Service Placement in Fog Computing Using Constraint Programming

Farah Ait Salaht, Frédéric Desprez, Adrien Lebre, Charles Prud'Homme, Mohamed Abderrahim

Service Placement in Fog Computing Using Constraint Programming

F. Aït-Salaht
Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP,
LYON, France.
farah.ait-salaht@inria.fr

F. Desprez
Univ. Grenoble Alpes
Inria, CNRS, Grenoble INP, LIG
Grenoble, France
Frederic.Desprez@inria.fr

A. Lebre, C. Prud'homme, M. Abderrahim
IMT-Atlantique
Inria, LS2N Nantes, France
firstname.lastname@imt-atlantique.fr

***Abstract*—This paper investigates whether the use of Constraint Programming (CP) could enable the development of a generic and easy-to-upgrade placement service for Fog Computing. Our contribution is a new formulation of the placement problem, an implementation of this model leveraging Choco-solver and an evaluation of its scalability in comparison to recent placement algorithms. To the best of our knowledge, our study is the first one to evaluate the relevance of CP approaches in comparison to heuristic ones in this context. CP interleaves inference and systematic exploration to search for solutions, letting users on what matters: the problem description. Thus, our service placement model not only can be easily enhanced (deployment constraints/objectives) but also shows a competitive tradeoff between resolution times and solutions quality.**

***Index Terms*—Fog Computing, Edge Computing, Services Placement, Constraint Programming, Choco Solver**

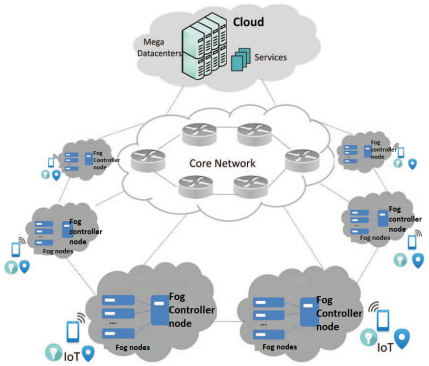


Fig. 1: A Fog Computing architecture as presented in [3].

I. Introduction

Cloud platforms are part of the distributed computing landscape. However, given the massive arrival of IoT devices, together with an exponential number of related applications (smart-metering, smart-transportation, augmented reality, wearable computing, etc.), the deployment of more decentralized infrastructures such as the Fog ones [1] is now crucial. Distributing the infrastructure itself and installing processing resource in different locations, closer to users, is the only way to cope with the requirements in terms of throughput and latency of the aforementioned applications. Processing a data stream as close as possible to where it has been generated, mitigates, for instance, network congestion [2]. Figure 1 gives an overview of a Fog Computing infrastructure: a large number of heterogeneous interconnected devices between clouds and IoT sensors.

While Fog Computing infrastructures will soon allow the deployment of processing/storage resources closer to the edge of the network, there are still open questions related to the resource management aspects. Among them, the service placement problem (i.e. *"how to assign IoT applications to computing nodes which are distributed in a Fog environment?"*) has generated lots of interests in the scientific community [4, 5]. However, all placement service proposals have been designed to solve one particular use case, which strongly limits their adoption into a general resource management system for Fog infrastructures. Although our community has been investigating this challenge for a couple of years [6–11], we claim that proposed approaches are not generic enough to deal with all applications' specifics that Fog infrastructures would have to host. Current algorithms rely mainly or Integer Linear Programming (ILP) or heuristic approaches that are not easily expandable (i.e. cannot easily evolve to integrate new application or infrastructure characteristics, and allow new placement constraints to be easily developed and integrated), or upgradeable in the sense of exploiting any resolution approach that is implied by the user, allowing independent sub-problems to be solved in parallel. In this paper, we investigate whether a model built on the Constraint Programming (CP) [12] can be envisioned to develop a generic and easy-to-upgrade placement service for Fog infrastructures. Besides being faster and more robust than the known ILP approach, the CP code is significantly smaller, which implies it would be easier to implement and maintain. Plus, many open-source yet efficient CP solvers are distributed whereas the ILP uses an external (potentially expensive) commercial ILP solver. In view of that, we propose in this work a new formulation and relatively simple model for the placement problem considering a general definition of service and infrastructure network through graphs using constraint programming. We underline that the contribution of this study is not only related to the model provided but also to its validation in terms of extensibility and scalability. Although the use of CP approaches have been proven efficient in several problem solvers, in particular in Cloud Computing

placement problems (i.e. Entropy VEEE, BTRPlace) [13], CP-solvers are still facing negative preconceived ideas regarding their capacity to deal with large problems. Here, we propose to extend those models to take into account Edge specifics, and show through our evaluation results that the implementation of our model in Choco-solver [14] is competitive w.r.t the most recent solutions of the literature.

The remaining of the paper is organized as follow. Section II presents the related work to placement challenges in Fog Computing platforms and motivates the need for a holistic placement service. Section III describes our model and the way we solve the placement problem using Choco-solver. Section IV discusses experiments we performed to evaluate the efficiency of our placement service. Finally, Section V concludes this paper and gives some perspectives.

II. Related Work and Motivation

The service placement problem in Fog infrastructures has been highly discussed with several solutions based on different application scenarios, network assumptions, and objective functions. Based on our analysis of the state-of-the-art, most of the scenarios undertaken so far can be categorized as follows:

- *Scenario 1: deploy a set of services each abstracted as a single component.* This scenario considers monolithic applications (e.g. data) [6, 10]. In [10], the authors investigate the problem of IoT data placement in a Fog infrastructure. They propose to formulate the problem as a generalized assignment problem by considering the resource capabilities constraints and minimize the overall latency while storing and retrieving the data set. In order to accelerate the solving time, the authors provide a heuristic based on geographical zoning to reduce search space and find out a placement in a reasonable time. In [6], Yousefpour et al. study the dynamic Fog service provisioning problem. The study aims to dynamically provisioning IoT applications on the fog nodes, to comply with the QoS terms (delay threshold, penalties, etc.) with minimal resource cost. The problem is formulated as an integer non linear programming task, and two heuristics are proposed to solve it efficiently.
- *Scenario 2: deploy a set of services each abstracted as a set of interdependent components.* This scenario assumes that each service is pre-partitioned into a set of components (resp. modules), where only components requirements are taken into account. As an example, we cite the work of Skarlat et al. in [8], that propose to study the placement of IoT services on virtualized Fog resources. This work present a placement strategy that maximize the number of service placements to fog resources, while satisfying the QoS constraints. Only resource application components and application execution deadline are specified. The service placement problem is formulated as an ILP and solved by CPLEX. A genetic algorithm in a centralized manner is proposed to reduce and accelerate the execution time of ILP.
- *Scenario 3: deploy a set of services each abstracted as a Directed Acyclic Graph (DAG).* The service in this scenario is assumed to be pre-partitioned into interconnected components (resources and networking requirements are considered) having a DAG topology [7, 9, 11]. The DAG graph models a wide range of realistic IoT applications like healthcare, latency-critical gaming that involve a hierarchical set of processes (or virtual machines), including streaming, multicasting, and data aggregation applications. Among the works considering this scenario, we cite the work of Brogi et al. [7] that propose a model to support the QoS-aware deployment of multi-component IoT applications in Fog infrastructures. The proposed model provides eligible deployments (if any) of an application by performing pre-processing plus backtracking approaches.

As we can remark here and in many other works that try to address the placement problem in Fog environments [4, 5], each solution targets a specific sets of applications topology, network assumptions, and objectives. If targeting a specific use case enables our community to move forward, a more general framework should be designed in order to handle requirements/constraints of the different applications. Our work targets this objective and handle all the aforementioned scenarios. Indeed, we provide in this paper a new formulation of the placement problem considering a general definition of service and infrastructure network through graphs using constraint programming as discussed in the following section.

III. Service deployment problem: system model and problem formulation

In this section, we propose to describe the accurate and the comprehensive framework provided to address the Service Placement Problem (SPP). The framework includes a system model and a constraint satisfaction model, to represent the problem using attributes that are critical for SPP in Fog environments. Before describing the constraints and the optimization problem, we propose to model the infrastructure network, and the Fog applications first.

A. System model

a) Infrastructure model:

Definition 1. *The Fog Computing infrastructure is defined as a directed graph* $G^I =< \mathcal{H}, \mathcal{E} >$*, where* $\mathcal{H}$ *is the set of host nodes (Cloud, Fog, and Things nodes) and* $\mathcal{E}$ *the network connections between the nodes.*

The infrastructure nodes may represent Cloud nodes (data centers), Fog nodes (resources which provide computational power and/or storage. *e.g.,* routers, switches, PCs, etc.), or IoT devices (sensors and actuators), at possibly different hierarchical layers. Each infrastructure node f is characterized by its capacities $U(f)$ such that U=(CPU, RAM, ...). Metrics like: storage, network, energy resources, etc, can also be specified.

Nodes are interconnected via a set of links $\mathcal{E}$, $\mathcal{E} = \mathcal{H} \times \mathcal{H}$. Each link $e_{i,j} \in \mathcal{E}$, is characterized by the latency

$LAT(e_{i,j})$ between source node i and sink node j, where $LAT(e_{i,i}) = 0$, and the bandwidth of the link as $BW(e_{i,j})$, where $BW(e_{i,i}) = \infty$.

b) Service model: The service model considered here encompasses the different scenarios depicted in Section II. Unlike the works found in the literature, which fits only in one of the scenarios mentioned, our work is distinguished by the genericity of the proposed model. Indeed, we consider any type of services topology, as long as it can be represented as a graph. So, let $\mathcal{A}$ be the set of services to be deployed in physical network, the services can be abstracted to application graphs. An application is denoted as a connected graph $G^a =< \mathcal{C}^a, \mathcal{L}^a >$, where $\mathcal{C}^a$ defines the set of all components of application a, and edges $\mathcal{L}^a \subseteq \mathcal{C}^a \times \mathcal{C}^a$ defines the inter-dependencies and communication demands between components. Components and links are respectively defined as in Definition 2, and Definition 3.

Definition 2. *A component $c \in \mathcal{C}^a$ is defined by a set of resource types denoted $u(c)$, such as $u =$ (Reqcpu, Reqram, ...), where $Reqcpu$ (resp. $Reqram$) is the number of cores (resp. the amount of memory) required by the component. Other requirements can be specified. An application component can be a fixed component, i.e., has a specific infrastructure node where it must be deployed. We denote by $\Phi^a \subseteq \mathcal{C}^a$ the set of fixed components and by $\phi^a(c)$ the physical node where the component must be deployed.*

Definition 3. *A link (or pair) $k \in \mathcal{L}^a$ is a couple $k = (source, sink)$, where $source, sink \in \mathcal{C}^a$ are respectively source and sink component of k. For each k, we define $Reqlat(k)$ as the maximum acceptable latency between components, and $Reqbw(k)$ as the bandwidth requirements for k.*

In the rest of the paper, we use interchangeably the term service and application.

c) Deployment: Deployment of a set of applications $\mathcal{A}$ over a Fog infrastructure G^I is a mapping of all the components $\mathscr{C}$ such that $\mathscr{C} = \bigcup_{a \in \mathcal{A}} \mathcal{C}^a$, over the nodes in $\mathcal{H}$ and of all the pairs in $\mathscr{L}$ such that $\mathscr{L} = \bigcup_{a \in \mathcal{A}} \mathcal{L}^a$ onto the physical links in $\mathcal{E}$. The deployment of $\mathscr{C}$ on $\mathcal{H}$ amounts to finding the infrastructure devices that satisfy the constraints on resource capacities, *i.e.,* ensure that the resources of all the deployed components do not exceed the resource capacities of the infrastructures nodes. The mapping of a pair $k \in \mathscr{L}$ is to find a path in the physical graph such that the networking constraints are respected *i.e.,* satisfy the required networking resources (in term of bandwidth, latency...).

Remark 1. *The mapping of a pair $k \in \mathscr{L}$ may form an Hamiltonian path.*

In the next section We define the deployment of components to the infrastructure's nodes and its related constraints.

B. A constraint-based model for SPP (CP-SPP model)

In this section, we describe how to tackle service placement problem in Fog infrastructure using constraint programming approach.

Constraint programming [12] has been widely used in a variety of domains such as production planning, scheduling, timetabling, and product configuration. The basic idea of constraint programming is that user formulates a real-world problem as a CSP (Constraint Satisfaction Problem) and then a general purpose constraint solver calculates solution for it. Formally, a CSP is defined by a triplet $\langle$V, D, C$\rangle$, where V is the set of variables, D is the set of domains associated with the variables, and C is the set of constraints. A constraint $cstr(j)$, associates with a sub-sequence $vars(cstr(j)) \in$ V, defines the allowed combinations of values which satisfy $cstr(j)$. A constraint is equipped with a monotonically decreasing function that removes from the domains of $vars(cstr(j))$ values that cannot satisfy $cstr(j)$. Combining the definition of service placement problem and the basic idea of constraint programming, the problem of applications mapping can be modeled by CSP in the following way:

1) Set of variables and their domains: To translate the component placement problem into a constraint satisfaction problem, we introduce a set of decision variables. We propose to distinguish here the variables related to nodes and those related to arcs respectively. We note that for the needs of the modeling, we increase the G^I by adding a supersink node to which all graph nodes can access (including itself). The supersink denoted by α has an unlimited resources (CPU, RAM...), and the edges that link the infrastructure node to α, and α to itself, have an infinite capacity.

a) Variables related to nodes: For each pair $k \in \mathscr{L}$, we define the following:

- $s = \{s_k | k \in [1, |\mathscr{L}|]\}$. Variable s_k denotes the node that hosts the source component of pair k. s_k takes these values in $\mathcal{H}$.
- $t = \{t_k | k \in [1, |\mathscr{L}|]\}$. t_k denotes the node that hosts the sink component of pair k. The variables t_k takes these values in $\mathcal{H}$.
- $n = \{n_{k,j} | k \in [1, |\mathscr{L}|], j \in [1, |\mathcal{H}|]\}$. $n_{k,j}$ denotes the host node at position j in the path of pair k. The variables $n_{k,j}$ takes these values in $\mathcal{H}$.
- $h = \{h_i | i \in [1, |\mathscr{C}|]\}$. h_i denotes the node that hosts component i, $\forall i \in \mathscr{C}$. h_i takes these values in $\mathcal{H}$.
- $p = \{p_k | k \in [1, |\mathscr{L}|]\}$. Variable p_k denotes the position of t_k in n_k (position of $s_k = 0$). p_k takes these values in $\{1, \dots |\mathcal{H}|\}$.

Remark 2. *Since the only outgoing arc of α is a loop, n always ends with at least one occurrence of α.*

b) Variables related to arcs: For each pair $k \in \mathscr{L}$, we define the following variables:

- $a = \{a_{k,j} | k \in [1, |\mathscr{L}|], j \in [1, |\mathcal{H}|]\}$. $a_{k,j}$ denotes the arc between $n_{k,j}$ and $n_{k,j+1}$ in path of pair k. The variables $a_{k,j}$ takes these values in $\mathcal{E}$.
- $b = \{b_{k,j} | k \in [1, |\mathscr{L}|], j \in [1, |\mathcal{H}|]\}$. $b_{k,j}$ is the bandwidth on arc $a_{k,j}$.
- $l = \{l_{k,j} | k \in [1, |\mathscr{L}|], j \in [1, |\mathcal{H}|]\}$. $l_{k,j}$ is the latency on arc $a_{k,j}$.

Remark 3. *From Remark 2, the only outgoing arc from t_k is the one to α.*

2) Set of constraints: Next we look at the relevant constraints of the problem. We introduce two types of constraints: constraints related to nodes, and constraints related to arcs.

a) Constraints on nodes:

- **C1**. BINPACKING constraints [15] on resource capacities of infrastructure nodes. These constraints are considered for physical network nodes and are related to computational and memory limitations respectively. In a simplified way, we can say that a binpacking constraint ensures that for all physical nodes, the sum of all mapped applications components demand not exceed their maximum available capacities, with adding transversal filtering.

$$\text{BINPACKING}(\langle h, Reqcpu\rangle, CPU),$$

$$\text{BINPACKING}(\langle h, Reqram\rangle, RAM).$$

- Links between variables h_k, $n_{k,j}$ and p_k
  - **C2**. The node at position 0 in the path n_k is the node that hosts the source component of pair k:

$$n_{k,0} = s_k, \quad \forall k \in \mathscr{L}$$

  - **C3**. The item at position p_k in path n_k is equal to t_k: $t_k = n_{k,p_k}, \quad \forall k \in \mathscr{L}$.
  - **C4**. Position of t_k if source and sink component of a pair are the same: $s_k = t_k \iff p_k = 1$.
  - **C5**. Avoid cycles in n_k (all items of n_k are different [16] except if $s_k = t_k$):

$$\text{ALLDIFFERENT}(n_{k,j},\ \forall j = \{1, \ldots, |\mathcal{H}|\}),\ \forall k \in \mathscr{L}$$

  - **C6**. Make sure that a path n_k ends with at least one occurrence of α (see Remark 2). REGULAR [17] constraint ensures that values assigned to sequence of variables belong to a given regular language, here expressed by a regular expression :

$$\text{REGULAR}(n_k, \text{“}[\hat{}\alpha] + [\alpha] + \text{”}), \quad \forall k \in \mathscr{L}$$

  - **C7**. Contiguous pairs: $t_k = s_{k'}$, for all two pairs k and k' sharing a same application component
- **C8**. Locality constraint: $h_i = \phi^a(i), \ \forall i \in \Phi^a, \forall a \in \mathcal{A}$. Restrict the fixed component to the specified locality (*i.e.,* to a given infrastructure node).

b) Constraints on arcs:

- **C9**. Respect bandwidth limit of each arcs:

$$\text{BINPACKING}(\langle b, Reqbw\rangle, BW)$$

  This constraint ensures that for each physical link, the bandwidth demands of all applications must not exceed maximum available bandwidth. This constraint is encoded as a BINPACKING constraint.
- **C10**. Satisfy latencies, per pair:

$$l_{k,j} = LAT(a_{k,j}), \quad \forall k \in \mathscr{C},\ \forall j \in \{0, \ldots, |n_k|\}$$

$$\sum_{j=0}^{|n_k|} l_{k,j} \leq Reqlat(k), \quad \forall k \in \mathscr{C}$$

  This constraint ensures that the end-to-end latency along a path of pair k does not exceed the maximum acceptable latency between the component of the pair.

C. Problem Optimization

An optimal solution to the CSP problem can be defined as: given an objective function F, where the optimal solution is achieved when F is minimized (resp. maximized), corresponds to find a mapping $\mathcal{M}^\star \in \mathcal{M}$, such that $\forall \mathcal{M}, \quad F(\mathcal{M}^\star) \leq F(\mathcal{M})$ (resp. $F(\mathcal{M}^\star) \geq F(\mathcal{M})$).

We notice that the expressive modeling power provided by CP enables the development of compact and "natural" models whereas the domain reduction by constraint propagation ensures the determination of the globally optimal solution. Additional features like the efficient determination of the multiple solutions and near best solutions makes CP an attractive optimization technique [18].

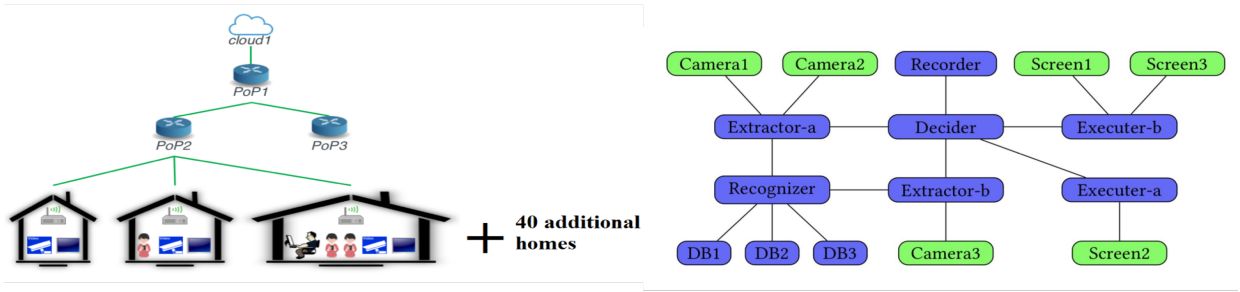
D. Search strategy

A key feature of constraint programming is the ability to design specific search strategies to solve problems. In this work a very basic enumeration strategy is applied on decision variables s, t and n. The next variable to branch on is selected in a sequential way (input order) and assigns to its current lower bound.

E. Implementation of Constraint Programming-Based Service Placement Algorithm (CP-SPP)

After formulating the service placement problem as CSP, we now choose a constraint solver to solve the problem. Constraint programming is often realized in imperative programming by software library, such as CPLEX CP Optimizer [19], JaCoP [20] and Choco [14]. We choose Choco as our constraint solver (many times awarded at international solver competitions MiniZinc[1]). Choco is a free Open-Source Java library dedicated to Constraint Programming. It aims at describing real combinatorial problems in the form of constraint satisfaction problems and solving them with constraint programming techniques by alternating constraint filtering algorithms with a search mechanism. The CP-SPP algorithm takes as inputs the infrastructure network and the set of applications graphs to deploy, and output the set of all feasible mapping. We can set an optimization function and timeout for this algorithm to respectively compute an optimal solution and reduce the search time of constraint solver. In this case, constraint solver will return local or global optimal solution until timeout expired. The performance of the elaborated algorithm is described in the next section.

[1]https://www.minizinc.org/challenge.html



(a) (b)

Fig. 2: Motivating example based on the experiments depicted in [9]. (a) Fog infrastructures, and (b) Smart Bell application.

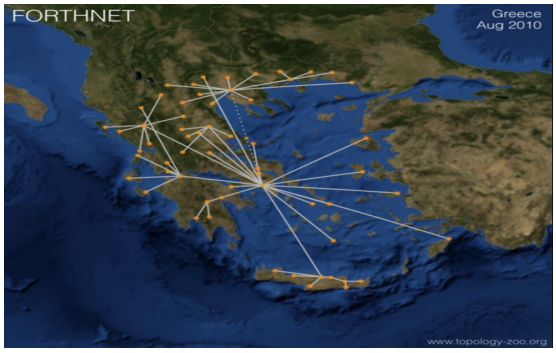


Fig. 3: Greek Forthnet topology.

IV. Experimental Evaluation

In this section, we evaluate the performance of the provided CP-SPP model built and solved using Choco solver [14]. The experiments presented here focus essentially on the computing times of our approach (how long it takes to solve the placement algorithm) and its scalability. We first provide an overview of our experimental setup before evaluating the benefits in solving time of our framework.

A. Experimental Settings

a) Experimental Setting 1: For the first experiment, the motivating example is based on the Fog infrastructure and Smart Bell application introduced in [9] and illustrated in Figure 2. The infrastructure contains a Cloud server, three network Points of Presence (PoP) in the edge layer and a number of end devices in forty-three homes in the extreme edge layer. End devices in each home has a box and a PC / mobile. The global infrastructure is composed by 91 fog nodes and 86 sensors (camera and screen). The considered application in this example is the Smart Bell application. It notifies home inhabitants when they get a visitor. It serves three homes in a same neighborhood. The main components of the application are: i) Extractor that extracts human faces in captured images, ii) DB that stores inhabitants' and friends' information, iii) Recognizer that try to recognize visitors, iv) Decider that makes reaction decisions for each visitor, v) Executor that generate and send commands to inform inhabitants through screens, and vi) Recorder that stores strangers' information and counting how many times a stranger appears.

The considered capacities of each fog node and links as well as the requirements of each component and each application binding are similar to those given in [9].

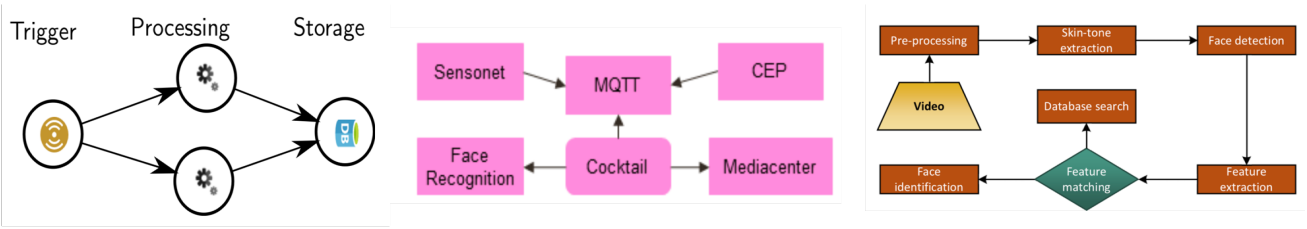
b) Experimental Setting 2: For the second part of our evaluations, we consider an infrastructure network selected from the Internet Topology Zoo[2] as shown in Figure 3. This practical topology allow us to evaluate in an accurate manner the performance of our proposal for a realistic infrastructure. The considered topology is based on the Greek Forthnet topology composed by 60 PoPs and 59 links. Because Fog Computing is a new paradigm which has not been practically deployed in a reasonably large scale, we propose in these experiments to derive larger graph by replicating the network as follows: G^k denotes the graph generated by replicating $k-1$ time the Forthnet topology. To ensure the connectivity of the generated graph, we randomly connect two nodes of the graph (n) and graph $(n-1)$, $\forall n = \{1, \ldots, k\}$. Here, we consider respectively the following graphs: G^2 (with 120 nodes), G^5 (300 nodes), G^{10} (600 nodes) and G^{20} (1200 nodes).

The processing and the memory capacities for each infrastructure node are chosen randomly from respectively 10 to 100 GHz for CPU and 20 to 200 GB for RAM. For each edge in the network, the bandwidth is chosen randomly to be between 100 and 10000 Mbps, and the latency to be between 1 and 10 ms.

We consider three types of application graphs as shown in Figure 4 to deploy on infrastructure graph. Each of the graphs feature a different number of components. For the first service, we propose to consider an application composed respectively by four components and four edges [11]. This application is characterized by three types of components: i) trigger service which sends tokens. These tokens transport the collected measurements related to the end devices', ii) processing service that emulates the performed application treatment, and iii) storage service that stores the received tokens in memory for further processing (if necessary).

For the second application, we consider the "Smart Bell application" [21] composed by six components and five edges.

[2]http://www.topology-zoo.org/



(a) (b) (c)

Fig. 4: Considered applications. (a) Storage Application [11], (b) Smart Bell application [21], and (c) A face recognition application [22].

Application is based on the following set of software entities: i) a Message Oriented Middleware (MQTT) that permits asynchronous communications and eventing among the software entities, ii) A Complex Event Processor (CEP) that operates as an IoT event hub to aggregate, filter, and trigger IoT events according to business rules, iii) A Mashup Engine called Cocktail, that permits the execution of a graph of actions on actuators and services. iv) An IoT Capillary Router called Sensonet, which enlists heterogeneous connected things and collects data, and v) a Face Recognition enabler.

The third application (face recognition application [22]) is composed by eight components and seven edges. The face detection process starts from the Pre-Processing part that includes the necessary color conversion. Then, the Skin-tone extraction is performed in order to differentiate areas that are face candidates among other objects in the video frame. From that stage, the potential face region is detected and the equivalent facial features are extracted. This intermediate result is fed into the database that finally replies if the face that was detected exists in the database or not.

The applications presented here and considered for our evaluations allow us to capture a variety of application scenarios where Fog Computing is relevant. Regarding the application component requirements, the CPU and RAM, as well as bandwidth and latency are chosen randomly respectively: within the range of 1 to 2 for CPU (GHz), RAM (GB) and bandwidth (Mbps), and from 100 to 1000 ms for latency.

In these experiments, we assume that applications arrive by batch according to a Poisson law, and the placement service policy is called either periodically or each time a new job should be provisioned.

B. Performance Analysis

The goal of our analysis is to investigate the resolution trends of our model based on the following three factors: *number of applications, number of components per application, and infrastructure size*. First, we propose to compare the performance of our model with the algorithms provided in [9]. Next, we push the experiments further by analyzing the model and observing the solving time of CP-SPP under the second experimental setting.

Algorithm	Resolution Time (s)	Quality of the solution
ILP	343	100%
First Fit	265	186.8%
GA	28	143.9%
DAFNO-InitCO-DCO(0.3)	0.003	100.3%
CP-SPP	**0.559**	**100%**

TABLE I: Evaluation Results of Different Placement Algorithms for the Smart Bell application [9].

1) Comparison of the CP-SPP Model with Algorithms provided in [9]: This evaluation compares the CP-SPP model with (i) an Integer Linear Programming (ILP) algorithm implemented using IBM CPLEX [19]), (ii) First Fit heuristic (based on backtrack algorithm, that returns the first solution found (if any)), (iii) a metaheuristic Genetic Algorithm (GA) (based on refining a population (a set of placements) and continuously generates new placements and adds them into the population), and (iv) the heuristic "DAFNO-InitCO-DCO(0.3)" provided by Xia et al. [9] that relies on a backtrack search algorithm accompanied by two heuristics: (1) naive search that order the fog nodes and applications components, and (2) search based on anchors to minimize average latency. Here, we propose (as considered in [9]) to deploy a single Smart Bell instance on the infrastructure graph depicted in Figure 2.(a) while minimizing a weighted average latency. In Table I, we give the results (in term of quality of the solution and execution times) provided in [9] by these four algorithms. We present also the results obtained by the elaborated CP-SPP model. The column "Quality of the solution" in Table I describes the difference of the solution found compared to the optimal ones. The solution provided by the approaches is normalized according to the optimum. The optimal solution is considered as 100%.

By integrating the objective function considered in [9], the provided CP-SPP allows obtaining the optimal solution almost instantaneously compared to the traditional ILP approach that takes more than 5 minutes. For the first fit algorithm, we remark that the heuristic provides quite bad results whether in terms of computing time or in terms of the quality of the solution obtained. Indeed, the solution found is 87% higher than the optimal solution, this is due to the risk of doing a search based on backtracking that returns the first solution found. GA meta-heuristic provides better results than the First

fit however, in terms of the quality of the solution, it remains far from optimum with difference of 43%. The heuristic proposed by Xia et al. [9] provides a better resolution time but with a slight deviation from the optimum solution. Given these first results, we can say that the CP-SPP model elaborated in this work provide an interesting trade-off between quality of the solution provided and the computing time which is significantly better compared to conventional approaches.

2) Execution Times vs. Number of Deployed Services and Infrastructure Size: These evaluations are based on the second experimental settings presented in Section IV-A. In most works found in the literature, to evaluate the performance of their approaches and placement strategies, authors propose to deploy only one application at each time interval on a relatively small or medium infrastructure [3], our work aims to push the evaluations further by deploying a batch of applications at once and displays the advantage of our model in the context of Fog Computing regarding the scalability issue. Hence, to show the practicality of our framework, we propose to vary the number of deployed service from 1 to 100 applications and observe respectively the building and the resolution time of our model for the first feasible solution that minimizes the number of hosts in Fog infrastructure. We emphasize that the solutions provided by the CP-SPP model present a proven upper bound for the considered minimization problem. Figures 5 depict the results of this analysis for the different network sizes.

For Figure 5, we notice that the deployment of a single service and therefore the definition of a solution that satisfies all the constraints is done almost instantaneously with a very low impact of the infrastructure size, and this for all types of considered applications, i.e, application with 1, 4, 6, or 8 components. The same trend is observed for the deployment of monolithic services. Indeed, for this type of application the size of the infrastructure has a slight influence on the search time of an admissible solution.

For an infrastructure of 120 nodes, we observe that the deployment of 100 applications with number of components less than six are almost instant, and remain low when increasing the services size (less than 1 minute for application with six and eight components). For infrastructure of size 300, we remark that the computing time remains relatively low. For larger infrastructures (*e.g.,* for G^{10} and G^{20}), we remark that our model is not intractable and it provides solutions to such a problem size, which represents a significant advantage. Regarding the execution time trend, it is straightforward to see that this time increase when increasing the number of deployed services and components. This aspect can be favored by using another search strategy than the one used in this paper (see Section III-D). This flexibility is provided by the CP modeling, because we are not dependent on how the problem is solved. So, the improvement and the modification of the model is very simplified in this case.

[3] *e.g.,* in [7], the authors considers an application of three components and proposes to deploy it on infrastructure of size 9. In [11], one application is deployed composed by four components every 2 minutes on infrastructure with 70 nodes. Ditto in many other papers.

V. Conclusion

This paper presents the use of constraint programming to address in a more generic manner the service placement problem in Fog Computing infrastructures. Our model considers the applications and the Fog infrastructure following different scenarios. We provide a set of constraints that formulates the problem and solve it using Choco-solver.

The evaluations we performed show that our implementation on top of Choco-solver provides a good tradeoff between resolution times and solutions quality. We thus argue that CP-based approaches can be effectively used to address this kind of problems. The expressive and flexible modeling language CP natively comes with minimizes placement description effort and eases its enhancement. In addition, the systematic resolution algorithm is almost configuration-less and provides competitive performance.

As future works, we identify three important directions. First, we plan to extend our model to include the notion of the service sharing. Our idea is to add new rules that will enable the merge of identical software components (i.e. processing units that provide the same service) and thus reduce the processing footprint overall. The use-cases we envision are related to the (i) network function chaining and (ii) data-stream problems. In both cases similar functions (load-balancing, firewall, data-filtering, etc.) might be deployed several times across the Fog infrastructure. Second, we will investigate the relevance of our model in the context of the reconfiguration (in the current study, only the initial placement has been considered). The challenges is related to the modeling of the dynamic aspect of both IoT applications and the Fog Computing infrastructure. Finally, it would be interesting to study how our CP-based approach can be extended to answer questions such as "How many replicas of a service we should deploy to ensure a certain quality of services to all users ?"

Acknowledgments

This work was performed and funded by the Inria Project Lab Discovery. This is Open-Science Initiative aiming at implementing a fully decentralized IaaS manager. See http://beyondtheclouds.github.io for more information.

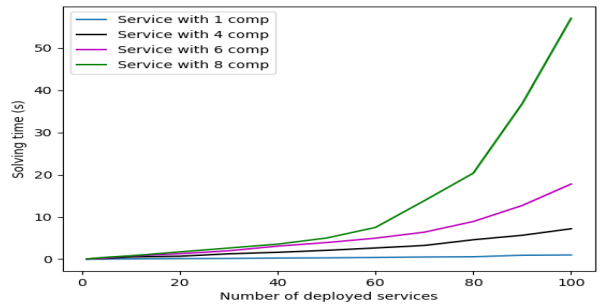
References

[1] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," in *MCC*. ACM, 2012, pp. 13–16.

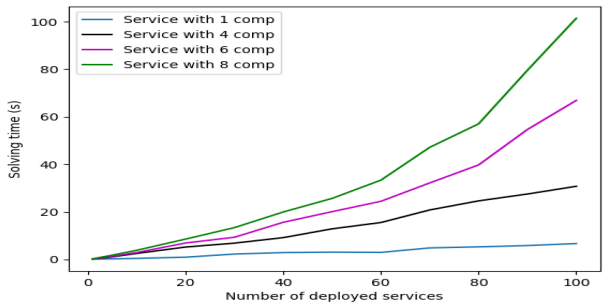
[2] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. A. Lee, and J. Kubiatowicz, "The Cloud is Not Enough: Saving IoT from the Cloud." in *HotStorage*, 2015.

[3] Z. Wen, R. Yang, P. Garraghan, T. Lin, J. Xu, and M. Rovatsos, "Fog Orchestration for Internet of Things Services," *IEEE Internet Comp.*, vol. 21, no. 2, pp. 16–24, Mar 2017.

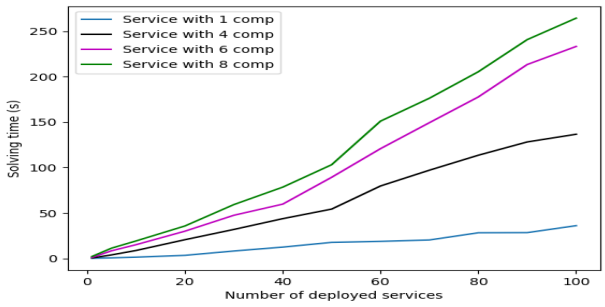
[4] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, and J. P. Jue, "All One Needs to Know about Fog Computing and Related Edge



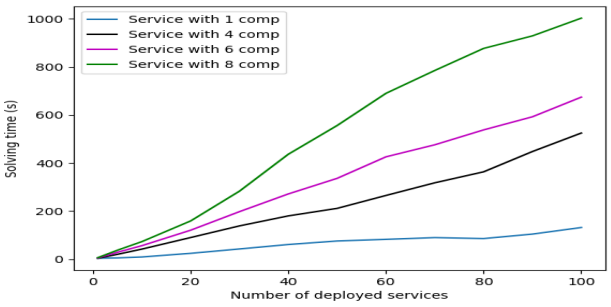
(a) For G^2 (with 120 nodes).



(b) For G^5 (with 300 nodes).



(c) For G^{10} (with 600 nodes).



(d) For G^{20} (with 1200 nodes).

Fig. 5: Execution times vs. number of deployed services.

Computing Paradigms: A Complete Survey," *CoRR*, vol. abs/1808.05283, 2018.

[5] S. B. Nath, H. Gupta, S. Chakraborty, and S. K. Ghosh, "A Survey of Fog Computing and Communication: Current Researches and Future Directions," *CoRR*, vol. abs/1804.04365, 2018.

[6] A. Yousefpour, A. Patil, G. Ishigaki, J. P. Jue, I. Kim, X. Wang, H. C. Cankaya, Q. Zhang, and W. Xie, "QoS-aware Dynamic Fog Service Provisioning," 2017.

[7] A. Brogi and S. Forti, "QoS-Aware Deployment of IoT Applications Through the Fog," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1185–1192, Oct 2017.

[8] O. Skarlat, M. Nardelli, S. Schulte, M. Borkowski, and P. Leitner, "Optimized IoT Service Placement in the Fog," *SOC*, vol. 11, no. 4, pp. 427–443, Dec 2017.

[9] Y. Xia, X. Etchevers, L. Letondeur, T. Coupaye, and F. Desprez, "Combining Hardware Nodes and Software Components Ordering-based Heuristics for Optimizing the Placement of Distributed IoT Applications in the Fog," in *Proc. of the ACM SAC*, 2018, pp. 751–760.

[10] M. I. Naas, P. R. Parvedy, J. Boukhobza, and L. Lemarchand, "iFogStor: An IoT Data Placement Strategy for Fog Infrastructure," in *ICFEC'17*, 2017, pp. 97–104.

[11] B. Donassolo, I. Fajjari, A. Legrand, and P. Mertikopoulos, "Fog Based Framework for IoT Service Provisioning," in *IEEE CCNC*, Jan. 2019.

[12] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence. Elsevier, 2006, vol. 2.

[13] F. Hermenier, X. Lorca, J.-M. Menaud, G. Muller, and J. Lawall, "Entropy: A Consolidation Manager for Clusters," in *ACM SIGPLAN/SIGOPS Int. Conf. on Virtual execution env.*, 2009, pp. 41–50.

[14] C. Prud'homme, J.-G. Fages, and X. Lorca, *Choco Documentation*, TASC - LS2N CNRS UMR 6241, COSLING S.A.S., 2017. [Online]. Available: http://www.choco-solver.org

[15] P. Shaw, "A Constraint for Bin Packing," in *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, Ed. Springer Berlin Heidelberg, 2004.

[16] J. Régin, "A Filtering Algorithm for Constraints of Difference in CSPs," in *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, 1994, pp. 362–367.

[17] G. Pesant, "A Regular Language Membership Constraint for Finite Sequences of Variables," in *Principles and Practice of Constraint Programming – CP 2004*, M. Wallace, Ed., 2004, pp. 482–495.

[18] P. R. Kotecha, M. Bhushan, and R. D. Gudi, "Efficient optimization strategies with constraint programming," *AIChE Journal*, vol. 56, no. 2, pp. 387–404, 2010.

[19] P. Laborie, J. Rogerie, P. Shaw, and P. Vilím, "IBM ILOG CP Optimizer for Scheduling," *Constraints*, vol. 23, no. 2, pp. 210–250, Apr. 2018.

[20] K. Kuchcinski and R. Szymanek, "JaCoP - Java Constraint Programming Solver," 2013.

[21] L. Letondeur, F. Ottogalli, and T. Coupaye, "A Demo of Application Lifecycle Management for IoT Collaborative Neighborhood in the Fog: Practical Experiments and Lessons Learned around Docker," in *2017 IEEE Fog World Congress (FWC)*, Oct 2017, pp. 1–6.

[22] S. S. N. Perala, I. Galanis, and I. Anagnostopoulos, "Fog omputing and Efficient Resource Management in the era of Internet-of-Video Things (IoVT)," in *IEEE Int. Symp. on Circuits and Systems (ISCAS)*, May 2018, pp. 1–5.