_{\uparrow})$, distinguishing the download and upload bandwidth of a link. Unknown/unspecified values of latency or bandwidth will be denoted by $\perp$.*

Def. 1 makes it possible to model the QoS attributes of communication links according to their historical behaviour, obtained via proper monitoring tools.

A Fog infrastructure consists of Things, Fog and Cloud nodes at a given location, featuring hardware, software and IoT capabilities. For Cloud nodes, hardware capabilities are considered unbounded, as mentioned in Section II.

Definition 2. *A* Fog infrastructure *is a 4-tuple $\langle T, F, C, L \rangle$ where:*

- *T is a set of* Things*, each denoted by a tuple $t = \langle i, \pi, \tau \rangle$ where i is the identifier of t, π denotes its location and τ its type,*
- *F is a set of* Fog nodes*, each denoted by a tuple $f = \langle i, \pi, \mathcal{H}, \Sigma, \Theta \rangle$ where i is the identifier of f, π denotes its location, $\mathcal{H}$ and Σ are the hardware and software capabilities it provides, and $\Theta \subseteq T$ contains all Things directly reachable from f,*
- *C is a set of available* Cloud data-centres*, each denoted by a tuple $c = \langle i, \pi, \Sigma \rangle$ where i is the identifier of c, π denotes its location and Σ the software capabilities it provides,*
- *$L \subseteq \{\langle N', N'', q \rangle | N', N'' \subseteq F \cup C \wedge q \in Q\}$ is a set of available Fog-to-Fog, Fog-to-Cloud, and Cloud-to-Fog* communication hyperlinks[7]*, each associated to its QoS profile.*

Applications have hardware and software requirements associated with their components, and desired QoS profiles for component-component and component-Things interactions. For what concerns Things, current proposals for the Fog computing architecture (e.g., [6], [12]) envision remote access to IoT resources via a specific middleware layer. Accordingly, we assume Fog and Cloud nodes being able to access directly connected Things with negligible latency and infinite bandwidth, and Things at neighbouring nodes through the associated communication links.

Definition 3. *An* application *is a triple $\langle \Gamma, \Lambda, \overline{\Theta} \rangle$ where:*

- *Γ is a set of* software components*, each denoted by a tuple $\gamma = \langle i, \overline{\mathcal{H}}, \overline{\Sigma} \rangle$ where i is the identifier of γ, and $\overline{\mathcal{H}}$ and $\overline{\Sigma}$ the hardware and software requirements it has.*
- *$\Lambda \subseteq \{\langle \gamma, \gamma', q \rangle | (\gamma, \gamma') \in (\Gamma \times \Gamma) \wedge q \in Q\}$ denotes the existing* interactions among components[8] *in Γ, each expressing the desired QoS profile for the connection that will support it.*
- *$\overline{\Theta}$ is a set of* Things requests *each denoted by $\langle \gamma, \tau, q \rangle$, where $\gamma \in \Gamma$ is a software component and τ denotes a type of Thing γ needs to reach with QoS profile q.*

To make sure applications sense from and act upon the correct IoT devices, deployment designers are asked to specify a *Things binding* between Things requests $\overline{\Theta} \in A$ of an application and actual Things in the infrastructure $T \in I$.

Definition 4. *Let $A = \langle \Gamma, \Lambda, \overline{\Theta} \rangle$ be an application and $I = \langle T, F, C, L \rangle$ a Fog infrastructure. A* Things binding *$\vartheta : \overline{\Theta} \to T$ for A over I is a mapping from each Thing request onto a specific Thing.*

Finally, when business-related constraints (e.g., commercial or legal) forbid some components from being deployed on any node of the infrastructure, deployment designers can specify the whitelist of Fog or Cloud nodes on which each component can be installed.

[7] Def. 2 refines [8] by representing communication links as *hyperlinks* [11] to model Internet access technologies in which single (xDSL, FttX, ...) access points may support many Fog-to-Cloud and Cloud-to-Fog connections, or may serve more than one Fog node simultaneously. We assume that if $\langle N', N'', q \rangle \in L$ then $\langle N', N'', q' \rangle \notin L$ with $q \neq q'$. We also assume that if $\langle N', N'', \langle \ell, b_{\downarrow}, b_{\uparrow} \rangle \rangle \in L$ then $\langle N'', N', \langle \ell', b'_{\downarrow}, b'_{\uparrow} \rangle \rangle \in L$ and $\ell = \ell'$, $b_{\downarrow} = b'_{\uparrow}$ and $b_{\uparrow} = b'_{\downarrow}$.

[8] As before, we assume that if $\langle \gamma, \gamma', q \rangle \in \Lambda$ then $\langle \gamma, \gamma', q' \rangle \notin \Lambda$ with $q \neq q'$. We also assume that if $\langle \gamma, \gamma', \langle \ell, b_{\downarrow}, b_{\uparrow} \rangle \rangle \in \Lambda$ and $\langle \gamma', \gamma, \langle \ell', b'_{\downarrow}, b'_{\uparrow} \rangle \rangle \in \Lambda$ then $\ell = \ell'$, $b_{\downarrow} = b'_{\uparrow}$ and $b_{\uparrow} = b'_{\downarrow}$.

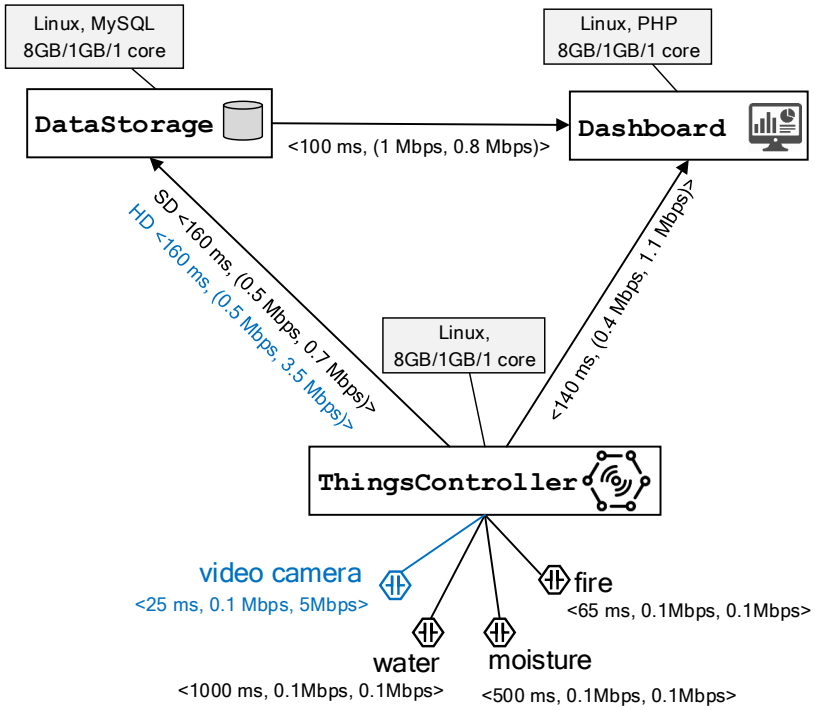


Fig. 1: A smart agriculture application.

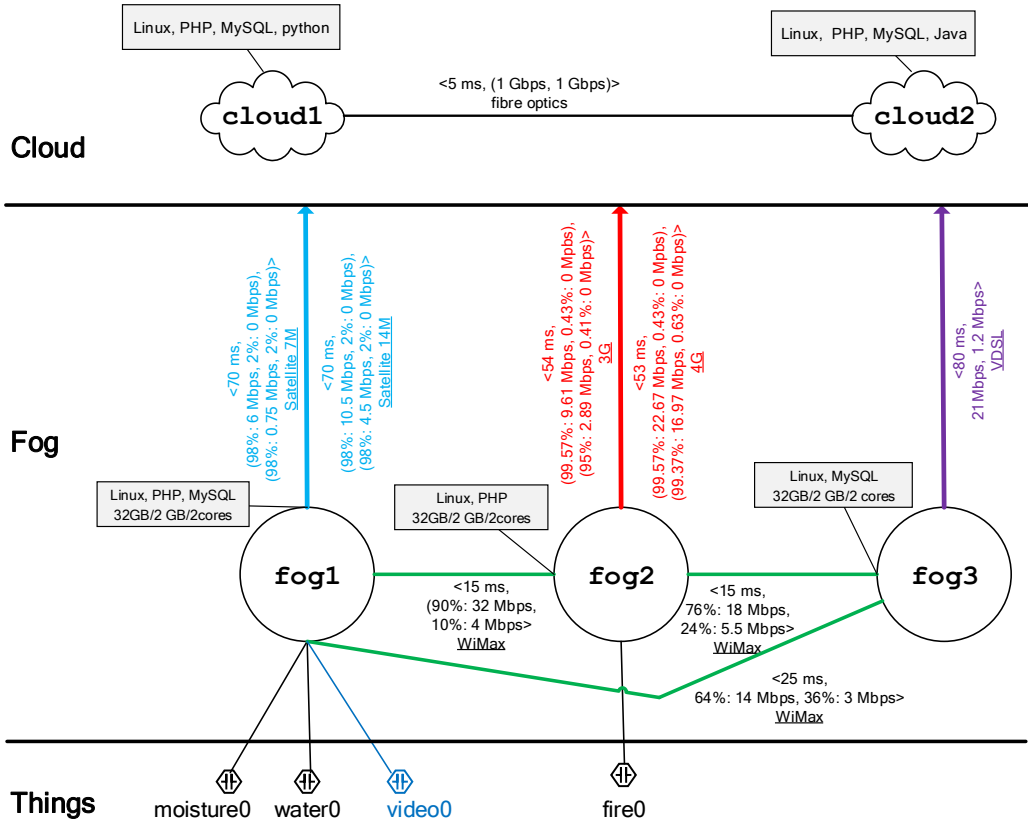


Fig. 2: Fog infrastructure for the smart agriculture example.

Definition 5. *Let $A = \langle \Gamma, \Lambda, \overline{\Theta} \rangle$ be an application and $I = \langle T, F, C, L \rangle$ a Fog infrastructure. A* deployment policy $\delta : \Gamma \to 2^{F \cup C}$ *for A over I is mapping from each*[9] *software component of A onto the set of nodes where its deployment is permitted (or has been already performed).*

In what follows, we will consider a software component *compatible* with a Fog or Cloud node when the software/hardware capabilities of the node can support *at least* the software/hardware requirements for that component [8].

An eligible deployment for an application A over an available Fog infrastructure I must: (1) satisfy compatibility and deployment policies, (2) not exceed hardware capacity at each Fog node, (3) fulfil Things requests binding, and (4) meet latency constraints without exceeding the available links bandwidths for both components interactions and remote Things access.

Definition 6. *Let $A = \langle \Gamma, \Lambda, \overline{\Theta} \rangle$ be an application, $I = \langle T, F, C, L \rangle$ a Fog infrastructure, δ a deployment policy and ϑ a Things binding for A over I. Then, $\Delta : \Gamma \to F \cup C$ is an* eligible deployment *for A on I that complies with δ and ϑ if and only if:*

1) *for each $\gamma \in \Gamma$, $\Delta(\gamma) \in \delta(\gamma)$ and γ is compatible with $\Delta(\gamma)$,*
2) *let $\Gamma_f = \{\gamma \in \Gamma \mid \Delta(\gamma) = f\}$ be the set of components of A mapped onto $f \in F$. Then*[10]*, for each $f = \langle i, \pi, \mathcal{H}, \Sigma, \Theta \rangle \in F$: $\mathcal{H} \geqslant \sum_{\langle j, \overline{\mathcal{H}}, \overline{\Sigma} \rangle \in \Gamma_f} \overline{\mathcal{H}}$,*
3) *for each Thing binding $\vartheta(\langle \gamma, \tau, q \rangle) = t$, there exists $f = \langle i, \pi, \mathcal{H}, \Sigma, \Theta \rangle \in F$ such that $t \in \Theta$ and $\Delta(\gamma) = f$ or $\langle N', N'', q' \rangle \in L$ such that $\Delta(\gamma) \in N'$ and $f \in N''$.*
4) *let $Q_{(N',N'')}$ be*[11] *the multi-set of QoS profiles associated with component-component and component-Thing interactions that are mapped on the communication (hyper)link $N' - N''$. Then*[10]*, for each $\langle N', N'', \langle \ell, b \rangle \rangle \in L$: $\langle \overline{\ell}, \overline{b} \rangle \in Q_{(N',N'')} \implies \ell \leq \overline{\ell} \ \wedge \ b \geqslant \sum_{\langle \overline{\ell}, \overline{b} \rangle \in Q_{(N',N'')}} \overline{b}$.*

B. The Algorithm

The algorithm consists of a preprocessing phase (PREPROCESS(A, I, δ, ϑ)) and of a backtracking search phase (BACKTRACKSEARCH(D, Δ, A, I, K, ϑ)), as shown in Fig. 3. Given an application A, a Fog infrastructure I, a deployment policy δ, and a Things binding ϑ, it returns the (possibly empty) set D of all eligible deployments Δ_j for A over I that comply with δ and ϑ.

Preprocessing builds, for each software component $\gamma \in \Gamma$ of A, the set $K[\gamma]$ of Fog and Cloud nodes that satisfy conditions (1) and (3) of Def. 6, and also meet latency requirements for the Things request of a component. If there exists even one component for which $K[\gamma]$ is empty, the algorithm immediately returns an empty set of deployments. This phase completes in $O(N)$ with $N = |F \cup C|$ time.

```
1: procedure FINDDEPLOYMENTS(A, I, δ, ϑ)
2:     K ← PREPROCESS(A, I, δ, ϑ)
3:     if K = failure then
4:         return ∅                          ▷ ∃γ̄ ∈ Γ s.t. K[γ̄] = ∅
5:     end if
6:     D ← ∅
7:     BACKTRACKSEARCH(D, ∅, A, I, K, ϑ)
8:     return D
9: end procedure
```

Fig. 3: Pseudocode of the exhaustive search algorithm.

```
1: procedure BACKTRACKSEARCH(D, Δ, A, I, K, ϑ)
2:     if ISCOMPLETE(Δ) then
3:         ADD(Δ, D)
4:     end if
5:     γ ← SELECTUNDEPLOYEDCOMPONENT(Δ, A);
6:     for all n ∈ SELECTDEPLOYMENTNODE(K[γ], A) do
7:         if ISELIGIBLE(Δ, γ, n, A, I, ϑ) then
8:             DEPLOY(Δ, γ, n, A, I, ϑ)
9:             BACKTRACKSEARCH(D, Δ, A, I, K, ϑ)
10:            UNDEPLOY(Δ, γ, n, I, A, ϑ)
11:        end if
12:    end for
13: end procedure
```

Fig. 4: Pseudocode for the backtracking search.

The backtracking search algorithm inputs the result of the preprocessing and looks for eligible deployments, as listed in Fig. 4. It visits a (finite) search space tree having at most $N = |F \cup C|$ nodes at each level and height $|\Gamma|$. Each node in the state space represents a (partial) deployment, where the number of deployed components corresponds to the level of the node. The root corresponds to an empty deployment, nodes at level i are partial deployments of i components, and leaves at level $|\Gamma|$ contain complete eligible deployments. Edges from one node to another represent the action of deploying a component onto some Fog node or Cloud node. Thus, search completes in $O(N^{|\Gamma|})$ time.

At each recursive call, BACKTRACKSEARCH(D, Δ, A, I, K, ϑ) first checks whether all components of A have been deployed by the currently attempted plan and, if so, adds the found deployment to set D. Otherwise, it selects a component yet to be deployed (SELECTUNDEPLOYEDCOMPONENT(Δ, A)) and attempts to deploy it to a node chosen in $K[\gamma]$ (SELECTDEPLOYMENTNODE($K[\gamma]$, A)). The ISELIGIBLE(Δ, γ, n, A, I, ϑ) procedure checks conditions (2) and (4) of Def. 6 and, when they hold, the DEPLOY(Δ, γ, n, A, I, ϑ) procedure decreases the available hardware resources and bandwidths in the infrastructure according to the new deployment association. UNDEPLOY(Δ, γ, n, I, A, ϑ) performs the dual operation, when needed.

[9] If $\delta(\gamma)$ is not specified we assume $\delta(\gamma) = F \cup C$.

[10] Abusing notation, $\sum$ is used to sum the hardware/bandwidth requirements, and $\geqslant$ to compare them with available offerings.

[11] Formally: $Q_{(N',N'')} =$
$\{\!\{ \langle \ell, b \rangle \mid \langle \gamma, \gamma', \langle \ell, b \rangle \rangle \in \Lambda \ \wedge \ \Delta(\gamma) \in N' \ \wedge \ \Delta(\gamma') \in N'' \}\!\} \cup$
$\{\!\{ \langle \ell, b \rangle \mid \ \langle \gamma, \tau, \langle \ell, b \rangle \rangle \in \overline{\Theta} \ \wedge \ \vartheta(\langle \gamma, \tau, \langle \ell, b \rangle \rangle) = t \ \wedge \ \Delta(\gamma) \in N' \ \wedge N'' \ni \langle i, \pi, \mathcal{H}, \Sigma, \Theta \rangle \in F \ \wedge \ t \in \Theta \}\!\}$.

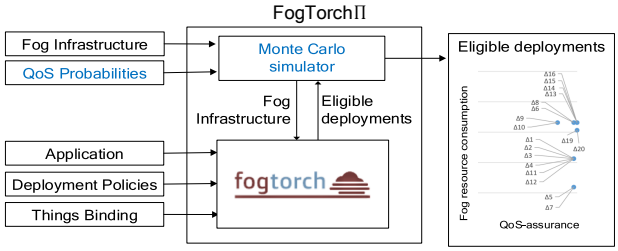


Fig. 5: Bird-eye view of FogTorchΠ.

```
1: p ∈ [0,1] ∧ q, q′ ∈ Q
2: procedure SamplingFunction(p, q, q′)
3:     r ← RandomDoubleInRange(0,1)
4:     if r ⩽ p then
5:         return q
6:     else
7:         return q′
8:     end if
9: end procedure
```

Fig. 6: Sampling function example.

C. Overview of FogTorchΠ

FogTorchΠ is an open source prototype[12], developed in Java, extending our previous prototype implementation (FogTorch) of the model and algorithms previously described. FogTorch takes into account all non-functional parameters within the model (i.e., hardware, software, QoS). In the case of hardware capabilities, it considers CPU cores, RAM and storage. Software capabilities are represented by a list of software names (operating system, programming languages, frameworks etc). As per Def. 1, FogTorch considers latency, and both download and upload bandwidths as QoS attributes. Latency is measured in milliseconds (ms), while bandwidth is given in Megabits per second (Mbps).

To account for probabilities when assigning QoS profiles to communication hyperlinks in L, FogTorchΠ (Fig. 5) employs the Monte Carlo method [9]. In addition to the input of FogTorch, FogTorchΠ permits to associate more than one QoS profile to each communication link, according to a probability distribution. FogTorchΠ is run a sufficiently large number of times and, at each run, it inputs a particular Fog infrastructure, choosing a QoS profile for each communication link. A sampling function – consider Fig. 6 as an example – is called to generate samples according to the given distributions. The output is the set of eligible deployments generated at each run. At the end of the simulation, results from all runs are aggregated by computing two metrics for each obtained eligible deployment:

1) *QoS-assurance*, corresponding to the percentage of runs that generated that deployment. Indeed, the more a deployment is generated during the simulation, the more it is likely to meet all desired QoS constraints in the actual varying infrastructure.

[12]Available at https://github.com/di-unipi-socc/FogTorchPI.

2) *Fog resource consumption*, indicating the aggregated averaged percentage of consumed RAM and storage in a subset $\overline{F} \subseteq F$ of Fog nodes ($\overline{F} = F$ by default), i.e. if $\Gamma_{\overline{F}} = \{\gamma \in \Gamma | \Delta(\gamma) \in \overline{F}\}$ then:

$$\frac{1}{2}\left(\frac{\Sigma_{\gamma \in \Gamma_{\overline{F}}} RAM(\gamma)}{\Sigma_{f \in \overline{F}} RAM(f)} + \frac{\Sigma_{\gamma \in \Gamma_{\overline{F}}} storage(\gamma)}{\Sigma_{f \in \overline{F}} storage(f)}\right)$$

To summarise, FogTorchΠ accounts for variations in the QoS profiles over the communication links and for resource consumption in the Fog layer. As we will discuss next, the proposed aggregate metrics may represent meaningful measures to evaluate eligible deployments at design time.

IV. Motivating Example (continued)

In this section we exploit FogTorchΠ to address the questions raised in the smart agriculture example of Section II. As per Section III, FogTorchΠ outputs all eligible deployments, accompanied by their QoS-assurance and Fog resources consumption values. The final choice of a particular deployment among the eligible candidates is however left to users (the IT experts in this example), leaving them the freedom of choosing how to trade-off between achievable QoS-assurance and Fog resources consumption (and possibly also taking into account other application specific, technical or business considerations). To help decision making, for all questions we discuss the eligible deployments, displayed as scatter plots on the axes QoS-assurance and Fog resource consumption.

To illustrate how FogTorchΠ works, we first describe how it executes to find the eligible deployments for instance for the scenario of **Q1.(a)**. In its preprocessing step FogTorchΠ determines, for each application component `c`, the set `K[c]` of eligible Fog and Cloud nodes for `c`:

```
K[ThingsController] = [fog3,fog1,fog2]
K[DataStorage] = [cloud2,cloud1,fog3,fog1]
K[Dashboard] = [cloud2,cloud1,fog1,fog2]
```

Note that `K` indicates that the `ThingsController` component can only be deployed on Fog nodes, while `DataStorage` and `Dashboard` can be deployed also on Cloud nodes.

In the search step (Fig. 4), FogTorchΠ selects one undeployed component at a time. If FogTorchΠ selects for instance component `ThingsController`, it checks whether `ThingsController` can be deployed on the first node `fog3` in `K[ThingsController]`. After checking that `fog3` satisfies all (hardware, software and QoS) requirements of `ThingsController`, FogTorchΠ adds the deployment of `ThingsController` on `fog3` to the currently computed deployment. FogTorchΠ also updates the amount of consumable resources (RAM, storage) available at `fog3`, as well as the available bandwith between `fog3` and `fog1` (as `ThingsController` will access Things `moisture0` and `water0`, which are physically connected to `fog1`) and the bandwith between `fog3` and `fog2` (as `ThingsController` will also access Thing `fire0`, which is connected to `fog2`).

If the algorithm chooses next component `DataStorage`, it checks whether `DataStorage` can be deployed on the first node `cloud2` in `K[DataStorage]`. After checking that `cloud2` satisfies all (hardware, software and QoS) requirements of `DataStorage`, the algorithm adds the deployment of `DataStorage` on `cloud2` to the currently computed deployment, and it updates the bandwith between `cloud2` and `fog3` (as `DataStorage` interacts with `ThingsController`).

Finally, the algorithm checks whether `Dashboard` can be deployed on the first node `cloud2` in `K[Dashboard]`. As `cloud2` does not satisfy all (hardware, software and QoS) requirements of `Dashboard`, the algorithm then checks `cloud1` and `fog1`. Since none of them satisfies all requirements, FogTorchΠ checks `fog2`, which actually satisfies all requirements. The algorithm hence adds the deployment of `Dashboard` on `fog2` to the currently computed deployment, and it updates the amount of consumable resources available at `fog2`, as well as the bandwith between `fog2` and `fog3`, and between `fog2` and `cloud2`. The algorithm will hence add the computed deployment Δ:

Δ(`ThingsController`) = `fog3`
Δ(`DataStorage`) = `cloud2`
Δ(`Dashboard`) = `fog2`

to the set of eligible deployments. Note that, if all Fog nodes fail to satisfy the requirements, the algorithm backtracks to the previously deployed component and tries to deploy it to another node. For example, if no nodes had satisfied the requirements of component `Dashboard`, the algorithm would have backtracked to try deploying `DataStorage` on another node (`cloud1`) to retry deploying `Dashboard`.

For question **Q1(a)** (Fig. 8), FogTorchΠ determines[13] 20 eligible deployments (Δ1—Δ20 of Fig.7). Fig. 8 shows for instance that deployments Δ1—Δ4 minimise Fog resource consumption, while satisfying all QoS requirements nearly 100% of the times. Dually, QoS-assurance is maximised (reaching 100%) by Fog-only deployments Δ13—Δ20, which on the other hand consume considerably more Fog resources.

As for question **Q1(b)** (Fig. 9), FogTorchΠ determines that it is indeed possible to avoid using `fog3` for deploying the application. Fig. 9 shows that if `fog3` $\notin \delta$(`ThingsController`) $\cup \delta$(`DataStorage`) $\cup \delta$(`Dashboard`) then the number of eligible deployments halves with respect to Fig. 8.

Fig. 10 shows the answer provided by FogTorchΠ to **Q1(c).** Namely, if resource consumption is assessed only over the set of Fog nodes $\overline{F} = \{$`fog1`, `fog2`$\}$ then deployments Δ5 and Δ7 minimize such resource consumption, while featuring a high QoS-assurance.

Fig. 11 shows the answer provided by FogTorchΠ to **Q2**, namely, that SD video streaming is possible. However, keeping

[13]The number of Monte Carlo runs was 100,000 for all questions. All FogTorchΠ outputs of the example are available at https://github.com/di-unipi-socc/FogTorchPI/tree/master/results.

Deployment ID	Things Controller	Data Storage	Dashboard
Δ1	fog2	cloud2	cloud1
Δ2	fog2	cloud2	cloud2
Δ3	fog2	cloud1	cloud2
Δ4	fog2	cloud1	cloud1
Δ5	fog3	cloud1	fog2
Δ6	fog2	cloud2	fog2
Δ7	fog3	cloud2	fog2
Δ8	fog2	cloud1	fog2
Δ9	fog1	cloud2	fog2
Δ10	fog1	cloud1	fog2
Δ11	fog2	fog3	cloud2
Δ12	fog2	fog3	cloud1
Δ13	fog2	fog3	fog1
Δ14	fog1	fog3	fog2
Δ15	fog1	fog3	fog1
Δ16	fog2	fog3	fog2
Δ17	fog2	fog1	fog1
Δ18	fog2	fog1	fog2
Δ19	fog3	fog1	fog2
Δ20	fog3	fog1	fog1
Δ21	fog1	cloud2	fog1
Δ22	fog1	cloud1	fog1
Δ23	fog2	fog1	cloud2
Δ24	fog2	fog1	cloud1
Δ25	fog1	fog3	cloud1
Δ26	fog3	fog1	cloud2
Δ27	fog1	fog3	cloud2
Δ28	fog3	fog1	cloud1

Fig. 7: Index of all deployments generated by FogTorchΠ (IDs are used in Figs. 8 –14). Note that, due to changes in the input (infrastructure, application), not all deployments are generated for each question.

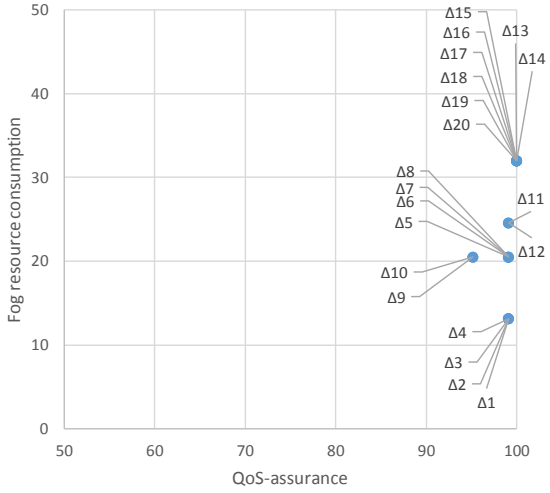


Fig. 8: Eligible deployments for **Q1(a)** — *Which are the eligible deployments that minimise resource consumption (of Fog nodes) and "comply most" with the desired QoS constraints?*

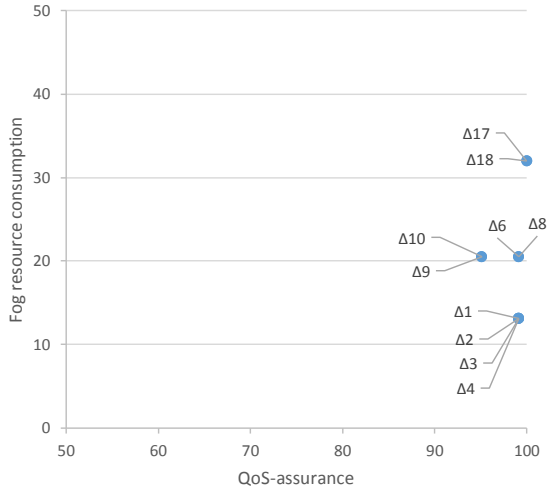


Fig. 9: Eligible deployments for **Q1(b)** — *Is it possible to avoid using (and paying for) the Fog node of the consortium (*`fog3`*)?*

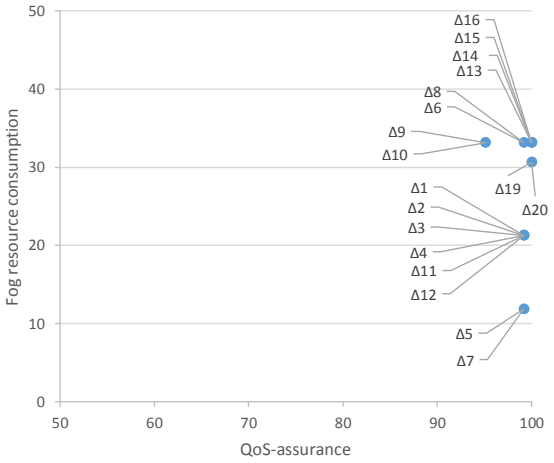


Fig. 10: Eligible deployments for **Q1(c)** — *Is it possible to reduce the resource consumption of (our) Fog nodes* `fog1` *and* `fog2`*?*

high values of QoS assurance (and avoiding Fog-only deployments) requires increased consumption of Fog resources, as for instance in deployments $\Delta 9$ and $\Delta 10$.

Fig. 12 shows that enabling HD videostreaming on the available infrastructure (**Q3(a)**) cannot ensure QoS-assurance higher than 90%, not even by heavily exploiting Fog resources, and hence an upgrade to the infrastructure is needed to support such new feature.

Fig. 13 provides a comparison of the two candidate upgrades (**Q3(b)**). It is easy to see that both upgrades improve the results of **Q3(a)** for HD video streaming. However, exploiting few more Fog resources, the satellite upgrade features higher QoS-assurance ($\geq 95\%$) than the 4G upgrade.

Assuming that the satellite upgrade (from 7 to 14 Mbps) is performed, Fig. 14 shows that it is then possible to support the higher storage requirement of (question **Q3(c)**) by deploying `CloudStorage` in the Cloud, while having very good values

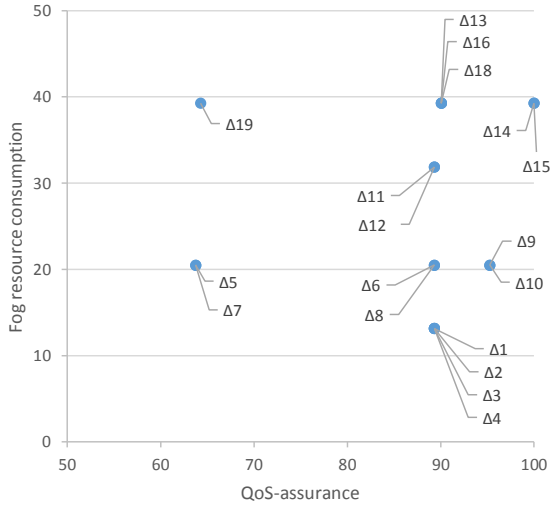


Fig. 11: Eligible deployments for **Q2** —*Are there eligible deployments after the application upgrade with SD video streaming? What is their compliance with the desired QoS?*

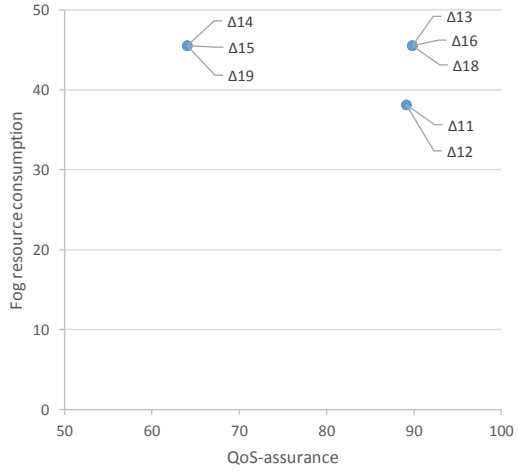


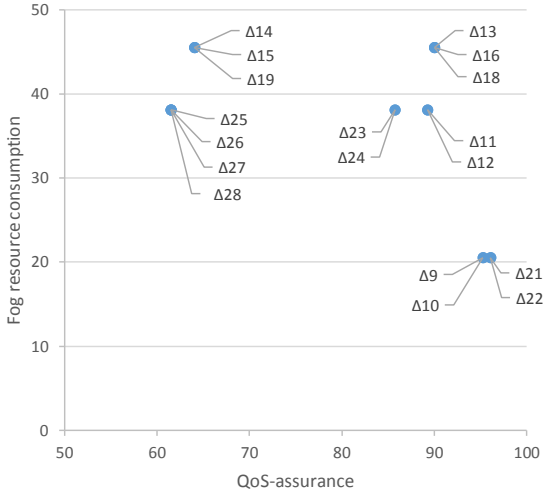
Fig. 12: Eligible deployments for **Q3(a)** — *Are there eligible deployments which support HD video streaming with the available infrastructure? Do they comply with the desired QoS?*

of both QoS-assurance and Fog resource consumption.

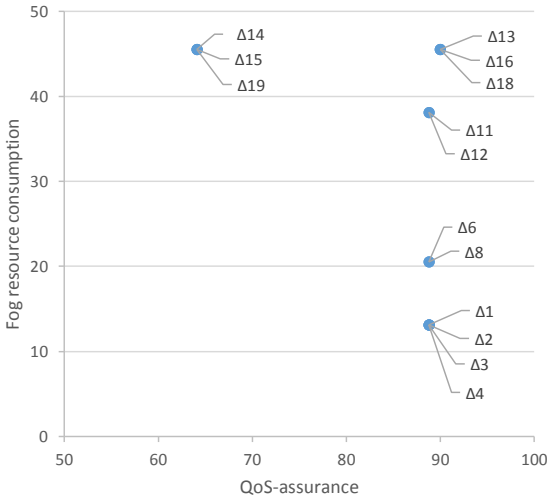
V. Related Work

The problem of deploying multi-component applications has been thoroughly studied in the Cloud scenario. Projects like SeaClouds [13], Aeolus [14] or Cloud4SOA [15], for instance, proposed optimised planning solutions to deploy software applications across different (IaaS or PaaS) Clouds. [16] proposed to use OASIS TOSCA [17] to model IoT applications in Cloud+IoT scenarios.

Recently, [18] has linked services and networks QoS by proposing a QoS- and connection-aware Cloud service composition approach to satisfy end-to-end QoS requirements in the Cloud. The Fog, however, introduces new problems, mainly due to its need for connection-awareness and interactions with the IoT, that were not taken into account by [18].



(a) Satellite 14 Mbps upgrade.



(b) 4G upgrade.

Fig. 13: Eligible deployments for **Q3(b)** — *If the current infrastructure does not support HD video streaming adequately, is there an upgrade for the access technology at either* `fog1` *(from 7 to 14 Mbps) or* `fog2` *(from 3G to 4G) that permits HD streaming?*

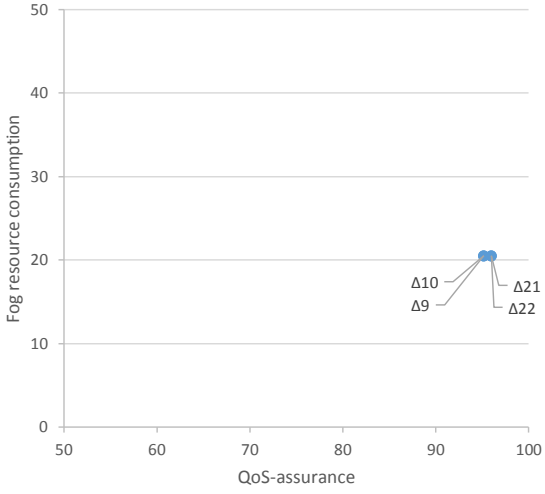


Fig. 14: Eligible deployments for **Q3(c)** — *Is it possible to support the recording of one week of streamed HD video (1 TB of data for* `DataStorage`*)?*

To the best of our knowledge, very few approaches have been proposed so far to specifically model Fog infrastructures, applications and application deployments. [19] aims at evaluating service latency and energy consumption of the new Fog paradigm applied to the IoT, as compared to traditional Cloud scenaria. The model of [19], however, deals only with applications already deployed over Fog infrastructures. [20] follows a more pragmatic approach, by proposing a C++ programming framework for the Fog that provides APIs for resource discovery and QoS-aware incremental deployment via containerisation. With respect to our work, [20] takes into account Fog nodes workload but it does not consider bandwidth, Things requests, and deployment policies as leading parameters for deployment. Also, programmers have to manually segment functionalities of their applications, by *a priori* determining the number of layers needed in the Fog hierarchy. Recently, [21] has proposed a Fog middleware capable of provisioning resources to scheduled tasks. A control node in the Fog hierarchy manages the distribution of tasks between the Fog and the Cloud so to minimise overall delay. Our model differs from [21] since we account for multi-component applications to be deployed and we focus on their compliance with user-provided requirements.

[22] prototyped a simulator to evaluate resource management and scheduling policies applicable to Fog environments with respect to their impact on latency, energy consumption and operational cost. [22] differs from our approach mainly since it only models tree-like infrastructure topologies (not accounting for the very possibility of sharing IoT resources among Fog nodes), and it only considers applications whose topology can be modelled by a DAG. Furthermore, it does not consider QoS requirements among the parameters defining the set of eligible deployments.

[23] presents a technique to estimate where to deploy components of a given application. The objective was to identify Fog or Cloud nodes to minimise execution time and energy consumption of an application. Akin processing and QoS requirements as in FogTorchΠ were considered (CPU, RAM, storage, bandwidth, and latency), but application components were deployed monolithically on single Fog or Cloud nodes. Furthermore, [23] does not consider interaction with the IoT nor resource consumption at deployment time. Apropos, [10] presents a model to estimate resource consumption for Fog applications. The model is based upon the fact that available resources on Fog and Cloud nodes vary over time, according to users behaviour. [10] probabilistically estimates the amount of available resources to minimise over-provisioning rather than to determine eligible deployments or to estimate their QoS-assurance.

Finally, [24] presents a model and algorithms to perform workload management in Fog infrastructures. Their objective

is to distribute tasks on Fog nodes while minimising their average completion time based upon demanded resources. [24] focuses on tasks scheduling (accounting for non-functional parameters), but without considering specific QoS requirements for the applications to be deployed.

The problem of finding an eligible deployment of components over a Fog infrastructure resembles the Subgraph Isomorphism problem (although it also includes the possibility of mapping more than one component onto the very same node). Solutions to Subgraph Isomorphism have been proposed in the context of Virtual Network embedding [25], [26] and deployment over WAN [27], by performing node and link mapping in a single phase, as we do.

VI. Concluding Remarks

In this paper we presented the FogTorchΠ prototype, which permits to determine eligible deployments of composite applications over Fog infrastructures.

FogTorchΠ can be exploited to simulate and compare different Fog scenarios at design time, determining context-, resource- and QoS-aware deployments of IoT applications over the Cloud-Fog continuum. To do so, FogTorchΠ takes into account both processing (e.g., CPU, RAM, storage, software) and QoS (e.g., latency, bandwidth) constraints that are relevant for real-time Fog applications.

To the best of our knowledge, FogTorchΠ is the first prototype capable of estimating the QoS-assurance of composite Fog applications deployments based on probability distributions of bandwidth and latency featured by communication links. FogTorchΠ also estimates resource consumption in the Fog layer, which can be used to minimise the exploitation of certain Fog nodes with respect to others, depending on the user needs. The potential of FogTorchΠ has been illustrated by discussing its application to a smart agriculture Fog application, performing what-if analyses at design time, including changes in Fog nodes or applications requirements, exploited IoT devices and QoS featured by communication links.

We see three main directions for future work. One of them is to extend the model on which FogTorchΠ is based with other QoS attributes, and to include cost information to get a richer classification of eligible deployments. Another direction for future work is to account for multiple and multi-tenant deployments on the same infrastructure. Last, but not least, we intend to experiment FogTorchΠ on concrete case studies that are currently under development, in order to assess the effectiveness of and to refine the model on which FogTorchΠ is based, and to improve the scalability of FogTorchΠ by exploiting heuristics to reduce the search space.

Acknowledgements. This work was partly supported by project PRA_2016_64 *Through the Fog*, funded by the University of Pisa.

References

[1] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854–864, 2016.

[2] W. Shi and S. Dustdar, "The Promise of Edge Computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

[3] A. V. Dastjerdi and R. Buyya, "Fog Computing: Helping the Internet of Things Realize its Potential," *Computer*, vol. 49, no. 8, pp. 112–116, 2016.

[4] I. Stojmenovic, S. Wen, X. Huang, and H. Luan, "An overview of Fog computing and its security issues," *Concurrency and Computation: Practice and Experience*, 2015.

[5] R. Mahmud and R. Buyya, "Fog Computing: A Taxonomy, Survey and Future Directions," *arXiv preprint arXiv:1611.05539*, 2016.

[6] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big Data and Internet of Things: A Roadmap for Smart Environments*, 2014, pp. 169–186.

[7] J.-P. Arcangeli, R. Boujbel, and S. Leriche, "Automatic deployment of distributed software systems: Definitions and state of the art," *Journal of Systems and Software*, vol. 103, pp. 198–218, 2015.

[8] A. Brogi and S. Forti, *QoS-aware Deployment of IoT Applications Through the Fog*. University of Pisa, November 2016, vol. Technical Report. [Online]. Available: http://eprints.adm.unipi.it/2362

[9] W. L. Dunn and J. K. Shultis, *Exploring Monte Carlo Methods*. Elsevier, 2011.

[10] M. Aazam and E. N. Huh, "Dynamic Resource Provisioning Through Fog Micro Datacenter," in *2015 IEEE Int. Conf. on Pervasive Computing and Communication Workshops*, 2015, pp. 105–110.

[11] G. Gallo, G. Longo, S. Pallottino, and S. Nguyen, "Directed hypergraphs and applications," *Discrete Applied Mathematics*, vol. 42, no. 2, pp. 177 – 201, 1993.

[12] "OpenFog Consortium," http://www.openfogconsortium.org/, accessed: 20/09/2016.

[13] A. Brogi, A. Ibrahim, J. Soldani, J. Carrasco, J. Cubo, E. Pimentel, and F. D'Andria, "SeaClouds: a European project on seamless management of multi-cloud applications," *ACM SIGSOFT SEN*, vol. 39, no. 1, pp. 1–4, 2014.

[14] R. Di Cosmo, A. Eiche, J. Mauro, G. Zavattaro, S. Zacchiroli, and J. Zwolakowski, "Automatic deployment of software components in the cloud with the aeolus blender," in *ICSOC 2015*, 2015, pp. 397–411.

[15] A. Corradi, L. Foschini, A. Pernafini, F. Bosi, V. Laudizio, and M. Seralessandri, "Cloud PaaS Brokering in Action: The Cloud4SOA Management Infrastructure," in *VTC 2015*, 2015, pp. 1–7.

[16] F. Li, M. Vögler, M. Claeßens, and S. Dustdar, "Towards automated IoT application deployment by a cloud-based approach," in *SOCA 2013*, 2013, pp. 61–68.

[17] A. Brogi, J. Soldani, and P. Wang, *TOSCA in a Nutshell: Promises and Perspectives*. Proceedings of ESOCC 2014., 2014, pp. 171–186.

[18] S. Wang, A. Zhou, F. Yang, and R. N. Chang, "Towards Network-Aware Service Composition in the Cloud," *IEEE Transactions on Cloud Computing*, 2016.

[19] S. Sarkar and S. Misra, "Theoretical modelling of fog computing: a green computing paradigm to support IoT applications," *IET Networks*, vol. 5, no. 2, pp. 23–29, 2016.

[20] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *DEBS 2016*, 2016, pp. 258–269.

[21] O. Skarlat, S. Schulte, M. Borkowski, and P. Leitner, "Resource Provisioning for IoT Services in the Fog," in *SOCA*. IEEE, 2016, pp. 32–39.

[22] H. Gupta, A. V. Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A toolkit for modeling and simulation of resource management techniques in internet of things, edge and fog computing environments," *arXiv preprint arXiv:1606.02007*, 2016.

[23] M. A. Hassan, M. Xiao, Q. Wei, and S. Chen, "Help Your Mobile Applications with Fog Computing," in *IEEE Int. Conf. on Sensing, Communication, and Networking - Workshops*, June 2015, pp. 1–6.

[24] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System," *IEEE Transactions on Computers*, vol. 65, no. 12, pp. 3702–3712, Dec 2016.

[25] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proc. 1st ACM workshop on virtualized infrastructure systems and architectures*, 2009, pp. 81–88.

[26] M. Chowdhury, M. R. Rahman, and R. Boutaba, "Vineyard: Virtual network embedding algorithms with coordinated node and link mapping," *IEEE/ACM TON*, vol. 20, no. 1, pp. 206–219, 2012.

[27] T. Kichkaylo, A. Ivan, and V. Karamcheti, "Constrained component deployment in wide-area networks using AI planning techniques," in *IPDPS 2003*, 2003.