To appear in the Proceedings of ALGOCLOUD 2019, Lecture Notes in Computer Science, Springer.

Self-adaptive Container Deployment in the Fog: A Survey

Valeria Cardellini ✉[1] [0000−0002−6870−7083], Francesco Lo Presti[1] [0000−0002−7461−6276], Matteo Nardelli[1] [0000−0002−9519−9387], and Fabiana Rossi[1] [0000−0002−5263−2208]

Department of Civil Engineering and Computer Science Engineering
University of Rome Tor Vergata, Italy
{cardellini,nardelli,f.rossi}@ing.uniroma2.it, lopresti@info.uniroma2.it

Abstract The fast increasing presence of Internet-of-Things and fog computing resources exposes new challenges due to heterogeneity and non-negligible network delays among resources as well as the dynamism of operating conditions. Such a variable computing environment leads the applications to adopt an elastic and decentralized execution. To simplify the application deployment and run-time management, containers are widely used nowadays. The deployment of a container-based application over a geo-distributed computing infrastructure is a key task that has a significant impact on the application non-functional requirements (e.g., performance, security, cost). In this survey, we first develop a taxonomy based on the goals, the scope, the actions, and the methodologies considered to adapt at run-time the application deployment. Then, we use it to classify some of the existing research results. Finally, we identify some open challenges that arise for the application deployment in the fog. In literature, we can find many different approaches for adapting the containers deployment, each tailored for optimizing a specific objective, such as the application response time, its deployment cost, or the efficient utilization of the available computing resources. However, although several solutions for deploying containers exist, those explicitly considering the distinctive features of fog computing are at the early stages: indeed, existing solutions scale containers without considering their placement, or do not consider the heterogeneity, the geographic distribution, and mobility of fog resources.

Keywords: Containers · Elasticity · Fog computing · Placement · Self-adaptive systems.

1 Introduction

Fog computing promises to extend cloud computing exploiting the ever increasing presence of resources located at the edges of the network (e.g., single-board computers, wearable devices, smartphones). However, it introduces new challenges that mainly result from the heterogeneity of computing and networking

resources as well as from their decentralized distribution. Differently from cloud resources, fog resources typically offer a constrained environment, where changes in resource availability, efficiency, and energy consumption play a critical role in determining a successful computing platform. The presence of different Internet connectivity and bandwidth, as well as the dispersed resource distribution, calls for the study of deployment strategies that explicitly take into account at least the presence of heterogeneous resources and non-negligible network delays.

Extending cloud computing towards network edges, fog computing is well suited to manage Internet-of-Things (IoT) applications, whose data are generated and consumed at the network periphery. When interacting with IoT applications, the user requires the application to run with strict quality requirements (e.g., low latency response requirements), often expressed by means of Service Level Agreements (SLAs). In particular, IoT applications usually require reduced response time and high throughput, that should be obtained even in face of highly changing operating conditions. To satisfy these performance goals, the application deployment should be promptly adapted at run-time by conveniently acting according to two control directions: the *placement* and *elasticity* of the application. The application placement addresses the mapping of each application instance to a specific computing resource, while the elasticity feature aims at scaling at run-time the number of application instances and/or the amount of computing resources assigned to each of them. To simplify the deployment and run-time adaptation of applications, we can use software containers. Exploiting a lightweight operating system-level virtualization, software containers (e.g., Docker) have rapidly become a popular technology to run applications on any machine, physical or virtual. Containers enable to bundle together applications and their dependencies (i.e., libraries, code). Differently from virtual machines (VMs), they allow a faster start-up time and a reduced computational overhead.

In this paper, we survey existing solutions to adapt the deployment of container-based applications on fog and cloud computing resources, focusing on the algorithms used to control the adaptation. Different surveys (e.g., [16,26,46,60,77]) have recently investigated the challenges that arise in fog computing environments. Mahmud et al. [46] analyze the challenges in fog computing and discuss its differences with respect to other computing paradigms. Yi et al. [77] identify security and privacy as critical points that should be considered in every stage of fog computing platform design. Specifically, the authors believe that, in a fog environment, general application programming interfaces (APIs) should be provided to cope with existing protocols and APIs. Puliafito et al. [60] analyze the applicability of existing technologies in the fog computing environment in order to support IoT devices and services. Gedeon et al. [26] focus on the application perspective and present a classification and analysis of use cases of edge/fog computing. The survey by Brogi et al. [16] is the one most related to our work since they explore the existing methodologies and algorithms to place applications on fog resources. Differently from these works and in particular from [16], we focus on the runtime execution of fog applications, since their deployment should also efficiently self-adapt with respect to workload changes

and dynamism of the fog computing environment (e.g., fog resource constraints, network constraints in term of latency and bandwidth, fog resources that join or leave the system). Therefore, not only an effective application placement should be enacted as initial deployment, but it should be also conveniently modified at run-time so to be dynamism-aware and deal with the heterogeneity of the underlying fog resources. To this end, the application elasticity plays a key role. Indeed, fog-native applications should be able to adapt to workload changes by provisioning and de-provisioning resources in an autonomic manner, thus coping with the environment dynamism. While the elasticity issue has been well investigated in the cloud environment, as surveyed in [4], as well as in specific domains such as data stream processing [62], to the best of our knowledge it has not yet been analyzed and categorized in the fog context, especially from an algorithmic perspective. Moreover, in this work we aim to identify fully-fledged deployment solutions that can jointly address the elasticity and placement of applications in fog computing environments. When the managed applications are geo-distributed, a fully centralized controller introduces a single point of failure and a bottleneck for scalability. Indeed, a centralized controller may be able to efficiently control the adaptation of only a limited number of entities, and its efficacy may be negatively affected by the presence of network latencies among the application components. Considering the new emerging environment, in this work we want to identify the existing solutions that can be used in practice to decentralize the self-adaptation functionalities.

The rest of the paper is organized as follows. First, we discuss the specific challenges of fog computing environments and their fundamental differences with cloud computing environments (Sect. 2). Second, we present a taxonomy on the existing approaches and deployment controllers used to adapt at run-time the deployment of applications on fog and cloud resources (Sect. 3). Then, in Sects. 4 and 5, we describe some container orchestration tools used to simplify the deployment and management of container-based applications, as well as some simulation tools proposed and used by the research community to perform experiments. We conclude by identifying open research challenges that can be explored to improve the deployment effectiveness in fog environments (Sect. 6).

2 Fog Environment Challenges

Fog computing extends the cloud computing paradigm by expanding computational and storage resources at the edge of the network, in a close proximity to where data are generated. As such, fog environment exposes many old and new challenges. In accordance with previous surveys on fog computing [16,26,46,60,77], we can identify the following most relevant challenges: heterogeneity, scale and complexity, dynamism and mobility, fault tolerance, and security.

Fog and cloud computing infrastructures provide computing resources with different characteristics. Cloud computing offers powerful and general purpose computing (and storage) resources on-demand. Conversely, fog computing usu-

ally exposes heterogeneous resources, with reduced computing and energy capacity, that can also change location at run-time. Also, fog computing can provide storage resources, usually of reduced capacity, that can be used to collect and distribute data from/to edge devices (e.g., AWS Snowball Edge). Being of limited capacity, fog computing resources are cheaper and more constrained than traditional cloud computing resources (e.g., Raspberry Pi); therefore, we assist to a large proliferation of devices standing at the network periphery [17]. As regards the connectivity among resources, we observe that cloud resources reside in a single data center or can be distributed among multiple data centers; either way, they rely on very a fast inter-connectivity that results in negligible communication delays. Conversely, fog resources can communicate using different (and mixed) technologies (e.g., wired, wireless, Bluetooth) that may introduce non-negligible network latency. Such a delay can impact on performance, and be detrimental for latency-sensitive applications.

To rule the complexity of the emerging fog computing environment, efficient algorithms to drive the application deployment are needed. They should explicitly address its heterogeneity and dynamism, which also include the presence of mobile resources (e.g., smartphones). Due to these features and the increased number of constraints, deploying application in a fog computing environment is challenging. As such, many fog computing architectures and platforms have been proposed in literature, aiming to simplify the application distribution and execution (e.g., [30,41]). Most of them resort on lightweight virtualization technologies, i.e., software containers, to simplify the application management (e.g., [12,81]).

Similarly to cloud applications, the user wants to obtain specified levels of Quality of Service (QoS), e.g., in terms of response time, or Quality of Experience (QoE). In cloud computing, the user and the service provider often stipulate a contract referred as SLA. It represents an agreement between the customer and service provider, and is characterized by quantified objectives and metrics (Service Level Objectives, SLOs) which the provider undertakes to respect during service delivery. Defining such kind of agreement is particularly challenging in the fog computing environment, because the SLOs satisfaction is often affected by many factors, which might also be out of the provider's control (e.g., connectivity, mobility).

In the past few years, cloud applications stressed the importance of fault tolerance, and the key role it plays when the application requires a distributed execution. Although many mechanisms can be used to increase fault tolerance (e.g., check-pointing, replication), their implementation in a fog environment is not trivial due to the increased scale, heterogeneity, and complexity with respect to a cloud scenario. However, fault tolerance is a key enabler for the deployment of applications in the fog environment. So far, only a limited number of works explore fault tolerance in the emerging scenario, resulting in an important open challenge to be addressed in the near future [13]. For example, Javed et al. [33] propose a fault-tolerant architecture for IoT applications in edge and cloud infrastructure. Specifically, the proposed solution replicates the processing instancing using the fault-tolerance functionality by Kubernetes; to transfer data with

no loss, the architecture includes a fault-tolerant message broker, implemented using Apache Kafka.

When distributed applications, possibly with IoT sensors and actuators, are deployed on fog resources, the overall system may expose a large number of vulnerabilities, which can represent security threats. Geographically distributed computing and storage resources, that communicate through Internet, might not be easily controlled by a single provider. This further exposes the system to attacks, data leaks, impersonations, and hijacking. So far, many fog platforms and their deployment algorithms have been designed without considering security as a first-class pillar. Moreover, the limited capabilities of fog resources may compromise the applicability of widely adopted security mechanisms [67].

Considering the dynamism and heterogeneity of the fog environment, the discussed challenges and the (unpredictable) changes in the application workload make of primary importance the run-time self-adaption of the deployment of container-based applications. In the next section, we therefore survey existing models and algorithms that explore, possibly in a joint manner, the placement and elasticity control dimensions in a fog computing environment.

3 Approaches for Container-based Application Deployment

In this section, we analyze existing approaches that deal with the deployment of container-based applications on cloud and fog computing resources. We broaden the view also to the cloud environment because, so far, only few research works have specifically targeted the fog environment, especially with regards to the elasticity issue. As we will see, the different research efforts address a wide range of challenges that arise when applications with stringent QoS requirements run in a dynamic and geo-distributed environment. We can classify the existing research works according to: (1) the deployment goals, (2) the scope, (3) the deployment actions, (4) the methodologies used to adapt the deployment, and (5) the deployment controllers. Figure 1 illustrates a taxonomy of the design choices to control the container deployment, whereas Table 1 classifies with respect to the taxonomy the application deployment approaches in literature.

3.1 Deployment Goals

The deployment adaptation of applications is carried out in order to satisfy a variety of QoS requirements. To quantify the deployment objective, several metrics have been adopted in literature; we can broadly distinguish them in user-oriented and system-oriented metrics. A *user-oriented* metric models a specific aspect of the application performance, as can be perceived by the user: e.g., throughput, response time, cost. A *system-oriented* metric aims to quantify a specific aspect of the system, following the service provider's viewpoint who wants to efficiently use the available resources. Considering the Cloud service stack, an IaaS provider wants to maximize profits, minimize resource utilization, while

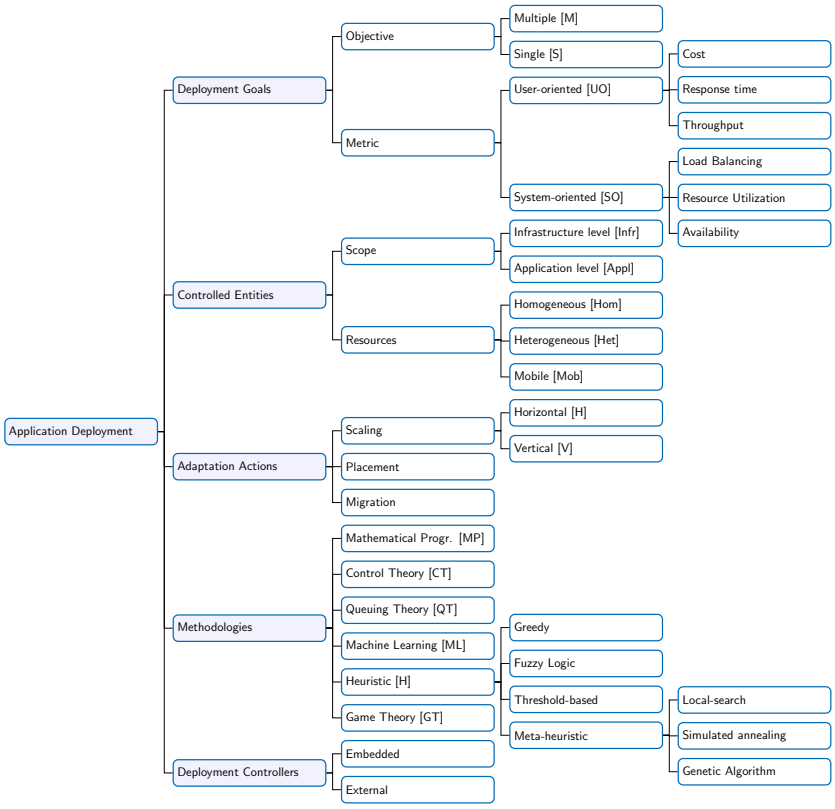

Figure 1: Taxonomy of existing container deployment solutions

fulfilling the SLA agreed with its customers. A PaaS provider can be interested in minimizing the cost associated to the infrastructure utilization. A SaaS customer aims at minimizing the service costs, while achieving a satisfactory QoE level. Deployment policies in literature aim to reduce the application response time (e.g., [64,10,31]), its deployment costs (e.g., [3,11,27,53,54,16]), and/or to save energy consumption (e.g., [9,27,36,37]). To better exploit the on-demand resource allocation, several approaches aim to optimize load balance and resource utilization (e.g., [1,35,47,28]), or to improve system availability (e.g., [39,47,40]). In the context of fog computing, most works consider user-oriented metrics. On the other hand, few works (e.g., [19,23,27,51,63,79]) consider a combination of deployment goals. Casalicchio et al. [19] aim to improve the resource allocation and fulfill application response time constraints. Zhao et al. [79] aim to improve data locality and load balance. Mseddi et al. [51] goal is to optimize the number of served end-users and resource utilization taking into account storage demands. Rossi et al. [63] propose a container-based application deployment strategy to

jointly optimize the 95th percentile of application response time and resource utilization. De Maio et al. [23] propose a hybrid approach for task offloading in mobile edge computing scenarios which jointly maximize user-oriented (i.e., user QoE) and system-oriented (i.e., provider profit) metrics.

Table 1: Classification of existing solutions for deploying applications in geo-distributed computing environments according to the taxonomy in Figure 1.

Ref.	*Depl. goals*		*Controlled entity*		*Adaptation Actions*			*Methodologies*
	Objective	Metric	Scope	Resources	Scaling	Placement	Migration	
Abdelbaky et al. [1]	M	UO + SO	Appl.	Het.	No	Yes	No	MP
Addya et al. [2]	M	SO	Infr.	Hom.	No	Yes	No	H
AlDhuraibi et al. [3]	M	UO + SO	Appl.	Hom.	V	No	Yes	H
Ali-Eldin et al. [6]	S	SO	Infr.	Hom.	H	No	No	CT + QT
Arabnejad et al. [7]	M	UO + SO	Infr.	Hom.	H	No	No	ML + H
Arkian et al. [8]	M	UO	Infr.	Het..	No	Yes	No	MP
Asnaghi et al. [9]	S	SO	Appl.	Hom.	V	No	No	CT + H
Baresi et al. [10]	S	UO	Appl.	Hom.	H+V	No	No	CT
Barna et al. [11]	S	SO	Infr. + Appl.	Hom.	H	No	No	QT + H
Brogi et al. [15]	M	UO + SO	Appl.	Het.	No	Yes	No	ML + H
Casalicchio et al. [19]	M	UO + SO	Appl.	Hom.	H	No	No	H
Garefalakis et al. [25]	M	SO	Appl.	Hom.	No	Yes	No	MP
Guan et al. [27]	M	SO	Appl.	Hom.	H	Yes	No	MP
Guerrero et al. [28]	M	SO	Appl.	Het.	H	Yes	No	H
Horovitz et al. [31]	S	UO	Appl.	Hom.	H	No	No	ML + H
Huang et al. [32]	S	SO	Appl.	Hom.	No	Yes	No	MP
Kaewkasi et al. [35]	S	SO	Appl.	Hom.	No	Yes	No	H
Kaur et al. [36]	M	SO	Appl.	Hom.	No	Yes	Yes	H + GT
Kayal et al. [37]	M	SO	Appl.	Het.	No	Yes	No	MP
Khazaei et al. [39]	M	SO	Appl.	Hom.	H	No	No	H
Khazaei et al. [40]	M	SO	Appl.	Hom.	H	No	No	H
Mahmud et al. [45]	M	UO	Appl.	Het.	No	Yes	No	H
Mao et al. [47]	M	SO	Appl.	Het.	No	Yes	Yes	QT + H
Mennes et al. [49]	S	SO	Appl.	Het.	No	Yes	No	H
Mouradian et al. [50]	M	UO	Appl.	Het. + Mob.	No	Yes	No	MP + H
Mseddi et al. [51]	M	UO + SO	Appl.	Het. + Mob.	No	Yes	Yes	H
Naas et al. [52]	M	UO	Appl.	Het.	No	Yes	No	MP + H
Nardelli et al. [53]	M	UO + SO	Infr. + Appl.	Het.	H	Yes	No	MP
Nardelli et al. [54]	M	UO	Infr. + Appl.	Het.	H	Yes	No	MP
Nouri et al. [56]	M	SO	Appl.	Hom.	H	No	No	ML
Rossi et al. [63]	M	UO + SO	Appl.	Het.	H + V	Yes	No	MP + ML
Rossi et al. [64]	M	UO + SO	Appl.	Hom.	H + V	No	No	ML
Santos et al. [65]	S	UO	Appl.	Het.	No	Yes	No	H
Souza et al. [66]	S	UO	Appl.	Hom.	No	Yes	No	H
Tan et al. [69]	S	SO	Infr. + Appl.	Hom.	No	Yes	No	H
Tang et al. [70]	M	SO	Appl.	Het. + Mob.	No	No	Yes	MP + ML
Tesauro et al. [71]	S	UO	Infr.	Hom.	No	Yes	No	QT + ML
Townend et al. [72]	S	SO	Appl.	Hom.	No	Yes	No	H
Wu et al. [75]	S	SO	Appl.	Hom.	H	No	No	H
Yigitoglu et al. [78]	M	UO + SO	Appl.	Het. + Mob.	No	Yes	No	H
Zhao et al. [79]	M	UO + SO	Appl.	Het.	No	Yes	No	H
Zhu et al. [82]	M	UO + SO	Infr.	Hom.	V	No	No	CT + ML

3.2 Controlled Entities

To identify the *scope*, we observe that adaptation actions can be applied either at the *infrastructure level* [4] or at the *application level* [44]. At the *infrastructure level*, the elasticity controller changes the number of computing resources, usually by acquiring and releasing VMs, e.g., [6,54,71]. At the *application level*, the controller adjusts the computing resources directly assigned to the application (e.g., changing their parallelism degree [3,27,64]).

Fog environments can include resources with different computing and storage capacity as well as network connectivity. Therefore, some deployment solutions explicitly consider *resource heterogeneity*, i.e., they take into account specific features of computing and networking resources, such as processing or storage capacity of resources, available resources, or network delay (e.g., [28,53,63,70]). Nonetheless, a large number of solutions model only a homogeneous computing infrastructure (e.g., [6,7,9,64,66,82]). Moreover, user devices and/or fog resources (e.g., smart cars, drones) can be *mobile*. Most works consider only user mobility and address the application migration among multiple resources (e.g. [58]) or the placement of static edge nodes in a cellular network [23] with the goal to satisfy the application SLOs. To the best of our knowledge, only the work by Mouradian et al. tackles the mobility of fog nodes [50], while there are more efforts in the research area of vehicle cloud computing (e.g., [80]).

Software containers offer a lightweight virtualization solution, which is often adopted in the context of fog computing (e.g., [30]), even in extremely constrained nodes as fog gateways [12]. Souza et al. [66] analyze the challenges of fog computing environments and propose containers as a possible solution to smoothly deploy application across geo-distributed fog nodes. When applications are containerized, a *single-level deployment* regards the container placement on the underlying (physical or virtual) resources. In addition, depending on the virtualization layering, a double-level deployment can involve the placement of virtual resources (i.e., VMs) onto physical computing resources. Most works consider a single level of deployment (e.g., [3,27,79,2,64]), while only a few solve a *multi-level deployment* problem [11,53,69].

3.3 Adaptation Actions

The *adaptation actions* to control at run-time the deployment of container-based applications include the application placement, the application elasticity according to two possible directions (i.e., horizontal and vertical scaling), and the migration of some application components. The *elasticity problem* determines *how* and *when* to perform scaling operations, thus enabling elastic applications that can dynamically adapt in face of workload variations. Horizontal scaling allows to increase (scale-out) and decrease (scale-in) the number of application instances (e.g., containers or VMs). Vertical scaling allows to increase (scale-up) and decrease (scale-down) the amount of computing resources assigned to each application instance. A fine-grained vertical scaling is preferred to more quickly

react to small workload changes, while a horizontal scaling operation makes easier to react to sudden workload peaks. However, most of the existing solutions consider either horizontal or vertical scaling operations to change at run-time the application deployment (e.g., [3,31,7,9,11,54,53]).

Differently from cloud computing environment, the presence of heterogeneous fog resources emphasizes the importance of the application *placement problem*. Its goal is to define the computing resources that will host and execute each application instance. Most of the existing solutions consider the two problems separately and focus either on the placement or on the elasticity of application instances (e.g., [8,10,71]). So far, only a limited number of works have studied how to jointly solve the two problems (e.g., [53,27,28,63]).

When the application placement is updated at run-time, it results in (stateless or stateful) migrations of virtualized resources (i.e., containers or VMs), that can be moved from one location to another. *Migration* is used to improve system performance, seeking to balance load or to maximize resource utilization. In addition, it allows to cope with user and/or resource movement across different geographical locations. For example, Kaur et al. [36] propose a technique that allows task scheduling on lightweight containers and supports container migration within or between the VMs. Elliott et al. [24] present a novel approach that enables the rapid live migration of stateful containers between hosts belonging to different cloud infrastructures. However, migration has a cost, because the application downtime during migration, although minimal, cannot be avoided. Therefore, a trade-off between migration benefits and cost should be considered.

3.4 Methodologies

The methodology identifies the class of algorithms used to plan how the application deployment should be changed so to achieve the deployment goals. Elasticity and placement are often considered as two orthogonal problems [9,32,66]. Nonetheless, few research efforts propose policies that jointly address the two problems (e.g., [27,63]). Considering that scaling in the fog environment take place in a geo-distributed context, where network latencies among computing resources cannot be neglected as when scaling inside a data center, we believe that the two issues cannot be separately solved.

We classify the methodologies in the following categories: mathematical programming, control theory, queuing theory, machine learning, and heuristics.

Mathematical Programming. *Mathematical programming* approaches exploit methods from operational research in order to determine or adapt at runtime the placement of application instances, to change the application parallelism, or a combination thereof (e.g., [8,27,47,52]). The formulation and resolution of Integer Programming (IP) problems belongs to this category.

When the deployment problem is formulated as an IP optimization problem, its most general definition can be described as follows. Given an application with n instances, a deployment strategy can be modeled by associating to each

application instance $i = 1, \ldots, n$ a vector $\mathbf{x}^i = (x_1^i, \ldots, x_R^i)$, with R the set of fog resources, where $x_r^i = 1$ if an application instance is placed on the fog resource $r \in R$, 0 otherwise. The deployment problem can be expressed as:

$$\begin{aligned} \textbf{min } & F(\mathbf{x}) \\ \textbf{subject to: } & Q^\alpha(\mathbf{x}) \leq Q^\alpha_{\max} \\ & Q^\beta(\mathbf{x}) \geq Q^\beta_{\min} \\ & \mathbf{x} \in D \end{aligned} \tag{1}$$

where $\mathbf{x} = (\mathbf{x}^1, \ldots, \mathbf{x}^n)$ is the vector of the application instances deployment variables. $F(\mathbf{x})$ is a suitable deployment objective function to be optimized. $Q^\alpha(\mathbf{x})$ and $Q^\beta(\mathbf{x})$ are, respectively, those QoS attributes whose values are bounded by a maximum and a minimum, respectively, and $\mathbf{x} \in D$ is a set of functional constraints.

Most of the existing solutions use IP formulations to solve (only) the placement problem of application instances. Mao et al. [47] present an IP formulation of the initial container placement aiming to maximize the available resources in each hosting machine. Garefalakis et al. [25] propose Medea, a new cluster scheduler based on Apache Hadoop YARN. Medea solves an Integer Linear Programming (ILP) placement problem to meet global cluster objectives, such as to minimize the number of application constraint violations, reduce resource fragmentation, balance node load, and minimize number of active computing nodes. However, fog-based deployment goals are not considered. Arkian et al. [8] solve a Mixed-ILP (MILP) problem to deploy application components (i.e., VMs) on fog nodes to satisfy end-to-end delay constraints. Huang et al. [32] model the mapping of IoT services to edge/fog devices as a quadratic programming problem, that, although simplified into an ILP formulation, may suffer from scalability issues. To reduce the resolution time that limits the system size scalability, Naas et al. [52] exploit the geographic distribution of fog resources so to identify subproblems that are then solved separately. Zhao et al. [79] deal with the scheduling of containerized cloud applications with the goal to make them more aware of their data locality. To address the limited scalability of the proposed mathematical optimization problem (which is a variant of the Multiple Knapsack Problem and therefore NP-hard), they devise heuristic algorithms, tackling the problem in a bottom-up fashion. Such a resolution approach is well rooted in the fog environment, characterized by a hierarchical architecture. Kayal et al. [37] present an autonomic service placement strategy based on Markov approximation to map microservices to fog resources without any central coordination.

In literature there are some works that consider mathematical approaches not only to address the application placement problem but also to jointly solve the elasticity problem (e.g., [27,53,63]). For example, Guan et al. [27] present a LP formulation to determine the number of containers and their placement on a static pool of physical resources; nevertheless, vertical scaling operations are not considered. Nardelli et al. [53] propose an optimization problem formulation of the elastic provisioning of VMs for container deployment taking into account

the time needed for the deployment reconfiguration. A multi-level optimization problem is defined: at the first level, it deals with the elastic adaptation of the number and type of application instances (i.e., containers); at the second level, it defines the container placement on a set of VMs that can be elastically acquired and released on demand. Rossi et al. [63] propose a two-step approach that manages the run-time adaptation of container-based applications deployed over geo-distributed VMs. An ILP problem is formulated to place containers on VM, with the aim of minimizing adaptation time and VM cost.

Some works have addressed the problem of offloading computation in a fog environment, For example, Liu et al. [42] formulate a multi-objective optimization problem, which involves minimizing the energy consumption, delay, and payment cost. Chang et al. [21] propose an energy-efficient optimization problem to find the optimal offloading probability and transmission power. By using the method of multipliers [14], they allow to deal with it in a distributed manner.

The main drawback of the mathematical programming approaches is scalability. Indeed, the deployment problem is NP-hard and resolving the exact formulation may require prohibitive time when the problem size grows.

Control Theory. A deployment policy based on *control theory* usually identifies three main entities: decision variables, disturbance, system configuration. Then, it adapts consolidate theory to determine the next system configuration that satisfies the deployment objectives. The decision variables identify the placement or replication of each application instance. The disturbances represent the events that cannot be controlled, e.g., incoming data rate, load distribution, and processing time; nevertheless, it is usually assumed that their future value can be predicted, at least in the short term. By combining the decision variables, alternative configurations of the application deployment can be obtained, which result in different performance, e.g., in terms of application latency or throughput. There are three types of control systems: open-loop, feedback and feed-forward. Open-loop controllers (without feedback) are based exclusively on system input, not being able to analyze the output. Feedback controllers, on the other hand, monitor the output of the system in order to correct any deviations from the final goal. Feed-forward controllers can be used to implement a proactive approach as they predict, using a model, the behavior of the system and react before the error is produced.

Baresi et al. [10] model a control system for horizontal and vertical scaling of applications. They combine infrastructure and application adaptation using a novel deployment planner that consists of a discrete-time feedback controller. In their work, a nonlinear, time-invariant dynamic system controls the application response time as a function of the assigned CPU cores (decision variables) and the request rate (disturbance). Zhu et al. [82] use control theory combined with reinforcement learning techniques to adapt the applications deployment in cloud computing environments. To dynamically add or remove VMs of cloud services, Ali-Eldin et al. [6] propose two adaptive reactive/proactive controllers. They model a cloud service and estimate the future load using queuing theory.

Queuing Theory. *Queuing theory* is often used to estimate the application response time. The key idea is to model the application as a queuing network with inter-arrival times and service times having general statistical distributions (e.g., M/M/1, M/M/k, Gi/G/k). To simplify the analytical investigation, the application is considered to satisfy the Markovian property, thus leading to approximated system behavior (and performance metrics).

Since queuing theory allows to predict the application performance under different conditions of load and number of replicas, it is often used to drive scaling operations (e.g., [11,47]), also in combination with other techniques (e.g., [6,66,71]). Mao et al. [47] model a four-tier application using queuing theory. A centralized deployment controller takes scaling decisions using the queuing model of each application layer. Using a Layered Queuing Network, Barna et al. [11] use the number of user requests and the application topology to estimate the resource utilization and application response time. Ali-Eldin et al. [6] and Tesauro et al. [71] combine queuing theory with control theory and machine learning, respectively.

Machine Learning. In the field of *machine learning*, reinforcement learning (RL) is a special technique that has been used to adapt the application deployment at run-time. RL refers to a collection of trial-and-error methods by which an agent can learn to make good decisions through a sequence of interactions with a system or environment. As such, the agent learns from experience the adaptation policy, i.e., the best adaptation action to take with respect to the current system state. The system state can consider the amount of incoming workload, the current application deployment, or its performance (e.g., [64]). When the agent applies an action, the system transits in a new state and the agent receives a reward, that indicates the action goodness. The received reward and the next state transition usually depend on external unknown factors. One of the challenges that arise in RL is the trade-off between exploration and exploitation. To maximize the obtained reward, the RL agent must prefer actions known to provide high reward (exploitation). However, in order to discover such actions, it has to try actions not selected before (exploration). The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. To maximize the expected long-term reward, the agent estimates the so-called Q-function. It represents the expected long-term reward that follows the execution of an action in a specific system state. Different strategies can be used to estimate the Q-function, ranging from model-free (e.g., Q-learning, SARSA) to model-based solutions; these solutions exploit different degrees of system knowledge to approximate its behavior [68].

RL has mostly been applied to devise policies for VM allocation and provisioning (e.g., [7,71]) and, in a limited way, to manage containers (e.g., [31,64]). Horovitz et al. [31] propose a threshold-based policy for horizontal container elasticity using Q-learning to adapt the thresholds. Nouri et al. [56] describe a decentralized RL-based controller to scale a web application running on cloud computing resources. Interestingly, they design a decentralized architecture, where

each server is responsible for maintaining the performance of its own-hosted applications, while fulfilling the requirements of the whole system. This decentralized approach is well suited to rule complexity of nowadays fog computing environments.

Being model-free solutions, Q-learning and SARSA may suffer from slow convergence rate. To overcome this issue, Tesauro et al. [71] propose a hybrid RL method to dynamically allocate homogeneous servers to multiple applications. They combine the advantages of both explicit model-based methods and tabula rasa RL. Instead of training a RL module online, they propose to train offline the RL agent using collected data, while an initial policy (based on a queuing model) drives management decisions in the system. Arabnejad et al. [7] combine Q-learning and SARSA RL algorithms with a fuzzy inference system that drives VM auto-scaling. Rossi et al. [64] present RL policies to control (horizontal and vertical) elasticity of containers so to satisfy the average application response time. To speed-up the learning phase, they propose a model-based RL approach that exploits the (known or estimate) system dynamics.

Another approach to solve the slow convergence rate of RL consists in approximating the system state or the action-value function; as such, the agent can explore a reduced number of system configurations [68]. Tang et al. [70] propose a RL algorithm that controls the migration of containers in a fog environment. In particular, they define a multi-dimensional Markov Decision Process aimed to minimize communication delay, power consumption and migration costs; interestingly, to deal with the large number of system states, the authors integrate a deep neural network within the Q-learning algorithm.

Recently, RL approaches have also been used to drive the decision of offloading computation from mobile devices to cloud resources (e.g., [5,76]). Alam et al. [5] propose a deep Q-learning based offloading policy suited for mobile fog environments. To minimize the service latency, offloading decisions are taken by considering resource demand and availability as well as the geographical distribution of mobile devices. Xu et al. [76] present a post-decision state solution for managing computing resources, which learns on-the-fly the optimal policy of dynamic workload offloading and edge resource provisioning.

Heuristics. Different *heuristics* have been proposed to solve the placement and elasticity of container-based applications. The most popular heuristics include: greedy heuristics (e.g., [66,78]), fuzzy logic (e.g., [7,46]), threshold-based heuristics (e.g., [11,40]), meta-heuristics (e.g., [28,35]), and specifically designed solutions (e.g., [55]).

Due to their design simplicity, *greedy heuristics* are often adopted to allocate containers. Yigitoglu et al. [78] propose to place the application containers on the available fog resources in a greedy first-fit manner. Souza et al. [66] propose a greedy best-fit heuristic that first sorts the applications according to their processing demand, and then allocates them on the available fog resources; if there is not enough processing capacity available, cloud computing resources are used. Along with the simple best-fit solution, the authors also propose a "best-

fit with queue" heuristic that offloads applications to the cloud, exploiting the estimated application response time.

The purpose of *fuzzy logic* is to model human knowledge; it allows to convert knowledge in rules, that can be applied to the system to identify suitable deployment actions. The fuzzy logic usually includes three phases: fuzzification, fuzzy inference, and defuzzification. In fuzzification, system states or metrics are converted into equivalent fuzzy dimensions by using a membership function. During fuzzy inference, fuzzy inputs are mutually compared to determine the corresponding fuzzy output. A set of fuzzy rules assists in this case. Fuzzy rules are collections of *if-then* rules that represent how to take decisions and control a system according to human knowledge. In a fuzzy inference, any number of fuzzy rules can be triggered. Then, the fuzzy outputs are combined through a defuzzification function so to derive a metric related to the application placement request. Mahmud et al. [45] propose a QoE-aware placement policy based on fuzzy logic, which prioritizes different application placement requests and classifies fog computing resources. Arabnejad et al. [7] combine the fuzzy controller with a model-free RL algorithm to horizontally scale VMs at run-time.

Many solutions exploit best-effort *threshold-based policies* to change the application replication degree or to recompute the application instance placement at run-time. Threshold-based policies represent the most popular approach to scale at run-time application instances (i.e., containers) also for the cloud infrastructure layer. Orchestration frameworks that support container scaling (e.g., Kubernetes, Docker Swarm, Amazon ECS) usually rely on best-effort threshold-based policies based on some load metrics (e.g., CPU utilization). The main idea is to increase (or reduce) the application parallelism degree or to change the application instance placement as soon as a QoS metric is above (or below) a critical value. Several works use as QoS metric the utilization of either the system nodes or the application replicas. Most of works use policies based on the definition of static thresholds. Barna et al. [11] propose a static threshold-based algorithm which determines the scaling action taking into account the average CPU utilization of the containers in a cluster. Static thresholds are also used for planning the adaptation of container deployment (e.g., [39,40,3,36]). Khazaei et al. [39,40] take into account CPU, memory, network utilization to determine the scaling action of container-based application. Al-Dhuraibi et al. [3] propose ElasticDocker, which employs a threshold-based policy to vertically scale CPU and memory resources assigned to each container. Kaur et al. [36] use a static threshold-based approach to enable container migration. The migration would be initiated whenever the utilization of the computing nodes exceeds or falls behind the predefined upper and lower threshold limits, respectively. All these approaches require a manual tuning of the thresholds, which can be cumbersome and application-dependent. To overcome this limitation, Horovitz et al. [31], for example, propose a threshold-based policy for horizontal container elasticity that uses Q-learning to dynamically adapt the thresholds at run-time.

Among *meta-heuristics*, we can include local search, simulated annealing, and genetic algorithms. Greedy approaches or *local search* solutions that greedily

explore local changes may get stuck in local optima and miss the identification of global optimum configurations. Conversely, *simulated annealing* is a popular meta-heuristic that first aims to find the region containing the global optimum configuration, and then moves with small steps towards the optimum. To the best of our knowledge, simulated annealing has not been yet used in the context of fog computing. Starting from initial configuration, this technique randomly generates a new neighbouring configuration, aiming to find a better deployment solution. If the best computed solution does not improve the previous one, it can be accepted with a certain probability (referred as temperature), which decreases over time (e.g., [2]).

A *genetic algorithm* generates a random population of chromosomes, which represent deployment configurations. Then, it performs genetic operations, such as crossover and mutations, to obtain successive generations of these chromosomes. A crossover operator takes a pair of parent chromosomes and generates an offspring chromosome by crossing over individual genes from each parent. A mutation operator randomly alters some parts of a given chromosome so to avoid to get stuck in a local optimum. Afterwards, the genetic algorithm picks the best chromosomes from the entire population based on their fitness values and eliminates the rest. This process is repeated until a stopping criterion is met. Guerrero et al. [28] present a genetic algorithm for container horizontal scaling and allocation on physical machines; however, this solution does not take explicitly into account the characteristics of a geo-distributed environment (i.e., network delay between fog resources). To solve the fog placement problem, Tan et al. [69], Wen et al. [73], and Mennes et al. [49] propose service placement solutions based on genetic algorithms. Tan et al. [69] provide a novel problem definition of the two-level container allocation problem. Specifically, they design a genetic algorithm to automatically generate rules for allocating VMs to physical nodes. Even though genetic algorithms considerably reduce the need of systematically exploring large solution space (thus reducing the resolution time), they are not well suited to quickly react to the dynamism of a fog computing environment. To overcome this issue, recent approaches combine genetic algorithms with Monte Carlo simulations (e.g., [23,15]). De Maio et al. [23] focus on offloading application tasks in a mobile edge computing scenario, whereas Brogi et al. [15] target the multi-service application placement in the Fog.

Kaur et al. [36] consider a multi-layer computing infrastructure that allows to process tasks on fog and cloud computing resources. The scheduling problem maps tasks to broker and, then, from broker to containers across VMs. To solve the task scheduling problem, the authors propose a *game theoretical* solution. The primary objective of the cooperative game is to schedule the set of task requests to containers so that the overall energy utilization of VMs and response time of tasks are minimized. In the game, each player (i.e., broker) attempts to reduce the overall communication cost based on its current bandwidth and load status. The utility function of brokers is formulated using weighted contributions of these two metrics (i.e., bandwidth and load).

3.5 Deployment Controllers

The deployment controller is the software component in charge of controlling the deployment of applications or computing resources. In the context of fog computing, deployment controllers usually manage the execution of (containerized) applications on heterogeneous and geo-distributed computing resources. Besides determining the initial deployment, this controller can be used to adapt the application deployment at run-time so to respond to system or workload changes. The deployment controller usually provides deployment mechanism, so it can be equipped with centralized or decentralized deployment policies. Few solutions integrate the deployment controller within the application code (e.g., embedded elasticity [4]). Having no separation of concerns, the application itself should also implement mechanisms and policies steering the adaptation. Although this approach enables optimized scaling policies, it complicates the application design.

Conversely, most research efforts use an *external deployment controller* to carry out the adaptation actions (e.g., [22,11,31,40,9,30,41,64]). Such approach improves software modularity and flexibility. Kimoviski et al. [41], for example, propose SmartFog, a nature-inspired fog architecture. Modeling the fog environment as the human brain, SmartFog is capable of providing low-latency decision making and adaptive resource management. The fog nodes are modeled as neurons, while the communication channels as synapses. Fog nodes are capable of self-clustering into multiple functional areas. IoT devices and sensors are represented as the sensory nervous system. Cloud computing resources support communication between the different functional areas.

Extending the existing orchestration tools (see Section 4), the external controllers usually implement a MAPE control loop [38]. The latter includes four main components (Monitor, Analyze, Plan and Execute) that manage the self-adaptation functions. The Monitor collects data about the application and the execution environment. The Analyze component uses the collected data to determine whether an adaptation is beneficial. If so, the Plan component determines an adaptation plan for the application, which is enacted through the Execute component. Different patterns to design multiple MAPE loops have been used in practice by decentralizing the self-adaptation components [74], being the *master-worker* the most used one. In the master-worker decentralization pattern, the system includes a single master, which runs the centralized Analyze and Plan phases, and multiple independent workers, which run the decentralized Monitor and Execute phases. To manage services in a fog environment, De Brito et al. [22] propose an architecture that includes a multitude of decentralized agents, coordinated by a single orchestrator (which could be elected among the agents). For container deployment in a fog computing environment, Hoque et al. [30] extend an existing orchestration tool (i.e., Docker Swarm) according to a master-worker decentralization pattern. No fog-aware orchestration policy is provided. A centralized master component allows to more easily design the self-adaptation policies and compute globally optimal reconfiguration strategies. However, it may easily become the system bottleneck when it has to control a great number of entities in a large-scale geo-distributed system.

4 Container Orchestration Tools

To simplify the deployment and management of applications over fog and cloud computing resources, most of the existing solutions exploit software containers. A software container allows to tie an application with all the dependencies required for its execution, such as libraries, configurations, and data. Docker is the most popular container management system, which allows to create, distribute, and run applications inside containers. Although it is easy to manually deploy a single container, managing a complex application (or multiple applications) during its whole lifetime requires a container orchestration tool. The latter automatizes the container provisioning, management, communication, and fault-tolerance. Although several container orchestration tools exist [20,61], nowadays the most used ones in the academic and industrial scenarios are Docker Swarm, Apache Mesos, and Kubernetes.

Docker Swarm is an open-source platform that enables to simplify the execution and management of containers across multiple computing nodes[1]. There are two types of nodes: managers and workers. The manager nodes perform the orchestration and management functions required to maintain the desired cluster state; they elect a single leader to conduct orchestration and scheduling tasks. The worker nodes execute tasks received from the leader node; they do not participate in taking scheduling decisions and in maintaining the cluster state. To manage the global cluster state, the manager nodes implement the Raft algorithm for distributed consensus [57]. Let n be the number of managers, Raft tolerates up to $(n-1)/2$ failures and requires a quorum of $(n/2)+1$ managers to agree on the cluster state. Having the same consistent state across the cluster means that, in case of unexpectedly leader failure, any other manager can restore the services to a stable state.

Apache Mesos allows to share resources in a cluster between multiple frameworks ensuring resource isolation[2]. Mesos can be considered as a kernel for the data center: it provides a unified view of all node resources and shares the available capacity among heterogeneous frameworks. The main components of Mesos are the master, the workers and the (external) frameworks. The master is responsible for mediating between the worker resources and the frameworks. At any point, Mesos has only one active master, which is elected through distributed consensus using Zookeper. The master offers worker resources to frameworks, and launches tasks on workers for the accepted offers. The workers manage various resources (e.g., CPU, memory, storage), and can execute tasks submitted by the frameworks. A framework is an application to run on Mesos and consists of, at least, a scheduler and an executor. The framework scheduler is responsible for accepting or rejecting resources offered by Mesos, while the executors consume resources to run application-specific tasks.

[1] `https://docs.docker.com/engine/swarm/`
[2] `http://mesos.apache.org`

Kubernetes[3] is an open-source platform developed and released by Google to manage container-based applications in an autonomic manner. Kubernetes architecture also follows the master-worker decentralization pattern, where the master uses worker nodes to manage resources and orchestrate applications (using pods). Multiple master nodes provide a highly-available replicated cluster state through the Raft consensus algorithm. A worker node is a physical or virtual machine that offers its computational capability for executing pods in a distributed manner. A pod is the smallest deployment unit in Kubernetes, which consists of a single container or a reduced number of tightly coupled containers. When multiple containers run within a pod, they are co-located and scaled as an atomic entity. To provide a specific service, Kubernetes can ensure that a given number of pods are up and running using a ReplicaSet. To further simplify the deployment of applications, Kubernetes exposes DeploymentControllers, a higher-level abstraction built upon the ReplicaSet concept. Kubernetes includes Horizontal Pod Autoscaler, which automatically scales the number of pods in a DeploymentController by monitoring, as default metric, CPU utilization. Experimental results in [34] demonstrate that, for complex application deployments, Kubernetes performs better than other orchestration tools.

We observe that all the above-mentioned orchestration tools have been specifically designed for clustered environments, so they are not well-suited for managing applications in a geographically distributed environment. Indeed, their placement policies do not take into account the heterogeneity and geographic distribution of the available computing resources. For example, Kubernetes' default scheduler spreads containers on cluster's worker nodes, while Docker Swarm distributes containers so to optimize for the node with the least number of containers. We also note that, as regards elasticity, these orchestration tools are usually equipped with basic policies, such as static threshold-based policies on system-oriented metrics. As discussed in Sect. 3.4, setting such thresholds is a cumbersome and error-prone task and may require knowledge of the application's resource usage to be effective. To address these limitations, some research works aim to improve existing orchestration tools (e.g., [55,65,72,75]). Wu et al. [75] modify Kubernetes Horizontal Pod Autoscaler to adapt at run-time the deployment of containerized data stream processing applications according to the predicted load arrival rate. Netto et al. [55] propose a state machine approach to scale Docker containers in Kubernetes. Santos et al. [65] extend the default Kubernetes scheduler so to select nodes using a policy that minimizes the round trip time between the node and a target location (labels are used to statically assign the round trip time to each node).

5 Simulation Tools

A large number of research works resort on simulation to evaluate application performance in distributed computing environments (e.g., [2,27,32,45,66,69]). On the one hand, simulators enable to more easily evaluate deployment policies

[3] https://kubernetes.io

under different configurations and workload conditions. On the other hand, it is not often clear how accurately they capture the dynamism of distributed computing environments. Fog simulators allow to model the heterogeneity of computing resources, which can be geographically distributed. Fog resources are often organized as a graph; some simulators allow to further aggregate resources in groups (also called cloudlets or micro-data centers). Although most recent simulators model both cloud and fog computing resources, few existing solutions offer the possibility to simulate mobility.

ContainerCloudSim [59] is a discrete-event-based simulator that supports the evaluation of different container placement policies in cloud environments. Extending CloudSim [18], ContainerCloudSim allows to model hosts, VMs, containers, and tasks. For each host, its processing, memory, and storage capacity, as well as the belonging data center should be specified. Each host can run one or more VMs where containers can be deployed. For each container, it should be specified the required CPU and memory resources, needed to execute tasks.

EmuFog [48] is a framework for emulating a fog environment. In EmuFog, a network is modeled as an undirected graph of devices (switches and routers) connected together through communication channels (links). To create a fog environment, the first step is to translate the network topology (generated or imported) in a network topology supported by EmuFog. The second step consists in defining the type and location of nodes. Although EmuFog allows to easily create fog environments, it does not support application modeling.

iFogSim [29] provides a platform to simulate a fog environment and to deploy applications. Based on CloudSim, it supports elasticity and migration of VMs. The fog network topology structure should be tree-like: the deployment of application instances starts from tree leaves (fog nodes) and proceeds up to the tree root (usually, the cloud). iFogSim allows to monitor latency, network congestion, energy consumption and resource utilization of the application instances. The application is modeled as a directed graph: vertices represent the processing units (i.e., modules), whereas edges are the data flow between the modules. The communication between the different application modules occurs by sequentially sending tuples. With respect to the other simulators, iFogSim allows to model realistic multi-component applications. Nevertheless, it is not possible to express network topologies different from tree-like. Furthermore, it does not support node mobility. To overcome this limitation, Lopes et al. [43] proposed MyiFogSim, an extension that supports mobility.

6 Open Challenges and Research Directions

Extending cloud computing, fog computing promises to improve scalability of distributed applications and to reduce their response time. Nevertheless, the fog environment presents several key features (e.g., large-scale distribution, resource heterogeneity) that introduce new challenges. The research community has been dealing with these challenges in the last years; however, we are still at the first stages, and there are several open issues and research directions to investigate.

Among all the interesting challenges, we identify a few of them that we consider to be of utmost importance: elasticity and placement of multi-component applications, mobility, scalability, fault-tolerance, security, and SLA definition.

The existing deployment algorithms usually consider single-component applications. However, modern applications often result by composing multiple micro-services, where the adaptation of an application component is likely to affect other components. In a fog environment, the limitation of computing resources further stresses the need of optimized adaptation actions that proactively change the multi-component application deployment.

Today's applications exploit elasticity to efficiently use resources and react to dynamic working conditions. The fog environment comes with a high number of heterogeneous resources, which often rely on a poor Internet connection. These features call for efficient solutions for determining an application placement, which should efficiently deal with the uncertainty of computing resources and incoming workloads. So far, there is only a limited number of fog-specific and mobility-aware solutions (e.g., [50]); most of the existing approaches solve the application deployment problem in a centralized manner. Moreover, mobility of fog resources have been so far scarcely studied, notwithstanding that it can lead to new applications and research directions, where mobile resources are opportunistically exploited to reduce the dependence over geographically bounded fixed fog resources.

Nowadays, orchestration tools present only a partially decentralized architecture, which could not be suitable to manage complex applications in a geographically distributed environment. In a master-worker architecture, collecting monitoring data on the master and dispatching the subsequent adaptation actions to the decentralized executors may introduce significant communication overhead. Furthermore, the master may easily become the system bottleneck when it has to control a multitude of entities scattered in a large-scale geo-distributed environment. To increase scalability, a hierarchical architecture could be investigated: exploiting the benefits of both centralized and decentralized architectures and policies, it could be well suited for controlling applications in a fog environment. The hierarchical control pattern revolves around the idea of a layered architecture, where each layer works with time scales and concerns separation. Given the great amount of interconnected devices and the system dynamism, also the deployment algorithms should be as scalable as possible.

The definition of multi-component applications that run on edge devices also exposes new security risks and trustiness issues, which should be addressed to boost the utilization of fog computing. Most of the existing deployment solutions neglect security-related issues. However, security is a first-class citizen in the fog environment: while allocating containers on fog resources, privacy constraints should be taken into account, as well as the security of the communication channels among the fog resources. The limited energy, network, and computing capacity of fog resources also requires to investigate whether existing fault-tolerance mechanisms can be adopted in the fog. Processing data at the network periphery, device (or connectivity) failures can easily compromise

the application availability and integrity. Considerations should be also made observing that nearby fog resources are more likely to fail simultaneously (e.g., due to connectivity outage).

Also monitoring and enforcing the QoS of multi-component applications is challenging in a fog environment. SLAs as defined today do not fit well in the emerging environment, where applications can exchange data across multiple service providers and, most importantly, can run on resources under different administrative domains. In a fog environment, it could be also difficult to collect application and service provisioning metrics, needed to evaluate the SLA fulfillment. The dynamism and heterogeneity of fog resources further increase the difficulty of controlling the application performance.

To conclude, we can observe that deployment solutions for fog environments are at their early stages; therefore, novel solutions that account for the distinctive fog computing features are needed. Methodologies that have been successfully adopted for cloud resources can be considered for the fog environments. For example, it would be interesting to further investigate the applicability of evolutionary algorithms, e.g., deep learning, genetic algorithms, and game theory, for adapting the deployment of microservice-based applications.

References

1. Abdelbaky, M., Diaz-Montes, J., Parashar, M., Unuvar, M., Steinder, M.: Docker containers across multiple clouds and data centers. In: Proc. of IEEE/ACM UCC 2015. pp. 368–371 (2015). https://doi.org/10.1109/UCC.2015.58
2. Addya, S.K., Turuk, A.K., Sahoo, B., Sarkar, M., Biswash, S.K.: Simulated annealing based VM placement strategy to maximize the profit for cloud service providers. Eng. Sci. Technol. Int J. **20**(4), 1249–1259 (2017)
3. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., Merle, P.: Autonomic vertical elasticity of Docker containers with ElasticDocker. In: Proc. of IEEE CLOUD '17. pp. 472–479 (2017). https://doi.org/10.1109/CLOUD.2017.67
4. Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., Merle, P.: Elasticity in cloud computing: State of the art and research challenges. IEEE Trans. Serv. Comput. **11**, 430–447 (2018). https://doi.org/10.1109/TSC.2017.2711009
5. Alam, M.G.R., Hassan, M.M., Uddin, M.Z., Almogren, A., Fortino, G.: Autonomic computation offloading in mobile edge for IoT applications. Future Gener. Comput. Syst. **90**, 149–157 (2019). https://doi.org/10.1016/j.future.2018.07.050
6. Ali-Eldin, A., Tordsson, J., Elmroth, E.: An adaptive hybrid elasticity controller for cloud infrastructures. In: Proc. of IEEE NOMS '12. pp. 204–212 (2012)
7. Arabnejad, H., Pahl, C., Jamshidi, P., Estrada, G.: A comparison of reinforcement learning techniques for fuzzy cloud auto-scaling. In: Proc. of IEEE/ACM CCGrid '17. pp. 64–73 (2017). https://doi.org/10.1109/CCGRID.2017.15
8. Arkian, H.R., Diyanat, A., Pourkhalili, A.: MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. J. Netw. Comput. Appl. **82**, 152–165 (2017). https://doi.org/10.1016/j.jnca.2017.01.012
9. Asnaghi, A., Ferroni, M., Santambrogio, M.D.: DockerCap: A software-level power capping orchestrator for Docker containers. In: Proc. of IEEE EUC '16 (2016)

10. Baresi, L., Guinea, S., Leva, A., Quattrocchi, G.: A discrete-time feedback controller for containerized cloud applications. In: Proc. of ACM SIGSOFT FSE '16. pp. 217–228 (2016). https://doi.org/10.1145/2950290.2950328
11. Barna, C., Khazaei, H., Fokaefs, M., Litoiu, M.: Delivering elastic containerized cloud applications to enable DevOps. In: Proc. of SEAMS '17. pp. 65–75 (2017)
12. Bellavista, P., Zanni, A.: Feasibility of fog computing deployment based on Docker containerization over RaspberryPi. In: Proc. of ICDCN '17. ACM (2017)
13. Bermbach, D., Pallas, F., Pérez, D.G., Plebani, P., Anderson, M., Kat, R., Tai, S.: A research perspective on fog computing. In: Service-Oriented Computing – ICSOC 2017 Workshops. pp. 198–210. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-91764-1_16
14. Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J.: Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn. **3**(1), 1–122 (2011)
15. Brogi, A., Forti, S., Guerrero, C., Lera, I.: Meet genetic algorithms in Monte Carlo: Optimised placement of multi-service applications in the fog. In: Proc. of IEEE EDGE '19. pp. 13–17 (2019). https://doi.org/10.1109/EDGE.2019.00016
16. Brogi, A., Forti, S., Guerrero, C., Lera, I.: How to place your apps in the fog: State of the art and open challenges. Softw. Pract. Exp. (2019). https://doi.org/10.1002/spe.2766
17. Buyya, R., Srirama, S.N., Casale, G., Calheiros, R., Simmhan, Y., Varghese, B., et al.: A manifesto for future generation cloud computing: Research directions for the next decade. ACM Comput. Surv. **51**(5), 105:1–105:38 (2019)
18. Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C., Buyya, R.: CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. Pract. Exp. **41**(1), 23–50 (2011)
19. Casalicchio, E., Perciballi, V.: Auto-scaling of containers: The impact of relative and absolute metrics. In: Proc. of IEEE FAS*W '17. pp. 207–214 (2017)
20. Casalicchio, E.: Container orchestration: A survey. In: Systems Modeling: Methodologies and Tools, pp. 221–235. Springer International Publishing, Cham (2019)
21. Chang, Z., Zhou, Z., Ristaniemi, T., Niu, Z.: Energy efficient optimization for computation offloading in fog computing system. In: Proc. of IEEE GLOBECOM '17 (2017). https://doi.org/10.1109/GLOCOM.2017.8254207
22. de Brito, M.S., Hoque, S., Magedanz, T., Steinke, R., Willner, A., Nehls, D., Keils, O., Schreiner, F.: A service orchestration architecture for fog-enabled infrastructures. In: Proc. of FMEC '17. pp. 127–132. IEEE (2017)
23. De Maio, V., Brandic, I.: Multi-objective mobile edge provisioning in small cell clouds. In: Proc. of ACM/SPEC ICPE '19. pp. 127–138. ACM (2019)
24. Elliott, D., Otero, C., Ridley, M., Merino, X.: A cloud-agnostic container orchestrator for improving interoperability. In: Proc of IEEE CLOUD '18. pp. 958–961 (2018). https://doi.org/10.1109/CLOUD.2018.00145
25. Garefalakis, P., Karanasos, K., Pietzuch, P., Suresh, A., Rao, S.: Medea: Scheduling of long running applications in shared production clusters. In: Proc. of EuroSys '18. pp. 4:1–4:13. ACM (2018). https://doi.org/10.1145/3190508.3190549
26. Gedeon, J., Brandherm, F., Egert, R., Grube, T., Mühlhäuser, M.: What the fog? edge computing revisited: Promises, applications and future challenges. IEEE Access **7**, 152847–152878 (2019). https://doi.org/10.1109/ACCESS.2019.2948399
27. Guan, X., Wan, X., Choi, B.Y., Song, S., Zhu, J.: Application oriented dynamic resource allocation for data centers using Docker containers. IEEE Commun. Lett. **21**(3), 504–507 (2017). https://doi.org/10.1109/LCOMM.2016.2644658

28. Guerrero, C., Lera, I., Juiz, C.: Genetic algorithm for multi-objective optimization of container allocation in cloud architecture. J. Grid Comput. **16**(1), 113–135 (2018). https://doi.org/10.1007/s10723-017-9419-x
29. Gupta, H., Vahid Dastjerdi, A., Ghosh, S.K., Buyya, R.: iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Softw. Pract. Exp. **47**(9), 1275–1296 (2017). https://doi.org/10.1002/spe.2509
30. Hoque, S., d. Brito, M.S., Willner, A., Keil, O., Magedanz, T.: Towards container orchestration in fog computing infrastructures. In: Proc. of IEEE COMPSAC 2017. vol. 2, pp. 294–299 (2017). https://doi.org/10.1109/COMPSAC.2017.248
31. Horovitz, S., Arian, Y.: Efficient cloud auto-scaling with SLA objective using Q-learning. In: Proc. of IEEE FiCloud '18. pp. 85–92 (2018)
32. Huang, Z., Lin, K.J., Yu, S.Y., Hsu, J.Y.j.: Co-locating services in IoT systems to minimize the communication energy cost. J. Innovation Digital Ecosyst. **1**(1), 47–57 (2014). https://doi.org/10.1016/j.jides.2015.02.005
33. Javed, A., Heljanko, K., Buda, A., Främling, K.: Cefiot: A fault-tolerant iot architecture for edge and cloud. In: Proc. of IEEE WF-IoT '18. pp. 813–818 (2018)
34. Jawarneh, I.M.A., Bellavista, P., Bosi, F., Foschini, L., Martuscelli, G., Montanari, R., Palopoli, A.: Container orchestration engines: A thorough functional and performance comparison. In: Proc. of IEEE ICC '19. pp. 1–6 (2019)
35. Kaewkasi, C., Chuenmuneewong, K.: Improvement of container scheduling for Docker using ant colony optimization. In: Proc. of KST '17. IEEE (2017)
36. Kaur, K., Dhand, T., Kumar, N., Zeadally, S.: Container-as-a-service at the edge: Trade-off between energy efficiency and service availability at fog nano data centers. IEEE Wireless Commun. **24**(3), 48–56 (2017)
37. Kayal, P., Liebeherr, J.: Autonomic service placement in fog computing. In: Proc. of IEEE WoWMoM '19. pp. 1–9 (2019)
38. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer **36**(1), 41–50 (2003). https://doi.org/10.1109/MC.2003.1160055
39. Khazaei, H., Bannazadeh, H., Leon-Garcia, A.: SAVI-IoT: A self-managing containerized IoT platform. In: Proc. of IEEE FiCloud '17. pp. 227–234 (2017)
40. Khazaei, H., Ravichandiran, R., Park, B., Bannazadeh, H., Tizghadam, A., Leon-Garcia, A.: Elascale: Autoscaling and monitoring as a service. In: Proc. of CASCON '17. pp. 234–240 (2017)
41. Kimovski, D., Ijaz, H., Saurabh, N., Prodan, R.: Adaptive nature-inspired fog architecture. In: Proc. of IEEE ICFEC '18. pp. 1–8 (2018)
42. Liu, L., Chang, Z., Guo, X., Mao, S., Ristaniemi, T.: Multiobjective optimization for computation offloading in fog computing. IEEE Internet Things J. **5**(1), 283–294 (2018). https://doi.org/10.1109/JIOT.2017.2780236
43. Lopes, M.M., Higashino, W.A., Capretz, M.A., Bittencourt, L.F.: Myifogsim: A simulator for virtual machine migration in fog computing. In: Proc. of IEEE/ACM UCC'17 Companion. pp. 47–52. ACM (2017)
44. Lorido-Botran, T., Miguel-Alonso, J., Lozano, J.A.: A review of auto-scaling techniques for elastic applications in cloud environments. J. Grid Comput. **12**(4), 559–592 (2014). https://doi.org/10.1007/s10723-014-9314-7
45. Mahmud, M., Srirama, S., Ramamohanarao, K., Buyya, R.: Quality of experience (QoE)-aware placement of applications in fog computing environments. J. Parallel Distrib. Comput. **123**, 190–203 (2018)
46. Mahmud, R., Kotagiri, R., Buyya, R.: Fog computing: a taxonomy, survey and future directions, pp. 103–130. Springer Singapore, Singapore (2018). https://doi.org/10.1007/978-981-10-5861-5_5

47. Mao, Y., Oak, J., Pompili, A., Beer, D., Han, T., Hu, P.: DRAPS: dynamic and resource-aware placement scheme for Docker containers in a heterogeneous cluster. In: Proc. of IEEE IPCCC '17 (2017). https://doi.org/10.1109/PCCC.2017.8280474
48. Mayer, R., Graser, L., Gupta, H., Saurez, E., Ramachandran, U.: Emufog: Extensible and scalable emulation of large-scale fog computing infrastructures. In: Proc. of IEEE FWC '17. pp. 1–6 (2017). https://doi.org/10.1109/FWC.2017.8368525
49. Mennes, R., Spinnewyn, B., Latré, S., Botero, J.F.: GRECO: A distributed genetic algorithm for reliable application placement in hybrid clouds. In: Proc. of IEEE CloudNet '16. pp. 14–20 (2016). https://doi.org/10.1109/CloudNet.2016.45
50. Mouradian, C., Kianpisheh, S., Abu-Lebdeh, M., Ebrahimnezhad, F., Jahromi, N.T., Glitho, R.H.: Application component placement in NFV-based hybrid cloud/fog systems with mobile fog nodes. IEEE J. Sel. Areas in Commun. **37**(5), 1130–1143 (2019). https://doi.org/10.1109/JSAC.2019.2906790
51. Mseddi, A., Jaafar, W., Elbiaze, H., Ajib, W.: Joint container placement and task provisioning in dynamic fog computing. IEEE Internet Things J. (2019)
52. Naas, M.I., Parvedy, P.R., Boukhobza, J., Lemarchand, L.: iFogStor: An IoT data placement strategy for fog infrastructure. In: Proc. of IEEE ICFEC '17. pp. 97–104 (2017). https://doi.org/10.1109/ICFEC.2017.15
53. Nardelli, M., Cardellini, V., Casalicchio, E.: Multi-level elastic deployment of containerized applications in geo-distributed environments. In: Proc. of IEEE FiCloud '18. pp. 1–8 (2018). https://doi.org/10.1109/FiCloud.2018.00009
54. Nardelli, M., Hochreiner, C., Schulte, S.: Elastic provisioning of virtual machines for container deployment. In: Proc. of ACM/SPEC ICPE '17 Companion. pp. 5–10 (2017). https://doi.org/10.1145/3053600.3053602
55. Netto, H.V., Luiz, A.F., Correia, M., de Oliveira Rech, L., Oliveira, C.P.: Koordinator: A service approach for replicating Docker containers in Kubernetes. In: Proc. of IEEE ISCC '18. pp. 58–63 (2018)
56. Nouri, S.M.R., Li, H., Venugopal, S., Guo, W., He, M., Tian, W.: Autonomic decentralized elasticity based on a reinforcement learning controller for cloud applications. Future Gener. Comput. Syst. **94**, 765–780 (2019)
57. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: Proc. of USENIX ATC '14. pp. 305–319 (2014)
58. Ouyang, T., Zhou, Z., Chen, X.: Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing. IEEE J. Sel. Area Comm. **36**(10), 2333–2345 (2018). https://doi.org/10.1109/JSAC.2018.2869954
59. Piraghaj, S.F., Dastjerdi, A.V., Calheiros, R.N., Buyya, R.: ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. Softw. Pract. Exp. **47**(4), 505–521 (2017)
60. Puliafito, C., Mingozzi, E., Longo, F., Puliafito, A., Rana, O.: Fog computing for the Internet of Things: A survey. ACM Trans. Internet Technol. **19**(2), 18:1–18:41 (2019). https://doi.org/10.1145/3301443
61. Rodriguez, M.A., Buyya, R.: Container-based cluster orchestration systems: A taxonomy and future directions. Softw. Pract. Exp. **49**(5), 698–719 (2019)
62. Röger, H., Mayer, R.: A comprehensive survey on parallelization and elasticity in stream processing. ACM Comput. Surv. **52**(2), 36:1–36:37 (2019)
63. Rossi, F., Cardellini, V., Lo Presti, F.: Elastic deployment of software containers in geo-distributed computing environments. In: Proc. of IEEE ISCC '19 (2019). https://doi.org/10.1109/ISCC47284.2019.8969607
64. Rossi, F., Nardelli, M., Cardellini, V.: Horizontal and vertical scaling of container-based applications using Reinforcement Learning. In: Proc. of IEEE CLOUD '19. pp. 329–338 (2019). https://doi.org/10.1109/CLOUD.2019.00061

65. Santos, J., Wauters, T., Volckaert, B., De Turck, F.: Towards network-aware resource provisioning in Kubernetes for fog computing applications. In: Proc. of IEEE NetSoft '19. pp. 351–359 (2019). https://doi.org/10.1109/NETSOFT.2019.8806671
66. Souza, V., Masip-Bruin, X., Marín-Tordera, E., Sànchez-López, S., Garcia, J., Ren, G., Jukan, A., Ferrer, A.J.: Towards a proper service placement in combined fog-to-cloud (F2C) architectures. Future Gener. Comput. Syst. **87**, 1–15 (2018)
67. Subashini, S., Kavitha, V.: A survey on security issues in service delivery models of cloud computing. J. Netw. Comput. Appl. **34**(1), 1–11 (2011)
68. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, 2 edn. (2018)
69. Tan, B., Ma, H., Mei, Y.: A hybrid genetic programming hyper-heuristic approach for online two-level resource allocation in container-based clouds. In: Proc. of IEEE CEC '19. pp. 2681–2688 (2019). https://doi.org/10.1109/CEC.2019.8790220
70. Tang, Z., Zhou, X., Zhang, F., Jia, W., Zhao, W.: Migration modeling and learning algorithms for containers in fog computing. IEEE Trans. Serv. Comput. **12**(5), 712–725 (2019). https://doi.org/10.1109/TSC.2018.2827070
71. Tesauro, G., Jong, N.K., Das, R., Bennani, M.N.: A hybrid Reinforcement Learning approach to autonomic resource allocation. In: Proc. of IEEE ICAC '06. pp. 65–73 (2006). https://doi.org/10.1109/ICAC.2006.1662383
72. Townend, P., Clement, S., Burdett, D., Yang, R., Shaw, J., Slater, B., Xu, J.: Improving data center efficiency through holistic scheduling in Kubernetes. In: Proc. of IEEE SOSE '19. pp. 156–166 (2019)
73. Wen, Z., Yang, R., Garraghan, P., Lin, T., Xu, J., Rovatsos, M.: Fog orchestration for Internet of Things services. IEEE Internet Comput. **21**(2), 16–24 (2017)
74. Weyns, D., Schmerl, B., Grassi, V., Malek, S., Mirandola, R., Prehofer, C., Wuttke, J., Andersson, J., Giese, H., Göschka, K.M.: On patterns for decentralized control in self-adaptive systems. In: Software Engineering for Self-Adaptive Systems II, LNCS, vol. 7475, pp. 76–107. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-35813-5_4
75. Wu, Y., Rao, R., Hong, P., Ma, J.: FAS: A flow aware scaling mechanism for stream processing platform service based on LMS. In: Proc. of ICMSS '17. pp. 280–284. ACM (2017). https://doi.org/10.1145/3034950.3034965
76. Xu, J., Chen, L., Ren, S.: Online learning for offloading and autoscaling in energy harvesting mobile edge computing. IEEE Trans. Cogn. Commun. Netw. **3**(3), 361–373 (2017). https://doi.org/10.1109/TCCN.2017.2725277
77. Yi, S., Hao, Z., Qin, Z., Li, Q.: Fog computing: Platform and applications. In: Proc. of HotWeb '15. pp. 73–78. IEEE (2015). https://doi.org/10.1109/HotWeb.2015.22
78. Yigitoglu, E., Mohamed, M., Liu, L., Ludwig, H.: Foggy: A framework for continuous automated IoT application deployment in fog computing. In: Proc. of IEEE AIMS '17. pp. 38–45 (2017). https://doi.org/10.1109/AIMS.2017.14
79. Zhao, D., Mohamed, M., Ludwig, H.: Locality-aware scheduling for containers in cloud computing. IEEE Trans. Cloud Comput. (2018)
80. Zhou, Z., Liu, P., Feng, J., Zhang, Y., Mumtaz, S., Rodriguez, J.: Computation resource allocation and task assignment optimization in vehicular fog computing: A contract-matching approach. IEEE Trans. Veh. Technol. **68**(4), 3113–3125 (2019)
81. Zhu, J., Chan, D.S., Prabhu, M.S., Natarajan, P., Hu, H., Bonomi, F.: Improving web sites performance using edge servers in fog computing architecture. In: Proc. of IEEE SOSE '13. pp. 320–323 (2013)
82. Zhu, Q., Agrawal, G.: Resource provisioning with budget constraints for adaptive applications in cloud environments. IEEE Trans. Serv. Comput. **5**(4), 497–511 (2012). https://doi.org/10.1109/TSC.2011.61