UNIVERSITÀ DI PISA
DIPARTIMENTO DI INFORMATICA

TECHNICAL REPORT

QoS-aware Deployment of IoT Applications Through the Fog

Antonio Brogi, Stefano Forti
Department of Computer Science
University of Pisa, Italy

November 30, 2016

ADDRESS: Largo B. Pontecorvo 3, 56127 Pisa, Italy. TEL: +39 050 2212700 FAX: +39 050 2212726

QoS-aware Deployment of IoT Applications Through the Fog

Antonio Brogi, Stefano Forti
Department of Computer Science
University of Pisa, Italy

Abstract—**Fog computing aims at extending the Cloud by bringing computational power, storage and communication capabilities to the edge of the network, in support of the IoT. Segmentation, distribution and adaptive deployment of functionalities over the continuum from Things to Cloud are challenging tasks, due to the intrinsic heterogeneity, hierarchical structure and very large scale infrastructure they will have to exploit.**

In this paper we propose a simple, yet general, model to support the QoS-aware deployment of multi-component IoT applications over Fog infrastructures. The model describes operational systemic qualities of the available infrastructure (latency and bandwidth), interactions among software components and Things, and business policies. Algorithms to determine eligible deployment plans for an application over a Fog infrastructure are presented. A Java tool, FogTorch, based on the proposed model has been prototyped.

Index Terms—**Fog computing, IoT, QoS-aware deployment.**

I. Introduction

CONNECTED devices are changing the way we live and work. In the next years, the IoT is expected to bring more and more intelligence around us, being embedded in or interacting with the objects that we use everyday. By 2020, CISCO expects 50 billion of connected devices [1] with an average of almost 7 per person. Self-driving cars, autonomous domotics systems, energy production plants, agricultural lands, supermarkets, healthcare, schools exploit Things that are integral part of the Internet and of our existence without us being aware of them.

As a consequence, enormous amounts of data – the so called *Big Data* [2] – are collected by IoT sensors to be stored in Cloud data-centres [3]. There, they are subsequently analysed to determine reactions to events or to extract analytics or statistics. Whilst data-processing speeds have increased rapidly, bandwidth to carry data to and from datacentres has not increased equally fast [4]. On one hand, supporting the transfer of data from/to billions of IoT devices is becoming hard to accomplish in the IoT+Cloud scenario due to the volume and geo-distribution of those devices. On the other hand, the need to reduce latency, to eliminate mandatory connectivity requirements, and to support computation or storage closer to where data is generated 24/7, is evident [5]. The time has come to extend the Cloud all through to the IoT, so as to virtualise and exploit a new hierarchy of resources from the core towards the edge of the network, where data can be used for prompter decision making and support [6].

Recent research efforts are investigating how to better exploit capabilities at the edge of the Internet to support the IoT and its needs. Computational nodes closer to the edge will act both as *filters* – reducing the amount of data sent to the Cloud – and as *processing capabilities* – producing analytics – closer to where data is being sensed or used. Fog (or Edge) computing [5] precisely aims at exploiting a large number of highly distributed edge nodes (e.g., mobile devices, routers, micro-datacentres), to selectively support time-sensitive, geo-distributed or mobile applications, where IoT sensors and actuators are used in hundreds of different cyber-physical processing contexts and services. Fog configures as a powerful enabling complement to the IoT+Cloud scenario, featuring a new layer of cooperating devices that can run services and complete specific business missions, independently from and contiguously with existing Cloud systems.

One of the problems raised by the aforementioned scenario is how to master the complexity of deploying applications over the Fog, mainly due to the scale and heterogeneity of the Fog infrastructure. While some functions are naturally suited to the Cloud layer (e.g., service backends) and others are naturally suited to the Fog layer (e.g., industrial control loops), other functions (e.g., medium term analytics) may be better dynamically assigned to different nodes depending on QoS attributes of the infrastructure. Freeing developers from having to segment the functionalities of their applications over the continuum from Cloud to IoT is crucial to achieve scalability and extensibility, hence for the success of the Fog/Edge paradigm [4]. Concrete questions like:

- *"How many and how powerful Fog nodes should I (buy and) install to adequately deploy my application?"*,
- *"Should I deploy this component to the Cloud, to the new Fog-as-a-Service (FaaS) opened in my city, or on my premises gateway?", or*
- *"Is there any component I should deploy on a different node after this link failure?"*

may be hard to answer promptly even for simple applications. Early efforts to approach these problems have been tuned manually [7] or have considered only tree-like network topologies [8].

Fog computing should support adaptive deployment over the entire available infrastructure, dynamically taking into account both the *application requirements* and the *current state of the infrastructure* for what concerns hardware and software capabilities, link bandwidths and latencies and fault events [9]. The availability of a suitable model of Fog infrastructures and applications is thus crucial to achieve QoS-aware deployments that are expected to fluidly span various federated providers over the continuum from Cloud to Things [5]. New methods, techniques and tools are to be devised so as to distribute application functionalities vertically and horizontally over the available nodes.

> *The goal of this work is to propose a general and extensible model to support QoS-aware deployment of IoT applications over Fog infrastructures.*

The model we propose describes, at a suitably abstract level, characteristics of interest and operational systemic qualities of Fog facilities and of the IoT applications to be deployed. In that, the model can be exploited to determine eligible deployments (if any) of an application over a given, intrinsically hierarchical, Fog infrastructure. As we will see, tools based on the proposed model can be fruitfully applied at design time, at deployment time, and at run time, supporting the whole application lifecycle.

The rest of this paper is organised as follows. Section II describes a motivating example of deployment of a simple IoT application. Section III describes our modelling of Fog infrastructures, IoT applications and eligible deployments considering IoT devices and QoS constraints. Section IV describes the offline multi-constrained algorithms to find eligible deployments of an application over an infrastructure in a context-aware manner and introduces our tool FogTorch. Section V illustrates applicability of the model and of FogTorch over the example of Section II. Related work is discussed in Section VI, while some concluding remarks are drawn in Section VII.

II. Motivating Example

Consider a simple fire alarm IoT application offered by an insurance company to its customers. The application is made out of three components, as illustrated in Figure 1:

- γ_0. a Fire Manager, monitoring the environment to start extinguishing a fire as soon as it is detected,
- γ_1. an Insights Backend, for visualisation of collected data and manual system tweaking, and
- γ_2. a Machine Learning Engine, managed by the insurance company, to be deployed to the Cloud for historical data storage and fire detection model updates.

The average RAM consumption of all components is shown on the left of each of them in Figure 1. On the right hand-side, the list of software capabilities needed by each component is shown. Components interact through the depicted links that must meet the associated QoS constraints in terms of latency and uplink/downlink bandwidth. Also, to promptly manage fire emergencies, component γ_0 must reach out both a fire sensor and an extinguisher actuator, and this should happen within 10 milliseconds from where γ_0 is deployed to where the sensor and the actuator are installed. Note that Fog or Cloud nodes are expected to be able to remotely access Things at neighbouring nodes, through APIs offered by the Fog middleware layer [5].

Figure 2 sketches the Fog infrastructure available to a company that wants to deploy the fire alarm application to protect a warehouse at their premises. The company IT division has installed three Fog nodes (fog_1,fog_2 and fog_3) and selected two candidate data-centres (cloud_1, cloud_2) for deployment purposes.

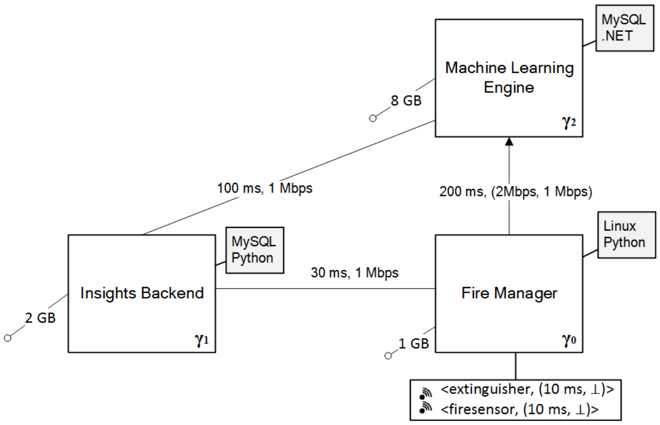


Fig. 1: The multi-component software of the example. Arrows on the asymmetric links (i.e., uplink and downlink differ) indicate the upload direction.

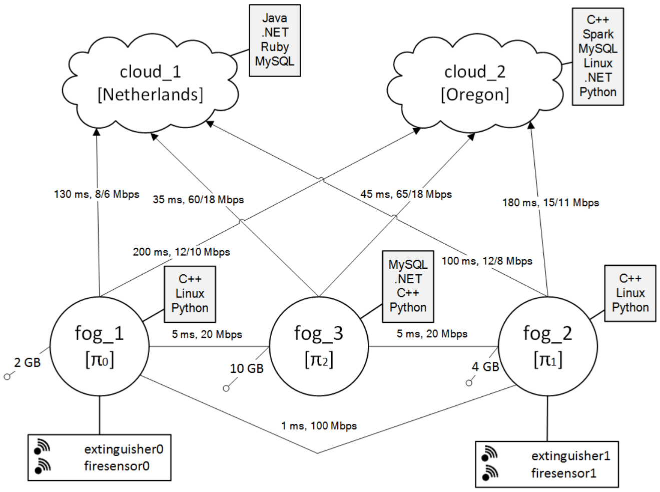


Fig. 2: The available Fog infrastructure.

Software capabilities of each node are listed in the box on the right hand-side and RAM offerings on the left, as in Figure 1. Node fog_2 is installed in the company warehouse and directly connects to the fire sensor (firesensor1) and to the extinguisher (extinguisher1) that γ_0 should exploit at run time. Average latency and bandwidth of the available communication links are reported over the links themselves.

The problem the IT division should solve is how to deploy the three components in such a way all specified non-functional constraints (software, hardware, software interactions, and remote access to IoT) can be met. Even for this simple example, to find an eligible mapping from software components to Fog or Cloud nodes, IT experts at the company would have to evaluate up to 50 candidate deployments. This is because more than one component can be deployed to the same node (based upon the available resources): γ_0 and γ_1 could be deployed on any Fog or Cloud node, and γ_2 could go on either cloud_1 or cloud_2. Determining eligible deployments becomes humanly infeasible as the infrastructure and the number of application components grow, having worst-case exponential complexity (the considered problem is NP-hard).

As we will show in the next sections, and illustrate over this motivating example in Section V, the model we propose permits a description of infrastructures and of IoT applications that can be exploited to algorithmically determine deployments compliant to the desired QoS constraints.

III. Modelling the Fog

A Fog infrastructure consists of IoT devices, one or more layers of Fog computing nodes, and at least one Cloud data-centre. In sections to follow we will formally define the concepts of QoS profiles, Fog infrastructures, IoT applications and eligible deployments.

A. QoS Profiles

Among the possible QoS metrics, we consider only *latency* and *bandwidth* since, whilst *loss* and *jitter* can be remedied through retransmission and buffering respectively, nothing can be done to tame the former two at run time. Since Fog computing will exploit wireless access to the Internet and will bring computation at or nearer the end user, it is realistic to model asymmetric link bandwidth (viz., uplink $\neq$ downlink).

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 $\perp$.*

B. Fog Infrastructures

A Fog infrastructure includes IoT devices, Fog nodes and Cloud data-centres.

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,*
- *$L \subseteq \{\langle n, n', q \rangle | (n, n') \in (F \times F) \cup (F \times C) \cup (C \times F) \cup (C \times C) \wedge q \in Q\}$ is a set of available Fog-to-Fog, Fog-to-Cloud and Cloud-to-Cloud* communication links[1]*, each associated to its QoS profile.*

Some observations are now due to justify the choices made in Definition 2. Firstly, all the elements included in a Fog infrastructure are characterised by their current geographical location[2], assuming that is known at some level of detail. On one hand we assume that sensors/actuators and Fog nodes are provided with geo-spatial location technologies such as GPS. On the other hand Cloud providers usually disclose to customers the geographical area of their data-centres. Identifiers for the modelled entities help managing the case in which more than one entity resides in the same location.

Secondly, we abstract from the type of connection technologies employed both at the Wireless Sensor Network (Bluetooth, Zigbee, RFID, etc.) and at the Access Network (xDSL, FttX, 5G, etc.) levels since our focus is on the QoS a given communication link between two endpoints can offer. As a consequence, all available links are modelled uniformly in Θ and L. Things in Θ directly connected to a Fog node – either via wired or wireless connection – are assumed to have negligible latency and infinite bandwidth. Since security issues are considered orthogonal to (and outside the scope of) this work, we assume that Fog and Cloud nodes can reach the sensors and actuators of all their neighbour nodes through some middleware APIs, experiencing the QoS of the associated links in L.

Thirdly, the model does not deliberatly bind to any particular standard for hardware and software capabilities specification. Realistically, hardware specification will include both *consumable* (e.g., RAM, storage) and *non-consumable* resources (e.g., architecture, CPU cores), whereas software capabilities may concern the available OSs (Linux, Windows, etc.) and the installed platforms or frameworks (.NET, JDK, Hadoop MapReduce, etc.). The choice of how to specify hardware and software capabilities – e.g., with TOSCA YAML [10] like in the SeaClouds project [11], or with other formalisms – is however not bound in the model.

Finally, Cloud computing is modelled according to the hypothesis that it can offer a virtually unlimited amount of hardware capabilities to its customers. This simplification permits the description of any among SaaS, PaaS and Iaas providers, and eliminates the need to describe any particular commercial offering. Overall, when compared to Fog nodes capabilities, it is true that a Cloud customer can always add processing power and storage by purchasing extra VM instances, *as if* they were unbounded.

C. Applications

Modern large scale applications are not monolithic anymore [12]. Therefore, an application running over a Fog computing infrastructure can be thought as a set of independently deployable components that are working together and must meet some QoS constraints.

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.*

[1] 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 $\langle n, n', \langle \ell, b_\downarrow, b_\uparrow \rangle \rangle \in L$ and $\langle n', n, \langle \ell', b'_\downarrow, b'_\uparrow \rangle \rangle \in L$ then $\ell = \ell'$, $b_\downarrow = b'_\uparrow$ and $b_\uparrow = b'_\downarrow$.

[2] For instance, if GPS coordinates are used to model the location of Things, Fog nodes and Cloud datacentres, then $\pi = \langle \pi_{lat}, \pi_{lon} \rangle$.

- $\Lambda \subseteq \{\langle \gamma, \gamma', q\rangle | (\gamma, \gamma') \in (\Gamma \times \Gamma) \wedge q \in Q\}$ *denotes the existing* interactions among components[3] *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 the component needs to reach with QoS profile* q *so to work properly.*

The modelling of applications comprises QoS profiles for the interactions between components to express the desired operational systemic qualities, together with hardware, software and Things requirements for a component to work properly[4].

We now formalise the notion of compatibility of a software component with a node of a Fog infrastructure, be it a Fog node or a Cloud data-centre. Fog nodes must offer the needed software and non-consumable hardware capabilities, and enough consumable hardware to support *at least* that component. Compatibility of Cloud data-centres only requires the needed software capabilities to be available for deployment.

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 component* $\langle i, \overline{\mathcal{H}}, \overline{\Sigma}\rangle \in \Gamma$ *is* compatible *with a node* $n \in F \cup C$ *if and only if:*

- *if* $n = \langle j, \pi, \mathcal{H}, \Sigma, \Theta\rangle \in F$*, then* $\overline{\Sigma} \sqsubseteq \Sigma$ *and* $\overline{\mathcal{H}} \preceq \mathcal{H}$*, or*
- *if* $n = \langle j, \pi, \Sigma\rangle \in C$*, then* $\overline{\Sigma} \sqsubseteq \Sigma$*.*

The relations $\sqsubseteq$ and $\preceq$ read as *is satisfied by*. $\overline{\Sigma} \sqsubseteq \Sigma$ when software offerings Σ at a node fulfil all software requirements $\overline{\Sigma}$ of a component. $\overline{\mathcal{H}} \preceq \mathcal{H}$ when non-consumable and consumable hardware resources $\mathcal{H}$ at a node are enough to support a given component requirements $\overline{\mathcal{H}}$.

D. Deployments

We now formalise the notion of deployment of an application over an infrastructure. Since an IoT application deployment is much related to the Things it should manage, the deployment designer, given an application $A = \langle \Gamma, \Lambda, \overline{\Theta}\rangle$, should be able to define the Things the application will rely upon, once deployed, i.e. she should be allowed to specify bindings between Things requests in $\overline{\Theta}$ and actual Things in T. This requires that one exactly controls the Things that the application will exploit, what is essential for the whole deployment to work, sensing from and actuating upon the correct Things.

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* Things binding $\vartheta : \overline{\Theta} \to T$ *for* A *over* I *is a mapping from each Thing request onto a specific Thing.*

Additionally, the deployment of multi-scale software systems may depend on legal or commercial *business policies*.

Definition 6. *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*[5] *software component of* A *onto the set of nodes where its deployment is permitted (or has been already performed).*

Function δ specifies the nodes where a certain component can be (or is already) deployed, according to current business policies. It implements a *whitelisting* strategy that is safer to avoid exploiting undesired computational capabilities for deployment purposes.

The following definition sets all constraints that an eligible deployment must meet. Condition (1) guarantees that the business policies of A are met and checks hardware and software compatibility of each component with the node onto which it will be deployed, as per Definition 4. Condition (2) checks those hardware capabilities that are consumed when installing more than one component onto the same Fog node (e.g. RAM, storage) so that components deployed on a single node cannot exceed its hardware capacity. Condition (3) ensures that Thing requests of each component are satisfied. Note that a component $\gamma \in \Gamma$ deployed on $n \in F \cup C$ can reach Things directly (when n directly connects to them) or remotely access them (when n reaches a Fog node m that directly connects to them). Both situations are taken into account by our model. Condition (4) ensures that latency offered by a link is not greater th required one, and bandwidth consumed by components interactions and by remote Things access over the same link does not exceed the link capacity. This concerns interactions in Λ as well as remote Things access.

Definition 7. *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*[6]*, for each* $f = \langle i, \pi, \mathcal{H}, \Sigma, \Theta\rangle \in F$: $\displaystyle\sum_{\langle j, \overline{\mathcal{H}}, \overline{\Sigma}\rangle \in \Gamma_f} \overline{\mathcal{H}} \preceq \mathcal{H}$,
3) *for each Thing request* $\langle \gamma, \tau, q\rangle \in \overline{\Theta}$ *such that* $\vartheta(\langle \gamma, \tau, q\rangle) = t$*, there exists* $f = \langle i, \pi, \mathcal{H}, \Sigma, \Theta\rangle \in F$ *such that* $t \in \Theta$ *and* $\Delta(\gamma) = f$ *or* $\langle \Delta(\gamma), f, q'\rangle \in L$.
4) *let* $Q_{(m,n)}$ *be*[7] *the multi-set of QoS profiles associated with component-component and component-Thing interactions that are mapped on the communication link between* m *and* n*. Then*[8]*, for each* $\langle m, n, \langle \ell, b\rangle\rangle \in L$*:* $\langle \overline{\ell}, \overline{b}\rangle \in Q_{(m,n)} \implies \ell \leq \overline{\ell} \quad \wedge \quad b \geqslant \displaystyle\sum_{\langle \overline{\ell}, \overline{b}\rangle \in Q_{(m,n)}} \overline{b}$.

[3] 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 $b_\uparrow = b'_\downarrow$.

[4] For the sake of simplicity, we assume that one component can only require one Thing of each type. Such constraint can be relaxed by simply defining $\overline{\Theta}$ as a multiset.

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

[6] Abusing notation, $\sum$ is used to sum the *consumable hardware* requirements, and $\preceq$ to compare them with available offerings.

[7] Formally: $Q_{(m,n)} =$

$$\{\!\!\{ \langle \ell, b\rangle \mid \langle \gamma, \gamma', \langle \ell, b\rangle\rangle \in \Lambda \ \wedge \ \Delta(\gamma) = m \ \wedge \ \Delta(\gamma') = n \}\!\!\} \cup$$
$$\{\!\!\{ \langle \ell, b\rangle \mid \ \langle \gamma, \tau, \langle \ell, b\rangle\rangle \in \overline{\Theta} \ \wedge \ \Delta(\gamma) = m \ \wedge \ \vartheta(\langle \gamma, \tau, \langle \ell, b\rangle\rangle) = t$$
$$\wedge \ n = \langle i, \pi, \mathcal{H}, \Sigma, \Theta\rangle \in F \ \wedge \ t \in \Theta \}\!\!\}.$$

[8] Abusing notation, $\sum$ and $\leqslant$ are used to sum and compare uplink and downlink bandwidths.

```
procedure FindDeployment($A$, $I$, $\delta, \vartheta$)
    $K \leftarrow$ Preprocess($A$, $I$, $\langle\delta, \vartheta\rangle$)
    if K = failure then
        return failure
    end if
    return BacktrackSearch($\emptyset$, $A$, $I$, $K$, $\vartheta$)
end procedure
```

Fig. 3: Pseudocode of the proposed solution. Given an application A, a Fog infrastructure I, a deployment policy δ, and a Things binding ϑ, it returns an eligible deployment Δ for A over I that complies with δ and ϑ, or a failure.

IV. Finding Eligible Deployments

Given an application $A = \langle\Gamma, \Lambda, \overline{\Theta}\rangle$ and a Fog infrastructure $I = \langle T, F, C, L\rangle$, as defined before, finding one or more eligible deployments $\Delta : \Gamma \rightarrow F \cup C$ is a *decidable* problem that requires a *worst-case* search among $O(N^{|\Gamma|})$ candidates, where $N = |F \cup C|$. Our problem can be proved NP-hard by reduction from the Subgraph Isomorphism problem.

A. Algorithms

Algorithms to determine eligible deployments over Fog infrastructures should manage latencies, uplink and downlink bandwidths, constraints over the Things that a component uses, business policies, context awareness, resource allocation, the possibility to deploy more than one component on a single computational node and more than one interaction onto a single communication link. The algorithms hereby proposed address all these aspects, adaptively selecting *where* a component is to be deployed within the continuum from Cloud to Things. In what follows we will present:

- a *preprocessing* procedure that *a priori* reduces the search space for eligible deployments, and
- a *backtracking* procedure and *heuristics*, to determine a single eligible deployment.

Figure 3 shows how the preprocessing and the bactracking search phase combine in order to find an eligible deployment. Our approach is similar to the SIP solution in [13]. Overall, our approach requires polynomial space in the dimension of the input and worst-case $O(N^{|\Gamma|})$ time, with $N = |F \cup C|$.

B. Preprocessing

The preprocessing procedure scans the input to reduce the search space by determining, for each software component $\gamma \in \Gamma$, the set $K_\gamma \subseteq (F \cup C) \cap \delta(\gamma)$ of compatible Fog nodes and Cloud data-centres, i.e. nodes that satisfy the conditions of Definition 4 on software and hardware requirements and comply with $\delta(\gamma)$ (condition (1) of Definition 7). Additionally, nodes in K_γ satisfy condition (3) of Definition 7 by fulfilling all Things requests associated to component γ in $\overline{\Theta}$. The procedure returns a key-value map K, indexed by component identifiers, such that $K[\gamma] = K_\gamma$, or a failure when even a single component has no candidate nodes for deployment. The latter case makes the whole search for a solution fail immediately, without further searching.

The preprocessing procedure completes in $\Theta(N|\Gamma|)$ that, assuming $|\Gamma| \ll N$, becomes $O(N)$ with $N = |F \cup C|$.

```
procedure BacktrackSearch($\Delta$, $A$, $I$, $K$, $\vartheta$)
    if isComplete($\Delta$) then
        return $\Delta$
    end if
    $\gamma \leftarrow$ selectUndeployedComponent($\Delta$, $A$);
    for all $n \in$ selectDeploymentNode($K[\gamma]$, $A$) do
        if isEligible($\Delta$, $\gamma$, $n$, $A$, $I$, $\vartheta$) then
            deploy($\Delta$, $\gamma$, $n$, $A$, $I$, $\vartheta$)
            result $\leftarrow$ BacktrackSearch($\Delta$, $A$, $I$, $K$, $\vartheta$)
            if result $\neq$ failure then
                return result
            end if
            undeploy($\Delta$, $\gamma$, $n$, $I$, $A$, $\vartheta$)
        end if
    end for
    return failure
end procedure

procedure isEligible($\Delta$, $\gamma$, $n$, $I$, $A$, $\vartheta$)
    return checkHardware($\gamma$, $n$)
            $\wedge$ checkLinks($\Delta$, $\gamma$, $n$, $I$, $A$, $\vartheta$)
end procedure
```

Fig. 4: Pseudocode for the backtracking search. It returns an eligible deployment Δ for A over I, or a failure.

C. Search

Bactracking works on the output of preprocessing, as listed in Figure 4. The algorithm computes nodes and links mapping at the same time. At each recursive call, BacktrackSearch(Δ, A, I, K, ϑ) firstly checks whether a deployment has been found (isComplete(Δ)). If not, it selects a component γ among the undeployed ones (selectUndeployedComponent(Δ, A)) and it attempts deployment on compatible nodes in $K[\gamma]$ one by one (selectDeploymentNode($K[\gamma]$, A)). The isEligible(Δ, γ, n, I, A, ϑ) procedure guarantees that if a deployment for all components is found then it is also an eligible one. The method checks if the action of deploying a certain software component γ on a given node $n \in K[\gamma]$ is applicable to the current partial deployment state. Particularly, for the considered partial deployment:

1) checkHardware(γ, n) returns true if and only if consumable hardware at each Fog node is not exceeded by requests mapped onto it (condition (2) of Definition 7),
2) checkLinks(Δ, γ, n, I, A, ϑ) returns true if and only if latency requirements are met and bandwidth capacity of each link is not exceeded by interactions mapped onto it (condition (4) of Definition 7).

Whenever one between (1) and (2) is not satisfied, the current branch of the search is pruned since isEligible(Δ, γ, n, A, I, ϑ) returns false. Otherwise, procedure deploy(Δ, γ, n, A, I, ϑ) is responsible for adding the association between γ and n to the partial deployment Δ and to update the current state of I by decrementing the consumable hardware available at n of the quantity required by γ and the bandwidth of the link that will support the interactions between γ and any deployed component γ' (also for remote Things). Function undeploy(Δ, γ, n, A, I, ϑ) performs the dual operation of deploy(Δ, γ, n, A, I, ϑ) in case of backtracking.

D. Greedy Behaviour

The heuristic adopted in SELECTUNDEPLOYEDCOMPONENT(Δ, A) is *fail-first* and always chooses the undeployed component γ that has fewer compatible nodes in $K[\gamma]$. It automatically selects those components that have more requirements in terms of Things or hardware and ensures they are deployed on the right nodes. Conversely, the heuristic used in SELECTDEPLOYMENTNODE($K[\gamma]$, A) is *fail-last* and picks candidate nodes, sorting them by: (1) decreasing number of Things required by γ that are directly connected to each node, (2) decreasing hardware capabilities that are offered at each node (e.g., orderly evaluating the available RAM, the storage capability and the number of CPU cores), considering Cloud data-centres as providing infinite hardware.

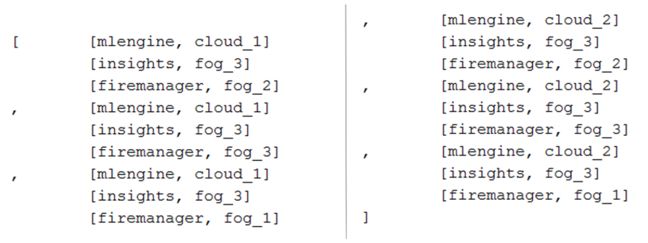
These heuristics guarantee that Things requests are satisfied exploiting the best node possible in terms of spatial proximity and they always try to deploy a component on the most powerful node that can support it. Our approach adheres to the assumption that more powerful nodes are usually also more *reliable* and require less "work" from final users to keep it operational. A good Cloud provider should be committed to transparently offer the best performance possible to its customers in terms of availability, a Fog node installed at the customers premises is more likely to be subject to breakdowns, a mobile phone may run out of battery whilst running some tasks.

E. FogTorch Prototype

We prototyped a proof-of-concept Java tool, named FogTorch[9], that implements the proposed model and algorithms with the purpose of demonstrating their technical feasibility. FogTorch inputs the specification of an infrastructure and of an application to be deployed, along with the related Things binding and deployment policy, and it outputs eligible deployment plans as per Definition 7 of Section III. The tool has been applied over the motivating example of Section II as we will discuss next.

V. MOTIVATING EXAMPLE (CONTINUED)

Consider again the example introduced in Section II. According to our model, a software component like the `Fire Manager` of Figure 1 is represented by a tuple of the kind $\langle\gamma_0,\ \mathtt{RAM:4GB},\ \{\mathtt{Linux},\mathtt{Python}\}\rangle$, the desired QoS of its interaction with the `ML Engine` component can be represented by the tuple $\langle\gamma_0, \gamma_2, \langle\mathtt{200\ ms}, (\mathtt{2Mbps}, \mathtt{1Mbps})\rangle\rangle$, while its Thing requests (specifying a maximum latency of 10 msecs between the component and the IoT devices) are represented as: $\langle\gamma_0, \mathtt{fire}, \langle\mathtt{10\ ms}, \perp\rangle\rangle$ and $\langle\gamma_0, \mathtt{extinguisher}, \langle\mathtt{10\ ms}, \perp\rangle\rangle$. A Fog node like `fog_1` can be represented by the tuple $\langle\mathtt{fog\_1}, \pi_0,\ \mathtt{RAM:2GB},\ \{\mathtt{C++},\mathtt{Linux},\mathtt{Python}\},\ \{\mathtt{extinguisher0},\mathtt{firesensor0}\}\rangle$, a Cloud datacentre like `cloud_1` can be represented by the tuple $\langle\mathtt{cloud\_1}, \mathtt{Netherlands}, \{\mathtt{Java}, \mathtt{.NET}, \mathtt{Ruby}, \mathtt{MySQL}\}\rangle$,and the average QoS of the communication link between them by $\langle\mathtt{fog\_1}, \mathtt{cloud\_1}, \langle\mathtt{130\ ms}, (\mathtt{8Mbps}, \mathtt{6Mbps})\rangle\rangle$.



```
[       [mlengine, cloud_1]
        [insights, fog_3]
        [firemanager, fog_2]
,       [mlengine, cloud_1]
        [insights, fog_3]
        [firemanager, fog_3]
,       [mlengine, cloud_1]
        [insights, fog_3]
        [firemanager, fog_1]
,       [mlengine, cloud_2]
        [insights, fog_3]
        [firemanager, fog_2]
,       [mlengine, cloud_2]
        [insights, fog_3]
        [firemanager, fog_3]
,       [mlengine, cloud_2]
        [insights, fog_3]
        [firemanager, fog_1]
]
```

Fig. 5: FogTorch output for the example of Section II.

The policy that component γ_2 should be deployed only on Cloud datacentres can be expressed by setting $\delta(\gamma_2) = \{\mathtt{cloud\_1}, \mathtt{cloud\_2}\}$. The bindings of `Fire Manager` Things requests to the actual IoT devices `fire1` and `extinguisher1` at the warehouse is expressed as $\vartheta(\langle\gamma_0, \mathtt{fire}, \langle\mathtt{10\ ms}, \perp\rangle\rangle) = \mathtt{fire1}$ and $\vartheta(\langle\gamma_0, \mathtt{extinguisher}, \langle\mathtt{10\ ms}, \perp\rangle\rangle) = \mathtt{extinguisher1}$.

FogTorch, given the input of the example returns the six eligible deployment plans (out of 50 possible) shown in Figure 5, ordered according to the heuristics mentioned in Section IV, the first of them corresponding to $\Delta(\gamma_0) = \mathtt{fog\_2}$, $\Delta(\gamma_1) = \mathtt{fog\_3}$, and $\Delta(\gamma_2) = \mathtt{cloud\_1}$. To summarise, FogTorch simplifies the IT division task of finding a deployment plan that meets all specified non-functional constraints of the given application.

VI. RELATED WORK

To the best of our knowledge, only three approaches have been proposed so far to specifically model Fog infrastructures and applications. [14] 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 [14], however, deals only with the behaviour of software already deployed over Fog infrastructures. [7] 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, [7] takes into account Fog nodes workload but it does not consider bandwidth, Things requests, and business 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. Finally, [8] 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. [8] 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.

The problem of deploying multi-component applications has been thoroughly studied in the Cloud scenario. Projects like SeaClouds [11], Aeolus [15] or Cloud4SOA [16], for instance, have proposed optimised planning solutions to deploy software systems to different (IaaS or PaaS) Clouds. [17] proposed

[9] Available at https://github.com/di-unipi-socc/FogTorch.

to use OASIS TOSCA [18] to model IoT applications in IoT+Cloud scenarios. Recently, [19] have 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 emerging Fog paradigm, however, introduces new problems, mainly due to its need for connection-awareness and interactions with the IoT, that were not considered by [19].

In the context of IoT deployments, formal modelling approaches have been recently proposed to achieve connectivity and coverage optimisation [20], [21], improved resource exploitation of Wireless Sensors Networks [22], and to estimate reliability and cost of service compositions [23]. Our modelling effort aims at complementing that work, by describing the interactions among software components and IoT devices at a higher level of abstraction to achieve segmentation of applications through the Fog—that, to the best of our knowledge, was not addressed by previous work.

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 [13], [24], [25] and deployment over WAN [26], by performing node and link mapping in a single phase, as we do. Recently [27] has modeled the problem of component deployment through Mixed Integer Linear programming, trying to optimise some metrics. This requires the availability of a cost model, which our model does not feature yet.

VII. Concluding remarks

The availability of suitable models of Fog infrastructures and applications is crucial to succeed in automating QoS-aware deployments over the continuum from Cloud to Things. Unfortunately, the majority of state-of-the-art tools for automated deployment of distributed software do not deal with non-functional properties to achieve eligible deployment plans [12].

As we anticipated in Section I, the model that we have proposed can be exploited:

- At *design time*, to run what-if analyses to perform capacity planning by identifying beforehand possible critical deficiencies in the Edge network capabilities, and to assess resiliency and robustness of the infrastructure to churn and failures.
- At *deployment time*, to automatically determine —with tools like FogTorch—where each application component can be deployed by satisfying the specified QoS constraints,
- At *run time*, by designing new tools to drive the monitoring of deployed applications and to trigger, when needed, reconfiguration processes.

The model and algorithms we have proposed can represent a first step to tackle the problem of finding a representation of Fog systems that enables cooperation among different providers. Indeed, our model takes into account various relevant aspects of the Fog in order to determine QoS-aware deployments of IoT applications:

1) Average *latency* and (consumable) *bandwidth* of communication links.

   Our model abstracts from the communication technologies employed at all layers of the architecture, and focuses only on their QoS. Depending on the scenario and on the state of the network, we envision that Things control-loops, medium-term operational support, and long-term business intelligence tasks of an application may spread all through the Cloud-Fog-IoT system or collapse into a single layer adaptively. Our model includes inter-Cloud and inter-Fog communication, to account for interoperability and federation at all layers [5].

2) *Application requirements* and *infrastructure capabilities*, in terms of hardware and software.

   Our model assumes that a Fog node can be any computational capability along the continuum from Things to Cloud (e.g., users mobile devices, network gateways, micro datacentres, etc.). The model does not bind to any particular solution for what concerns the specification of software/hardware offerings, and it can exploit existing domain-specific languages from the context of Cloud computing (like TOSCA YAML [18] or JSON-based CloudML [28]). The model permits to describe the Fog infrastructure, regardless of whether the latter will exploit only ISP/telco network apparatus or a wider gamut of devices, whilst taming its heterogeneous nature and abstracting from the type of service offered by the involved Cloud providers (SaaS, IaaS, PaaS).

3) *IoT devices* and *business policies* of application components.

   Fog applications will undoubtedly exploit the IoT to manage cyber-physical processes. Some IoT devices will be embedded in Fog nodes (e.g., smart vehicles), some will be fixed (e.g., smart traffic lights) and some other will move together with people carrying them (e.g., life-saving devices). Our model can capture all those cases and enable Fog nodes to access neighbouring IoT devices with the related QoS profile, so to account for federation among providers also for IoT exploitation. As it happens in modern enterprise IT, deployments of applications are decided also on the basis of legal or commercial policies, by specifying on which nodes a component is allowed to run (e.g., a start-up sponsored by a specific Cloud provider may enforce the free use of the data-centres owned by its sponsor).

We conclude by mentioning some directions for future work that we intend to follow:

- *Opportunistic exploitation of IoT devices.* Fog applications will opportunistically work on a certain number of devices within a certain area (e.g., smart cars control systems). We intend to extend the model to take that into account.

- *QoS profiles*. As our immediate future work, we intend to employ probability distributions to represent QoS profiles and to exploit Monte Carlo simulation (as in [29]) to predict the *reliability* of deployment plans. Other QoS metrics which can be considered include reliability of links and nodes, power consumption, security, and monetary costs.
- *Deployment scheduling*. The algorithms we presented adaptively select where to deploy each component by following a *plan first, schedule later* approach which simplistically considers the deployment of components as commutative. Future work in this direction can be devoted to determine appropriate schedulings of deployment operations.
- *Validation* – Future work will obviously have to include experimentations on real case studies for the Fog (currently under development) in order to compare the automatedly computed deployments with those determined by field experts in enterprise IT.

References

[1] CISCO, “Fog computing and the internet of things: Extend the cloud to where the things are,” https://www.cisco.com/c/dam/en_us/solutions/trends/iot/docs/computing-overview.pdf, 2015.

[2] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, “Big data: The next frontier for innovation, competition, and productivity. 2011,” vol. 5, no. 33, p. 222, 2014.

[3] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. U. Khan, “The rise of big data on cloud computing: Review and open research issues,” *Information Systems*, vol. 47, pp. 98–115, 2015.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] OpenFog, “An OpenFog Architecture Overview,” https://www.openfogconsortium.org/page-section/white-papers/, 2015.

[10] “TOSCA Simple Profile in YAML Version 1.0,” http://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.0/TOSCA-Simple-Profile-YAML-v1.0.html, accessed: 25/08/2016.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] A. Brogi, J. Soldani, and P. Wang, “Modelling and analysing cloud application management,” in *ESOCC 2014*, 2014, pp. 171–186.

[19] S. Wang, A. Zhou, F. Yang, and R. N. Chang, “Towards network-aware service composition in the cloud,” *IEEE Transactions on Cloud Computing*, 2016.

[20] J. Yu, Y. Chen, L. Ma, B. Huang, and X. Cheng, “On connected target k-coverage in heterogeneous wireless sensor networks,” *Sensors*, vol. 16, no. 1, p. 104, 2016.

[21] A. B. Altamimi and R. A. Ramadan, “Towards internet of things modeling: a gateway approach,” *Complex Adaptive Systems Modeling*, vol. 4, no. 1, p. 25, 2016.

[22] H. Deng, J. Yu, D. Yu, G. Li, and B. Huang, “Heuristic algorithms for one-slot link scheduling in wireless sensor networks under sinr,” *International Journal of Distributed Sensor Networks*, vol. 11, 2015.

[23] L. Li, Z. Jin, G. Li, L. Zheng, and Q. Wei, “Modeling and analyzing the reliability and cost of service composition in the iot: A probabilistic approach,” in *ICWS 2012*, 2012, pp. 584–591.

[24] Y. Zhu and M. H. Ammar, “Algorithms for assigning substrate network resources to virtual network components.” in *INFOCOM*, vol. 12, 2006.

[25] 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.

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

[27] A. Nazari, D. Thiruvady, A. Aleti, and I. Moser, “A mixed integer linear programming model for reliability optimisation in the component deployment problem,” *Journal of the Operational Research Society [P]*, no. In Press, pp. 1–11, 2016.

[28] A. Bergmayr, A. Rossini, N. Ferry, G. Horn, L. Orue-Echevarria, A. Solberg, and M. Wimmer, “The evolution of CloudML and its manifestations,” in *Proc. of the 3rd International Workshop on Model-Driven Engineering on and for the Cloud (CloudMDE)*, 2015, pp. 1–6.

[29] L. Bartoloni, A. Brogi, and A. Ibrahim, “Probabilistic prediction of the qos of service orchestrations: a truly compositional approach,” in *ICSOC 2014*, 2014, pp. 378–385.

Antonio Brogi is full professor at the Department of Computer Science, University of Pisa (Italy) since 2004. He holds a Ph.D. in Computer Science (1993) from the University of Pisa. His research interests include service-oriented, cloud-based and fog computing, coordination and adaptation of software elements, and formal methods. He has published the results of his research in over 150 papers in international journals and conferences.

Stefano Forti received the M.Sc.(Hons.) degree in computer science and networking (2016) jointly from the University of Pisa (Italy) and the Sant’Anna School of Advanced Studies, Pisa. He is currently Ph.D. student at the Department of Computer Science of the University of Pisa. His research interests include fog, cloud and service-oriented computing, formal methods and algorithms.