SOCA
DOI 10.1007/s11761-017-0219-8

SPECIAL ISSUE PAPER

Optimized IoT service placement in the fog

Olena Skarlat[1] · **Matteo Nardelli**[2] · **Stefan Schulte** [1] · **Michael Borkowski**[1] · **Philipp Leitner**[3]

Received: 15 March 2017 / Revised: 30 August 2017 / Accepted: 13 September 2017

Abstract The Internet of Things (IoT) leads to an ever-growing presence of ubiquitous networked computing devices in public, business, and private spaces. These devices do not simply act as sensors, but feature computational, storage, and networking resources. Being located at the edge of the network, these resources can be exploited to execute IoT applications in a distributed manner. This concept is known as fog computing. While the theoretical foundations of fog computing are already established, there is a lack of resource provisioning approaches to enable the exploitation of fog-based computational resources. To resolve this shortcoming, we present a conceptual fog computing framework. Then, we model the service placement problem for IoT applications over fog resources as an optimization problem, which explicitly considers the heterogeneity of applications and resources in terms of Quality of Service attributes. Finally, we propose a genetic algorithm as a problem resolution heuristic and show, through experiments, that the service execution can achieve a reduction of network communication delays when the genetic algorithm is used, and a better utilization of fog resources when the exact optimization method is applied.

Keywords Fog computing · Service placement · Resource provisioning · Internet of Things · Quality of service

✉ Olena Skarlat
o.skarlat@infosys.tuwien.ac.at

Matteo Nardelli
nardelli@ing.uniroma2.it

Stefan Schulte
s.schulte@infosys.tuwien.ac.at

Michael Borkowski
m.borkowski@infosys.tuwien.ac.at

Philipp Leitner
philipp.leitner@chalmers.se

[1] Distributed Systems Group, TU Wien, Vienna, Austria

[2] Department of Civil Engineering and Computer Science Engineering, University of Rome Tor Vergata, Rome, Italy

[3] Department of Computer Science and Engineering, Chalmers University of Technology, Gothenburg, Sweden

1 Introduction

Due to the wide adoption of virtualization and cloud technologies, companies and end users nowadays are able to lease computational resources on-demand [2]. As a second major technology trend, the advent of the Internet of Things (IoT) leads to an ever-growing presence of networked computing devices in public, business, and private spaces. These devices sense the environment, perform computations, and enact operations by working autonomously or by cooperating with other devices, being often enriched with Internet connectivity [3]. Furthermore, IoT devices can expose (for free or under incentives) their computing and storage capabilities.

Together, the proliferation of cloud and IoT technologies enables small-scale and large-scale smart environments and systems for various domains, such as smart healthcare, smart cities, smart energy grids, or smart factories [5]. However, from a technological point of view, the decentralized nature of the IoT does not match the rather centralized structure of the cloud. Today, IoT data are mostly produced in a distributed way, sent to a centralized cloud for processing, and then delivered to the distributed stakeholders or other distributed IoT devices, often located close to the initial data sources [5]. This centralized processing approach results in high communication delays and low data transfer rates between IoT devices as well as the IoT devices and potential users [4].

Published online: 04 October 2017

The support of decentralized processing of data on IoT devices in combination with the benefits of cloud technologies and virtualization has been identified as a promising approach to reduce communication overheads and data transfer times in the IoT [4,24]. To realize decentralized data processing, it is necessary to move parts of the computational and storage resources needed to process IoT data closer to data sources and service consumers (i.e., end users or data sinks) [10]. The underlying conceptual approach, i.e., the virtualization of IoT devices and the subsequent usage of the virtualized resources to process data, is known as *fog* or *edge computing* [4]. The community has not yet converged against crisp definitions of these terms [13]. In the following, we use the term fog computing for simplicity.

Fog computing mirrors the basic structure of the IoT, where a multitude of heterogeneous, networked devices cooperate [3,10]. *Fog cells*, i.e., single IoT devices coordinating a group of other IoT devices and providing virtualized resources, are located close to the edge of the network. These cells allow to execute IoT services to process data in close vicinity to the data sources and data sinks, instead of involving the cloud. This leads to lower communication delays, as well as to a better utilization of already available computational, storage, and networking resources in the fog. Potential use cases for fog computing include typical IoT scenarios, e.g., data prefiltering in Big Data scenarios [8], preprocessing of data streams from sensor nodes [19], or data processing in smart systems [26]. In many application areas, fog computing is used in combination with cloud computing, which can overcome the limited resource availability in the fog with resources acquirable on-demand.

While the basic idea and theoretical foundations of fog computing are already established [4,10], there is still a lack of concrete solutions on resource provisioning. Apart from the question of how to virtualize the resources offered by IoT devices, another major barrier for the uptake of fog computing is the question of how to distribute IoT services on available fog resources.

Hence, in this paper, we describe a conceptual framework for resource provisioning and service placement in the fog. For this, we apply the concept of *fog colonies*. Fog colonies are micro-data centers made up from an arbitrary number of fog cells. As in a cloud data center, within a fog colony, services and data can be distributed and shared between the single cells. The operational purpose of fog colonies is the cooperative execution of IoT applications, which are composed of a sequence of services, e.g., as a distributed data flow (DDF) [16]. Thus, fog colonies aim to move from centralized cloud-based processing to a decentralized processing network that includes networked IoT devices and allows cloud offloading and multi-cloud deployment.

Based on this concept of fog colonies, we are able to orchestrate fog cells and to provide a suitable service placement approach, i.e., a solution on how to place services on virtualized resources in a fog landscape. For this, we formalize an optimization problem that aims to adhere to the deadlines on deployment and execution time of applications and to maximize the utilization of existing resources in the fog. To solve the proposed optimization problem, we apply different approaches, namely the exact optimization method and its approximation through a greedy first fit heuristic and a genetic algorithm. Also, we compare the results to a classical approach that neglects fog resources and runs all services in a centralized cloud. The goal of the evaluation is to identify the best approach to solve the proposed optimization problem in terms of resulting Quality of Service (QoS) (i.e., application response times), QoS violations (i.e., application deadline violations), and cost.

This paper extends our previous work [30] (i) by providing a motivational scenario based on the application of fog computing for Cyber-Physical Systems (CPS), (ii) by adding a formal model of the fog landscape and IoT applications to be executed in that landscape, (iii) by providing a formal definition of the Fog Service Placement Problem (FSPP), and (iv) by implementing heuristics to solve the FSPP. With respect to [30], we have also replaced the simulation environment in favor of iFogSim [17].

The remainder of this paper is organized as follows: After motivating our work in Sect. 2, we describe the architecture of our conceptual fog computing framework in Sect. 3. Next, in Sect. 4, we formulate the envisioned optimization problem. We solve the problem by various methods and evaluate the results in Sect. 5. Afterwards, we discuss the state-of-the-art work in the area of the fog computing frameworks, resource provisioning and service placement in Sect. 6. Finally, we conclude the paper in Sect. 7.

2 Motivational scenario

To motivate our work, we use a scenario from the European H2020 Factories of the Future project Cloud-based Rapid Elastic Manufacturing (CREMA) [27,28]. The goal of the CREMA project is to realize *Cloud Manufacturing*, which is a paradigm to achieve the objectives of the Industrial Internet (also known as Industry 4.0) based on principles from cloud computing, Business Process Management, and the IoT [34]. Mapping these principles to the manufacturing domain allows a high level of flexibility and interoperability by integrating single distributed steps of manufacturing processes from different organizations as if the complete process was carried out on the same shop floor.

Various support systems for manufacturing, e.g., Enterprise Resource Planning or Manufacturing Execution Systems, are integrated in Cloud Manufacturing processes. In addition, IoT technologies like CPS, smart objects, or

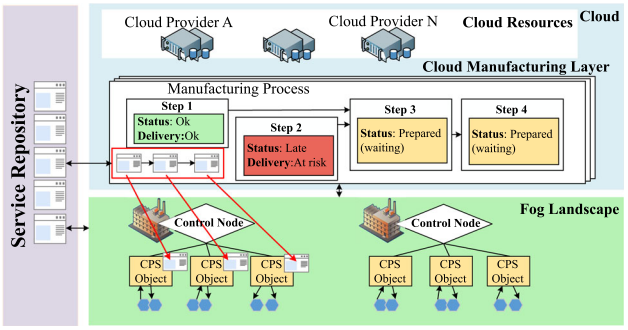

Fig. 1 Implementing cloud manufacturing with fog computing

sensor networks emit vast amounts of data in manufacturing scenarios. This data is consumed by distributed Cloud Manufacturing stakeholders. So far, cloud computing has been named as the primary enabler of Cloud Manufacturing with regard to the provisioning of computational resources [34,36]. However, manufacturers tend to create private clouds to process and store data within their own premises [15,22].

With the advent of fog computing, it is possible to go one step further using IoT resources instead of private cloud-based resources for some tasks in manufacturing processes. To realize fog computing in Cloud Manufacturing, the resources of IoT devices available at the shop floor (e.g., smart objects, sensor nodes, CPS, gateways) need to be virtualized and subsequently integrated into a fog landscape. Based on these virtualized resources, it is possible to deploy services belonging to manufacturing processes in the fog.

As an example, we consider a monitoring process that runs several services aiming to collect machine and software component data and monitor communications among multiple sources, customers, and equipment. Taking into account the amount of transferred data and the cost of the cloud, the according services from the process are placed on the computational resources in the available infrastructure of the shop floor (see Fig. 1). Thus, the major computationally intensive part of the considered process is located close to the data sources. The outcome of the monitoring process is then sent to the management system of the shop floor, which, in fact, can be located both on the local infrastructure and in the cloud. This approach facilitates the usage of IoT-based computational resources, eases the control and monitoring of online manufacturing devices, allows dynamic reconfigurations of the software infrastructure, and reduces the cost of using cloud resources [25]. However, to realize this setting, efficient strategies for defining the placement of manufacturing services in the fog are needed.

It should be noted that the example provided in this section is illustrative only. Fog computing is a promising approach in a number of IoT scenarios (e.g., in smart city, smart mobility, or smart grid scenarios), where a large volume of data needs to be processed, and the decentralized computation can be helpful to improve application performance and to relieve the network stress. Therefore, fog computing is well suited to achieve the overall objectives of such 'smart systems', i.e., to connect and process data from distributed data sources while using already existing computational resources, decreasing processing latency, and offering means to process data on-site in a privacy-aware manner.

3 Conceptual fog computing framework

In this section, we present the architecture of the fog computing framework depicted in Fig. 2. The framework enables the enactment of IoT services in an arbitrary fog landscape. This allows to optimize resource provisioning and service placement in the fog, as discussed in Sect. 4.

Following the basic structure of fog computing as presented in [4,10], we allow for resource provisioning and service placement in both the cloud and fog. To achieve this, a *cloud-fog control middleware* is introduced, which controls all fog cells. IoT applications have to be executable without any involvement of the cloud to reduce communication delays and cost. Hence, another level of control is necessary, which needs to run exclusively in the fog. For this, we introduce *fog orchestration control nodes*, which are a specific kind of fog cells. A fog orchestration control node manages a number of fog cells or other control nodes connected to it. We call such structures fog colonies. In our conceptual framework, we support a hierarchy of fog colonies with a head element in the cloud, i.e., the cloud-fog control middleware. The further layers of the hierarchy are the fog orchestration control nodes, the fog cells, and finally the IoT devices at the very bottom of the hierarchy (see Fig. 2).

In the following, we use the notion of *IoT applications* for tasks which need to be accomplished using computational resources provided by the cloud or by the fog. The details on what is an IoT application are given in Sect. 4.1.2. IoT applications are composed of a set of services to be executed. This generic definition of applications allows to model different

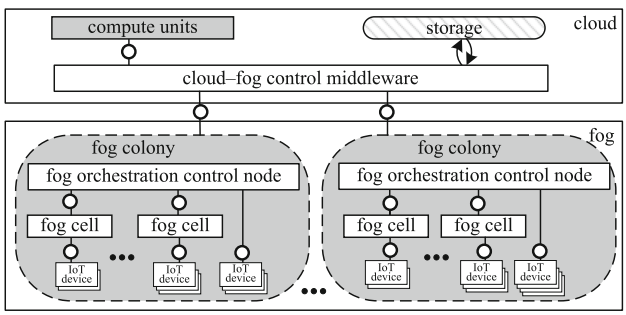

Fig. 2 Fog computing framework overview

kinds of IoT applications, e.g., data stream processing applications, MapReduce jobs, distributed data store services, and the processes presented in Sect. 2. In the following subsections, we will present the top-level components of the fog computing framework in more detail.

3.1 Cloud-fog control middleware

The cloud-fog control middleware is a central unit that manages the execution of applications in the cloud, and supports the underlying fog landscape. The middleware performs cloud resource provisioning and placement for services that are not delay-sensitive or cannot be executed in the fog, e.g., resource-intensive Big Data analysis. If necessary, the middleware performs global optimization of underlying fog colonies by restructuring them. For this, the cloud-fog control middleware is supplemented by both the means to control the cloud and the means to manage the underlying fog colonies. Such control is performed continuously or on-demand, depending on system events, e.g., if new fog devices appear which can be used to deploy fog cells, or to recover after faults of fog cells. Importantly, the cloud-fog control middleware can overrule fog orchestration control nodes in fog colonies, but the latter may also act autonomously in the case that no middleware is available.

3.2 Fog cells

Fog cells are software components running on IoT devices. They serve as access points allowing to control and monitor the underlying IoT devices, e.g., sensor and actuator nodes. They may interact with an arbitrary number of other IoT devices. However, in practice, the number of devices to be controlled by a fog cell is limited by its computational resources.

Each fog cell consists of the following components (see Fig. 3). The *listener* receives requests for service placement from the fog orchestration control node. The *monitor* observes service executions in the compute unit. The *database* stores data about received requests, the current system state of the fog cell, i.e., available resources, and monitoring data. The *fog action control* performs actions according to the service placement plan produced by the fog orchestration control node, e.g., to deploy and start a particular service (see Sect. 3.3). The *compute unit* provides the actual computational resources for the deployment and execution of services.

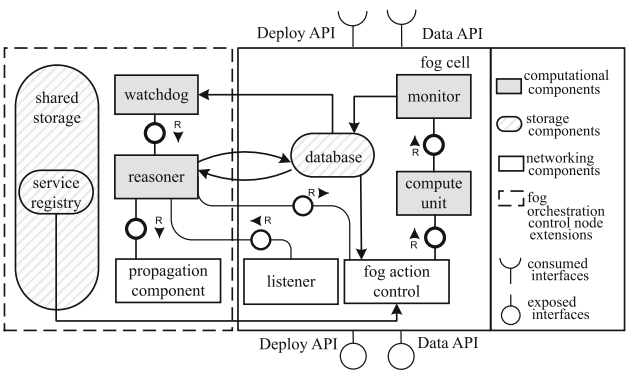

Fig. 3 Fog cell and fog orchestration control node architecture

Fog cells expose REST APIs for data transfer and control actions. The *Data API* allows basic CRUD operations over the data stored within a fog cell, and the *Deploy API* allows performing control actions for services running in the fog cell, i.e., instantiating, starting, stopping, and deleting services [4]. To become a member of a fog colony, a fog cell needs to use the Data and Deploy APIs of the corresponding fog orchestration control node, and at the same time expose its own Data and Deploy APIs for other fog cells.

3.3 Fog orchestration control nodes

The main task of a fog orchestration control node is to support a fog colony. Each fog colony features exactly one head fog orchestration control node which is a powerful fog cell with extended functionality for executing services and managing the resources offered by subordinated fog cells. Opposed to fog cells, fog orchestration control nodes can receive requests for execution of IoT applications from users. Additionally, the control node is able to propagate requests for service placement to the cloud-fog control middleware or to other fog colonies (via their fog orchestration control nodes), when services cannot be handled by the current fog colony. For this, a service placement mechanism is necessary to identify how services can be delegated in the entire fog landscape. Fog orchestration control nodes (i) perform infrastructural changes in the fog colony, (ii) analyze resource utilization within the colony, (iii) calculate a service placement plan to allocate resources for services, and (iv) monitor fog cells. An approach to optimize service placement is described in Sect. 4.

On the left-hand side of Fig. 3, the extensions needed for fog orchestration control nodes are depicted. The *reasoner* component produces a service placement plan that determines where each service of requested IoT applications needs to be deployed. The reasoner also gets information about the system state from the adjacent fog cells, i.e., available fog colony resources, controls the connected fog cells, and plans infrastructural changes in the fog colony, if necessary. If the considered fog colony does not provide sufficient resources or further processing is needed, such requests are separated and propagated to other fog colonies by the *propagation component* via the fog orchestration control node. The *watchdog* receives up-to-date information about the utilization of the

connected fog cells. It observes monitoring data from the database and compares that data to the expected QoS levels, i.e., measures the consumption of computational resources in the fog colony as well as QoS parameters, e.g., the execution time. This information influences the decision-making in the reasoner, i.e., if any expected QoS level is exceeded, the notification is sent to the reasoner to replan the service placement. The *service registry* hosts service implementations and enables the fog action control to search for services and to deploy them on the fog cell compute units. The service registry is located in the storage unit of control nodes, since storing service implementations is resource-consuming. In addition to the fog cell's functionality, the *listener* is extended to receive requests for application execution. These requests can be newly submitted to the listener of the fog orchestration control node, or propagated from another fog colony in the fog landscape. The *database* stores the service placement plan produced by the reasoner. The *fog action control* performs the service placement according to the plan.

It should be noted that an alternative approach would be a decentralized orchestration of fog cells in a fog colony, i.e., without a centralized control node. While this leads to higher fault tolerance, it also involves extensive coordination and voting between the involved fog cells. Therefore, we opt for a more centralized approach in this work. However, we still foresee that another fog cell becomes the fog orchestration control node in a fog colony, if necessary, e.g., in case the original fog orchestration control node fails.

4 Service placement in the fog

As discussed in Sect. 3, it is necessary to provide the fog orchestration control node of each fog colony as well as the cloud-fog control middleware with means to analyze submitted requests for IoT application executions and place according services onto specific virtualized resources. For this, the fog orchestration control node needs a complete overview of the system state of its fog colony and data about neighbor colonies. With this system state as input, the reasoner is able to compute a service placement plan and to send requests for service placement to particular fog resources. As stated in Sect. 3.3, we assume that each fog colony is autonomous, i.e., the cloud-fog control middleware is only involved if a fog colony needs additional cloud resources.

As in the field of cloud resource optimization, manifold goals for resource provisioning and optimal service placement are possible, e.g., time, cost, or energy efficiency optimization [1,24,31]. In the following, the goal of service placement optimization is to maximize the utilization of a fog landscape and to adhere to the QoS expectations of applications, i.e., deadlines on the application execution time. First, this means that the computational resources offered by fog cells have to be utilized as much as possible. Second, the data needed and sourced within a particular fog colony have to be handled by that particular colony, if possible. If the fog colony is overloaded, another fog colony in the fog landscape has to be utilized, e.g., the closest neighbor colony. If the utilization of the closest neighbor colony for requested service placements results in violations of deadlines of applications, then the corresponding services have to be deployed in the cloud.

Together, these goals form the foundation for an optimization problem, as will be presented in the upcoming subsections. Based on the solutions to this optimization problem, the fog orchestration control node either instantiates services on particular fog cells, or propagates requests for specific service execution to the closest neighbor colony or to the cloud. We refer to this problem as the Fog Service Placement Problem (FSPP). In addition to the optimization problem presented in Sect. 4.2, we design and implement a genetic algorithm (GA) as a heuristic approach to solve this problem.

4.1 System model

In order to formulate the FSPP, we define a model of the resources in a fog landscape in Sect. 4.1.1 and a model of IoT applications to be executed in the fog in Sect. 4.1.2. Afterwards, we formulate the FSPP in Sect. 4.2. In Table 1, we outline the notation used in the FSPP.

4.1.1 Fog landscape

The basic entity in a fog landscape is a fog colony. Each fog colony consists of 'thin' IoT devices which do not possess any computational power, i.e., sensors and actuators, and 'fat' IoT devices which possess computational power and can be virtualized, i.e., fog cells and fog orchestration control nodes. A fog orchestration control node F represents the head of a colony and oversees the service placement and execution. F controls subordinate fog cells, which are identified as the set $Res(F)$. Fog cells $f^j \in Res(F)$ are equipped with sensors and actuators. All the communication in the fog colony is performed through the fog orchestration control node. The communication link between the fog orchestration control node and a particular fog cell f^j is identified by a non-negligible delay d^j. The CPU utilization of the fog orchestration control node and fog cells is indicated by U^F and U^{f^j}, respectively. Analogously, M^F and M^{f^j} refer to the RAM capacities of the fog orchestration control node F and a fog cell f^j, and S^F and S^{f^j} refer to storage capacities of F and f^j, respectively.

Table 1 FSPP notation

	Notation	Definition
Time	t	Current time
	τ	Interval between two rounds of the FSPP (in s)
Fog Landscape	R	Cloud
	F	Fog orchestration control node of the fog colony
	N	Closest neighbor fog orchestration control node
	$Res(F)$	Fog cells in the fog colony
	U^F	CPU capacity of F
	M^F	RAM capacity of F
	S^F	Storage capacity of F
	f^j	Fog cell in a fog colony
	U^{f^j}	CPU capacity of f^j
	M^{f^j}	RAM capacity of f^j
	S^{f^j}	Storage capacity of f^j
	d^{f^j}	Link delay between F and f^j
	d^R	Link delay between F and R
	d^N	Link delay between F and N
Applications	A	Set of applications to be executed
	A_k	Application
	D_{A_k}	Application deadline
	w_{A_k}	Deployment time of an application
	m_{A_k}	Makespan of an application
	r_{A_k}	Response time of an application
	a_i	Service in an application
	U_{a_i}	CPU demand of a service
	M_{a_i}	RAM demand of a service
	S_{a_i}	Storage demand of a service
	m_{a_i}	Makespan of a service
	$Res^{a_i}(F)$	All fog cells able to host a service a_i

For each fog colony, we consider a neighborhood of fog colonies and identify the most efficient neighbor colony. For the purpose of the FSPP, the criterion for that is to find the closest neighbor colony by comparing communication link delays between the fog orchestration control node F and control nodes of all other neighbor colonies. We indicate N as the fog orchestration control node of the closest neighbor colony. The communication link between F and N is characterized by a non-negligible delay d^N.

A cloud-fog middleware is responsible for the utilization of resources of the cloud R; hence, it represents a communication bridge between fog colonies (by means of their fog orchestration control nodes) and the cloud. Even though the fog colonies are assumed to be autonomous, the cloud-fog control middleware can overrule fog orchestration control nodes if needed, e.g., in the case of a failure or fog colony overload. The communication link between F and R is characterized by a non-negligible delay d^R.

4.1.2 IoT applications and services

The fog colony controlled by F receives m requests for application executions. Let A be a set of m IoT applications to be executed. The IoT application follows the DDF deployment model [16]. Each $A_k \in A$ contains a set of services, where each service $a_i \in A_k$ has to be placed on a virtualized (cloud or fog) computational resource. The modeled applications require that each composing service has to be deployed before the application can start its execution, i.e., before data starts flowing between services. The application response time r_{A_k} is a sum of (i) the overall makespan duration m_{A_k} and (ii) the overall deployment time w_{A_k} of the application. The overall makespan duration of the application m_{A_k} results from the communication delays among fog devices and from the makespan duration of each application service $a_i \in A_k$. The makespan duration m_{a_i} of a service a_i is the time interval between starting and finishing the execution of a_i. The overall deployment time of the application is w_{A_k}, and it depends on the current deployment time $w^t_{A_k}$ of the application and the additional time that occurs when any service in the application is propagated to the closest neighbor colony. The current deployment time indicates the time elapsed since the request for execution of the application, e.g., when any service from this application was propagated from a neighbor colony, this time is stored in $w^t_{A_k}$. The details about the response time estimation are provided in Sect. 4.2. We assume the deployment time of application w_{A_k} already accounts for all the management operations which define and enact service placement. Each application possesses a deadline for deployment and execution D_{A_k} defined by users of the application. Each service a_i is defined by its demands of CPU U_{a_i}, RAM M_{a_i}, and storage S_{a_i}, and by a service type. The service type indicates that a service a_i has to be placed on specific kinds of virtualized resources in the fog landscape. Without loss of generality, we consider three different service types, namely sensing, processing, and actuating services. There is no need to introduce notation for service types, as they are accounted for in the system model (see variables in Sect. 4.2).

4.2 Optimization problem

Based on the introduced entities of the fog landscape and IoT applications, we define the FSPP. The FSPP aims to perform an optimal placement of services on resources in a fog landscape. The solution of the FSPP is a service placement plan that contains mapping decisions which place each service either on fog cells or on the fog orchestration control node, and propagation decisions, which propagate the placement requests to the closest neighbor colony or to the cloud. The fog orchestration control node periodically solves the FSPP, every τ time units, to compute the placement of applications

requested for execution. When requests are submitted to the fog colony, their placement is postponed to the next closest optimization round of the FSPP, indicated by its start time t. When all the application services are correctly assigned, the application can start its execution.

Problem variables First, we define the decision variables which form a service placement plan. The binary variables $x_{a_i}^{f^j}$ and $x_{a_i}^{F}$ denote whether service a_i has to be placed on a fog cell f^j or on the fog orchestration control node F, respectively. The binary variables $x_{a_i}^{N}$ and $x_{a_i}^{R}$ indicate whether service a_i has to be propagated to the closest neighbor colony, indicated by N, or to the cloud R, respectively. For each service a_i, we consider a set of fog cells $Res^{a_i}(F)$, with $Res^{a_i}(F) \subseteq Res(F)$, which can host and execute a_i. This definition allows to easily account for the compatibility between the service type (i.e., sensing, processing, and actuating services) and the allocated resource. In other words, all the fog cells in $Res^{a_i}(F)$ are ready to execute the service a_i.

Goal function The objective of the FSPP is to maximize the number of service placements to fog resources (rather than to cloud ones), while satisfying the requirements of each application, as in (1).

$$\max \sum_{A_k}^{A} \left(P(A_k) \cdot \sum_{a_i}^{A_k} \left(\sum_{f^j}^{Res^{a_i}(F)} x_{a_i}^{f^j} + x_{a_i}^{F} + x_{a_i}^{N} \right) \right) \tag{1}$$

Propagating any service to the cloud or to the closest neighbor colony introduces communication and deployment delays, which can be detrimental when the application response time is approaching the deadline. Therefore, the application requests for placement are prioritized using the coefficient $P(A_k)$ defined in (2). $P(A_k)$ depends on the distance between the application deadline D_{A_k} and its (already passed) deployment time in t, denoted as $w_{A_k}^{t}$. The latter accounts for the time waited by the application before it is correctly assigned to the computational resources. The key idea is to first grant fog resources to the applications that have spent a high waiting time for deployment, with respect to the application deadline (which cannot be violated, as specified in the following constraints).

$$P(A_k) = \frac{1}{D_{A_k} - w_{A_k}^{t}} \tag{2}$$

Constraints First, CPU, RAM, and storage demands of services placed on certain fog resources must not exceed the available resources of those devices, as defined in (3) and (4). The equations allow also to consider a percentage of system resources $\gamma \in [0, 1]$ that should be preserved free on each fog cell (e.g., to allow for its operational maintenance).

$$\sum_{A_k}^{A} \sum_{a_i}^{A_k} C_{a_i} x_{a_i}^{f^j} \leq \gamma C^{f^j}, \quad \forall f^j \in Res^{a_i}(F), C = \{U, M, S\} \tag{3}$$

$$\sum_{A_k}^{A} \sum_{a_i}^{A_k} C_{a_i} x_{a_i}^{F} \leq \gamma C^{F}, \quad C = \{U, M, S\} \tag{4}$$

Second, it has to be ensured that the response time r_{A_k} of each application does not violate the deadline D_{A_k} of that application, as defined in (5).

$$r_{A_k} \leq D_{A_k}, \quad \forall A_k \in A \tag{5}$$

The application response time r_{A_k} is calculated as the sum of the overall makespan duration of the application m_{A_k} and its deployment time w_{A_k}, according to (6).

$$r_{A_k} = m_{A_k} + w_{A_k} \tag{6}$$

The makespan duration m_{A_k} accounts for the time needed to execute all the application services (in the fog landscape or in the cloud), together with the time needed to perform communications among services (which traverse the fog orchestration control nodes). Therefore, the application makespan duration m_{A_k} is a sum of communication link delays in each case of placement of a service on a particular virtualized resource multiplied by an according decision variable as in (7). The factors $d(a_i, f^j)$, $d(a_i, F)$, $d(a_i, R)$, and $d(a_i, N)$ represent the makespan duration of a service a_i when it is executed on the fog cell f^j, the control node F, the cloud R, and the closest neighbor colony N, respectively. These factors are formalized in (8)–(11). Details on how the estimation of the response time is performed can be seen further in Example 1.

$$m_{A_k} = \sum_{a_i}^{A_k} \left(\sum_{f^j}^{Res^{a_i}(F)} d(a_i, f^j) x_{a_i}^{f^j} + d(a_i, F) x_{a_i}^{F} + \right. \\ \left. + d(a_i, R) x_{a_i}^{R} + d(a_i, N) x_{a_i}^{N} \right) \tag{7}$$

$$d(a_i, f^j) = d^{f^j} + m_{a_i} \tag{8}$$

$$d(a_i, F) = m_{a_i} \tag{9}$$

$$d(a_i, R) = 2d^{R} + m_{a_i} \tag{10}$$

$$d(a_i, N) = 2d^{N} + m_{a_i} \tag{11}$$

The application deployment time w_{A_k} considers the time elapsed before each service is correctly placed on the computational resources (either fog or cloud). Specifically, w_{A_k} accounts for the already passed plus the additional *expected* deployment time that appears if any service $a_i \in A_k$ is

propagated to the closest neighbor colony. To account for this additional deployment time, we rely on the auxiliary variable y_{A_k}. Let $y_{A_k} = 1$ if at least one service in A_k is propagated to the closest neighbor for colony, $y_{A_k} = 0$ otherwise. We formalize the application deployment time w_{A_k} as follows:

$$w_{A_k} = w^t_{A_k} + \tau y_{A_k} + \overline{T}^t_{w_N} y_{A_k} \tag{12}$$

The second and third terms provide a contribution only if $y_{A_k} = 1$. Specifically, τy_{A_k} models that, by propagating any service to the closest neighbor colony, the application deployment is postponed to at least τ time units, whereas $\overline{T}^t_{w_N} y_{A_k}$ models the additional deployment time spent in the closest neighbor colony before the application execution. Indeed, the closest neighbor colony can decide to place the service as soon as possible or to further postpone its execution by propagating the request to its related neighbor colony. To calculate the expected deployment time in the closest neighbor colony $\overline{T}^t_{w_N}$ requires to view forward in time, which is not feasible in practice. Therefore, we estimate $\overline{T}^t_{w_N}$ relying on historical data. $\overline{T}^t_{w_N}$ is obtained as the moving average of parameter α on the latest sampled deployment time $T^{t-\tau}_{w_N}$ per each service propagated to the closest neighbor colony:

$$\overline{T}^t_{w_N} = \alpha T^{t-\tau}_{w_N} + (1-\alpha)\overline{T}^{t-\tau}_{w_N} \tag{13}$$

where $\alpha \in [0, 1]$ represents the discounting factor of the moving average, $T^{t-\tau}_{w_N}$ is the recorded deployment time of the service forwarded to N at time $t - \tau$, and $\overline{T}^{t-\tau}_{w_N}$ is the expected average deployment time in N as estimated in $t - \tau$.

Provided that $|A_k|$ is the cardinality of A_k, this variable y_{A_k} is modeled by means of the equations (14) and (15) as a logical OR among the placement variables $x^N_{a_i}$ with $a_i \in A_k$. It has to be noted that y_{A_k} does not consider the case when a service is propagated to the cloud, where the service is deployed immediately, without waiting for the optimization procedure as in the closest neighbor fog colony.

$$y_{A_k} \leq \sum_{a_i}^{A_k} x^N_{a_i}, \quad \forall A_k \in A \tag{14}$$

$$y_{A_k} \geq \frac{\sum_{a_i}^{A_k} x^N_{a_i}}{|A_k|}, \quad \forall A_k \in A \tag{15}$$

Finally, we define that each service a_i can be placed or propagated on exactly one computational resource, i.e., fog cell f^j, fog orchestration control node F, the closest neighbor control node N, or to the cloud R:

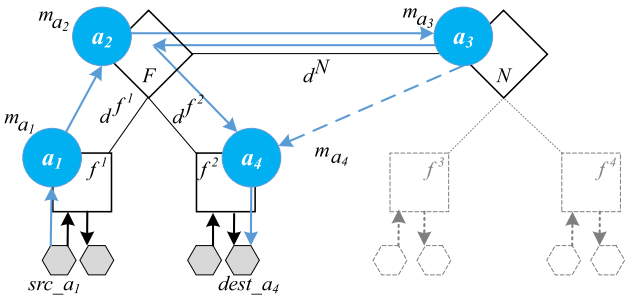

Fig. 4 Example of service placement

$$\sum_{f^j}^{Res^{a_i}(F)} \left(x^{f^j}_{a_i}\right) + x^F_{a_i} + x^N_{a_i} + x^R_{a_i} = 1\,, \quad \forall a_i \in A_k, \forall A_k \in A \tag{16}$$

Example 1 To clarify the calculation of the response time of an application, we provide an example for an application A_1 that consists of four services $A_1 = \{a_1, a_2, a_3, a_4\}$ and is distributed between two fog colonies. Let the assignments to the fog devices be as follows: Service a_1 is placed on f^1, a_2 is placed on F, a_3 is propagated to the closest neighbor colony N, and a_4 is placed on f^2 (Fig. 4). In this example, we assume that the application had no previous deployment time, so that $w^t_{A_1} = 0$, and in the closest neighbor colony with the control node N the average deployment time is $\overline{T}_{w_N}(t)$. The response time r_{A_1} is calculated according to (6)–(11) as shown in (17). To be able to calculate estimations of according factors of data transfers between services in an application, as necessary for (8)–(11), we use the notion of triangular inequality in the network [9], which helps to find an approximate distance and according delay between two network locations, which is assumed to be bigger than the direct delay between two locations (see Fig. 4).

$$\begin{aligned} r_{A_1} = {} & d^{f^1} + m_{a_1} + m_{a_2} + d^N + m_{a_3} + \\ & + d^N + d^{f^2} + m_{a_4} + \overline{T}_{w_N}(t) + \tau \end{aligned} \tag{17}$$

Domain definition The domain definition for the FSPP decision variables results from (18)–(22):

$$x^{f^j}_{a_i} \in \{0, 1\}, \quad \forall a_i \in A_k, \forall A_k \in A, \forall f^j \in Res^{a_i}(F) \tag{18}$$

$$x^F_{a_i} \in \{0, 1\}, \quad \forall a_i \in A_k, \forall A_k \in A \tag{19}$$

$$x^R_{a_i} \in \{0, 1\}, \quad \forall a_i \in A_k, \forall A_k \in A \tag{20}$$

$$x^N_{a_i} \in \{0, 1\}, \quad \forall a_i \in A_k, \forall A_k \in A \tag{21}$$

$$y_{A_k} \in \{0, 1\}, \quad \forall A_k \in A \tag{22}$$

4.3 Genetic algorithm

In general, the service placement problem has been shown to be NP-complete [21]. Hence, we present a heuristic to solve the FSPP. The choice to design and implement a GA to solve the FSPP was based on the popularity of GAs in solving service composition problems in cloud-based environments [35]. The main advantage of GAs is that they allow to investigate a large search space and provide a viable qualitative solution in polynomial time [38,39].

As the name implies, GAs intend to mimic evolutionary processes [33]. One iteration of a GA implies the application of three genetic operators in one generation, i.e., *selection*, *crossover*, and *mutation*. A generation consists of a population of individuals, where each individual is represented by its own chromosome. Each chromosome consists of genes. The selection operator considers the population of individuals in the current generation and chooses the best individuals to let them reproduce and have an offspring, i.e., new individuals, which form the next generation. The selection operator assesses the fitness function of the chromosome of each individual. The fitness function shows the level of 'health' of a chromosome, and is calculated based on a goal function and the constraints of an optimization problem. After a certain percentage of individuals in a population has been chosen for reproduction, the crossover operator starts swapping genes of chromosomes of chosen individuals to create an offspring. The elite of each generation, i.e., the individuals with the best fitness values, go to the next generation unaltered. The mutation operator performs a mutation of a certain number of individuals from the new offspring to support the diversity of generations, i.e., the mutation changes a random number of genes in the chromosome of an individual. Consequently, an old generation is evolving into a new generation with a population filled by both the unaltered elite and offspring. The algorithm repeats this process until a certain stopping condition is fulfilled. As stopping condition, various criteria may be used, e.g., the number of generations, the moment when the fitness function has no improvement with regard to a certain tolerance value, or a specific moment in time. In the following, we describe the concrete implementation of the GA to solve the FSPP.

Chromosome representation In our implementation, the chromosome encoding is a vector, which represents a service placement plan, i.e., a solution to the FSPP. The length of the chromosome is the total number of services from all applications, which were requested for execution at time t, i.e., $\sum_{A_k}^{A} |A_k|$. Therefore, each service can be easily identified by the position of the according gene in the chromosome. Each gene in a chromosome denotes a certain placement of a service on a specific fog resource. A gene is an integer value corresponding to the unique identifier of the computational resource (i.e., fog cell, fog orchestration control node, the closest neighbor colony, or the cloud). A position of a gene in a chromosome along with its integer value represents the service placement on the resource (see Fig. 5). Such chromosome encoding ensures the placement of all services.

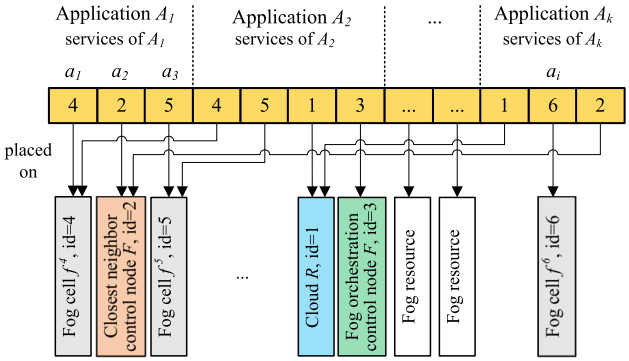

Fig. 5 Chromosome representation

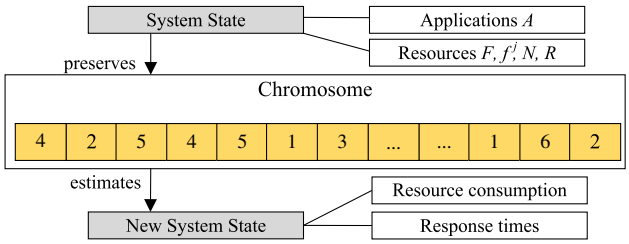

Fig. 6 Chromosome contents

Apart from encoding the placement, an estimated system state can be derived from the information in the chromosome (Fig. 6). This information includes the used CPU, RAM, and storage capacities of fog resources (both fog cells and fog orchestrator control nodes), and according response times of applications, which will be obtained if the service placement according to the considered chromosome representation is enacted. The estimated system state is calculated based on the genes of the chromosome. This calculation is performed whenever the chromosome is created, i.e., also after crossovers and mutations.

Fitness function The fitness function for each chromosome is calculated based on the principle of encouragement if the chromosome fulfills the constraints of the FSPP, and of punishment in the other case [37]. We consider three types of constraints: (i) a set of constraints Ψ on capacities of CPU, RAM and storage resources, (ii) a set Γ of implicit binary constraints derived from the model's goal function, i.e., conformance to service types, indications whether services are placed on the fog resources, and (iii) a set of constraints Υ causing the 'death' of chromosomes, specifically, service type violations and deadline violations. To calculate the fitness function of a chromosome c, we have to account for these three types of constraints.

First, we consider whether the constraints $\forall \beta_p \in \Psi$ are satisfied (i.e., if $\beta_p(c) \leq 0$) or not satisfied (i.e., if $\beta_p(c) > 0$), as in (23):

$$\delta_{\beta_p(c)} = \begin{cases} 0, & \text{if } \beta_p(c) \leq 0 \\ 1, & \text{if } \beta_p(c) > 0 \end{cases} \tag{23}$$

Similarly, $\delta_{\beta_\gamma(c)}$ denotes whether the constraints from Γ are satisfied ($\beta_\gamma(c) = 0$), or not ($\beta_\gamma(c) = 1$). Regarding the Υ constraints, the penalty distance from the satisfaction of Υ constraints for c is defined in (24), where β_υ denotes a constraint, and $\delta_{\beta_\upsilon(c)}$ indicates whether a constraint was violated in the current chromosome c, i.e., $\delta_{\beta_\upsilon(c)} = 1$.

$$D(c) = \sum_{\beta_\upsilon \in \Upsilon} \delta_{\beta_\upsilon(c)} \tag{24}$$

Provided that $\omega_{\beta_p(c)}$ is a weight factor of constraint $\beta_p \in \Psi$, $\omega_{\beta_\gamma(c)}$ is a weight factor of constraint $\beta_\gamma \in \Gamma$, and ω_p is the penalty weight factor for the Υ constraints, the fitness function is calculated according to (25):

$$F(c) = \sum_{\beta_p \in \Psi} \omega_{\beta_p}(1 - 2\delta_{\beta_p(c)}) + \\ + \sum_{\beta_\gamma \in \Gamma} \omega_{\beta_\gamma}(1 - 2\delta_{\beta_\gamma(c)}) - \omega_p D(c) \tag{25}$$

If constraints β_p or β_γ are satisfied in the considered chromosome c, then $\delta_{\beta_p(c)}$ and $\delta_{\beta_\gamma(c)}$ become 0, and the according values within the first and the second terms are added to the fitness function. When the constraints are not satisfied, $\delta_{\beta_p(c)}$ and $\delta_{\beta_\gamma(c)}$ become 1, and the according values resulting from the first and second terms are subtracted from the fitness function. The third term ensures penalty $\omega_p D(c)$ for having $D(c)$ other than 0, where the penalty factor ω_p has to be big enough to ensure that the worst chromosomes do not participate in breeding individuals of next generations in the GA.

Genetic operators As for the parameters of the GA operators, we use the 80%-uniform crossover, tournament selection, random gene mutation with 2% mutation rate, elitism rate of 20%, and a population size of 1000 individuals, which were set based on pre-experiments where these parameters were varied. The uniform crossover was chosen because genes are integer values. 80% of selected individuals perform crossovers. To combine genes from parent chromosomes, a fixed mixing ratio is used, e.g., a ratio value of 0.5 means that 50% of genes come from each parent. As for the tournament selection, each of the two chromosomes is selected based on the tournament with a certain arity. This is done by drawing a number of random chromosomes (here the arity is 2) without replacement from the population and then selecting the fittest chromosome among them. The 2% random gene mutation means random genes in chromosomes mutate with the probability of 2%.

Stopping condition The stopping condition of the GA is activated when the fitness function achieves a tolerance value of $\epsilon = 10^{-4}$, which is the average relative change in the fitness value over generations. During each run of the GA, the fitness function increases because less penalties are applied to the individuals. Therefore, the stopping condition performs only when the fitness function of the fittest individual in the generation is a positive value, i.e., when there are no 'death' penalties applied to the individual. Additionally, we include an auxiliary stopping condition with a maximum limit of generations achieved to eliminate unproductive time-consuming search.

In the next section, we evaluate our GA compared to the exact optimization method and to a baseline, namely the greedy first fit heuristics, and to the execution in the cloud.

5 Evaluation

Our evaluation aims to show the performance of various approaches of solving the FSPP. To apply the different approaches in a fog environment, we use the fog simulation toolkit *iFogSim* [17] to simulate the fog landscape and cloud resources.

5.1 Evaluation environment

While iFogSim features most of the entities necessary to model a fog landscape as described in Sect. 3, some modifications of the entities were necessary. The *Application* class is extended by the means to account for the deployment and response times of an application, and the application deadline. The *FogDevice* class is extended to perform as a fog orchestration control node. For that, the class was supplemented with the functionality to analyze the utilization of resources in the underlying fog colony and to identify the closest neighbor colony to propagate services to. To adhere to the FSPP constraints, the functionality to calculate a moving average of deployment times of applications in the closest neighbor colony was implemented (as described in Sect. 4.2).

5.2 Evaluation scenarios

We solve the FSPP by a greedy first fit heuristic (called the 'First Fit' scenario), which serves as a baseline for our evaluation, the exact optimization method (see Sect. 4.2) implemented by the means of the IBM CPLEX library[1]

[1] https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.

(called the 'Optimization' scenario), and the GA (called the 'Genetic' scenario) introduced in Sect. 4.3. If the fog landscape is not available at all, the execution is performed solely in the cloud (called the 'Cloud' scenario).

In the First Fit scenario, a service placement plan is produced by searching a *first fit* fog resource [6,30]. Hence, a list of all available fog devices in the fog landscape is sorted by the communication link delays between each resource and the fog orchestration control node, by available resource capacities, and by the types of services. The placement of services on fog resources is prioritized over the placement in the cloud. The first fit algorithm iterates over each service of each application in a cycle and checks if the current fog device in the sorted list of fog devices satisfies the service and application requirements. If the fog colony cannot host a service, the service request is propagated to the closest neighbor colony.

The Genetic scenario is based on the GA that is discussed in Sect. 4.3.

The solution of the FSPP in the Optimization scenario is computed by the exact optimization method implemented by the means of the IBM CPLEX solver. To simplify the use of this solver, the open source Java ILP library[2] is applied.

In the Cloud scenario, all the services are placed on cloud resources. This scenario aims to show the benefits of decentralization in fog landscape.

5.3 Experimental setup

As setup for the evaluation, we consider five different applications following the motivational scenario, i.e., motion, video, sound, temperature, and humidity processing applications in the manufacturing shop floor, and according sensors and actuators. The IoT applications are simulated by the means of iFogSim (iFogSim supports the DDF deployment model [17]). The needs in resources for application services are predefined to ensure that one computational device cannot host a whole application and to show that the FSPP is flexible and reacts on different input parameters. Service makespan durations were set based on received average data from pre-experiments run in iFogSim for specified services. A summary of the experimental setup is shown in Tables 2 and 3.

We observe a service placement in one fog colony that consists of ten fog cells connected to a fog orchestration control node. The communication link delays between the fog orchestration control node and the cloud, the closest neighbor colony, and fog cells are correspondingly 1, 0.5, and 0.3 s. In reality, the communication link delays depend on the physical distance between resources.

[2] https://sourceforge.net/projects/javailp/.

Table 2 Application resource demands

Service	U_{a_i} (MIPS)	M_{a_i} (MB)	S_{a_i} (MB)	m_{a_i} (s)
Sense	50	30	10	0.90
Process1	200	10	30	0.10
Process2	200	20	30	0.10
Process3	100	30	30	0.25
Actuate	50	20	10	0.50

Table 3 Application details

Application	D_{A_k} (s)	w_{A_k} (s)
A_1	120	60
A_2	300	0
A_3	300	60
A_4	360	60
A_5	240	0

In the closest neighbor colony, the expected deployment time of applications is $\overline{T}^t_{w_N} = 3$ min, and the sampled deployment time in the previous round of FSPP period is $T^{t-\tau}_{w_N} = 2$ min. Also, in the experiments, we additionally vary these parameters to show their influence on the system behavior. The CPU, RAM and storage resources in the fog orchestration control node are 1000 MIPS, 512 MB, and 8 GB accordingly. The respective resources of the fog cells are 250 MIPS, 256 MB, and 4 GB. In the experiments, CPU capacities of fog resources are also varied to show their influence on service placement. All three service types, i.e., sensing, processing, and actuating services, can be executed in the fog colony and the cloud, whereas only processing services can be propagated to the closest neighbor colony. The processing cost in the cloud is \$0.30 per billing time unit (BTU), i.e., 1 h, as in [30].

Finally, we use for the optimization model $\alpha = 0.5$ to update the expected deployment time in the closest neighbor fog colony, i.e., $\overline{T}^t_{w_N}$. In the Genetic scenario, we set the weight factor ω =1 for all constraints [37], and the penalty weight ω_p is 1000, which are explained in Sect. 4.3.

5.4 Evaluation metrics

To show how a service placement plan allows to meet the application deadlines, we observe the response times of applications. A difference between the application deadline and its response time $D_{A_k} - r_{A_k}$ shows how far the response time is from the respective deadline.

A service execution delay indicates how much time is spent by a service in the network. This metric is calculated by the means of the simulation environment depending on the communication link delays between resources.

Table 4 Scenario performance comparison

Scenario	r_{A_k} (s)					$D_{A_k} - r_{A_k}$ (s)					Placement (%)				Cost
	A_1	A_2	A_3	A_4	A_5	A_1	A_2	A_3	A_4	A_5	f_j	F	N	R	($)
Optimization	62.43	262.6	64.66	322.9	4.89	57.56	37.34	235.3	37.1	235.1	40	24	24	12	0.09
First fit	61.42	262.6	322.8	322.8	264.4	58.57	37.34	-22.87	37.11	-24.43	40	16	44	0	0.00
Cloud	60.00	0.001	60.00	60.00	0.001	59.99	299.9	239.9	299.9	239.9	0	0	0	100	4.81

Table 5 Genetic algorithm performance

Metrics	A_1	A_2	A_3	A_4	A_5	Average placement		Cost
						Resource	(% of all services)	($)
r_{A_k} (s)	65.65	262.47	65.83	322.64	5.19	f_j	28.8 ($\sigma = 3.92$)	
	($\sigma = 1.48$)	($\sigma = 0.27$)	($\sigma = 1.38$)	($\sigma = 0.25$)	($\sigma = 0.29$)	F	11.2 ($\sigma = 1.60$)	0.22
$D_{A_k} - r_{A_k}$ (s)	54.34	37.52	234.16	37.35	234.80	N	24.0 ($\sigma = 2.52$)	($\sigma = 0.02$)
	($\sigma = 1.48$)	($\sigma = 0.27$)	($\sigma = 1.38$)	($\sigma = 0.25$)	($\sigma = 0.29$)	R	36.0 ($\sigma = 4.38$)	

The utilization of the fog (cloud) is calculated as a ratio of the number of services placed on fog (cloud) resources to the total number of services. This metric demonstrates the usage of different resources.

In order to show how the model behaves in different conditions, we analyze intrinsic relationships between the service placement and different parameters (i.e., strict or loose deadlines, overloaded or underloaded resources, average deployment time in the closest neighbor colony, time between two subsequent optimization periods), which gives insights into the system behavior.

Assuming ownership of the fog landscape, the cost of service execution in the fog can be neglected. The cost of service execution is calculated for the usage of cloud resources as a product of the cost per processing in the cloud and time in seconds of using cloud resources divided by the number of seconds in 1 BTU.

5.5 Results and discussion

The aim of this evaluation is to observe the executions of service placement plans provided by various approaches, i.e., in the First Fit, Genetic, and Optimization scenarios. By applying service placement plans, we observe response times of applications, deadline violations, the utilization of resources, and the processing cost. Additionally, these results are compared with the results of the Cloud scenario. An overview of the results is shown in Table 4. A summary of the results of the Genetic scenario is shown in Table 5. These results were separated as the GA is a non-deterministic algorithm, which required running multiple repetitions of the experiments to achieve an average and deviation of the results.

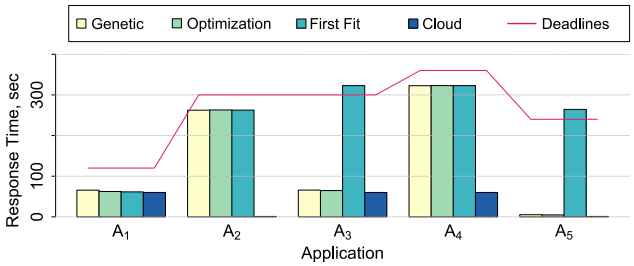

Fig. 7 Response times of applications

5.5.1 Deadline violations and service delays

The First Fit scenario results in deadline violations for the applications A_3 and A_5 by 22.87 and 24.43 s, respectively. In the Genetic, Optimization, and Cloud scenarios, the service placement solution does not violate deadlines; however, each approach leads to different results in terms of fog resource utilization and application response times (see Table 4). In the First Fit scenario, the services are propagated to the closest neighbor fog colony when the current colony is not able to execute them because of resource constraints. The cloud has the lowest priority in the desirable service placement. Therefore, in the First Fit scenario processing services are propagated to the closest neighbor colony. Because the resources in the fog landscape are less powerful than cloud resources, the spikes in Fig. 8 appear. Even though the deadlines in the Genetic and Optimization scenarios are not violated, the delays in single service executions on average are smaller in the Genetic scenario compared to the Optimization scenario.

The response times of applications and delays for single service executions in the four scenarios are depicted in Figs. 7 and 8. In Fig. 7, the Cloud scenario does not violate deadlines,

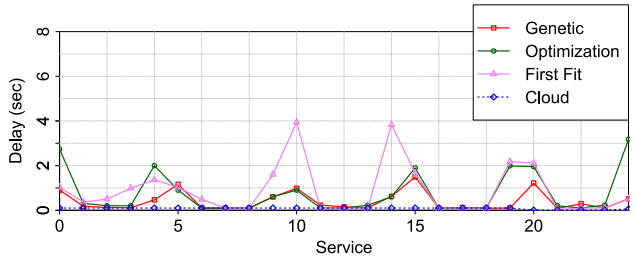

Fig. 8 Service execution delays

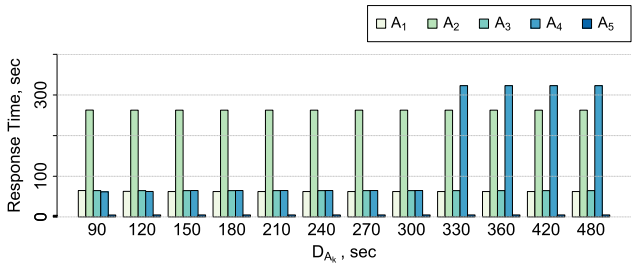

Fig. 9 Impact of application deadlines on response times

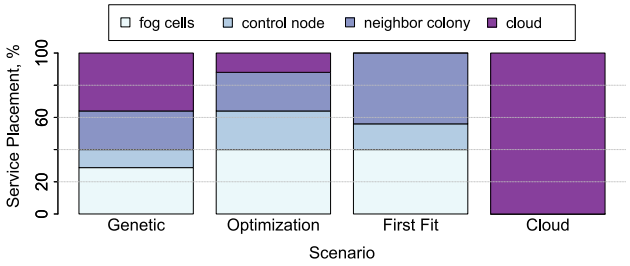

Fig. 10 Utilization of resources

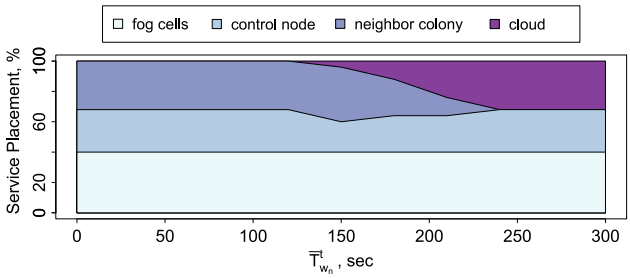

Fig. 11 Impact of $\overline{T}^{t}_{w_N}$ on service placement

and services are executed immediately after submission of application requests because of the theoretically unlimited resources of the cloud. However, in reality, the utilization of cloud resources leads to higher execution cost and higher communication delays due to the physical distance of the cloud data center.

To show the impact of application parameters on response times, we conduct another experiment which varies the deadline parameter of application A_4 (Fig. 9) and observes response times of all applications. A_4 was chosen for this experiment as it has the biggest response time (see Tables 4 and 5). Given the parameters of the applications as shown in Table 3, when the deadline is below 120 s, the processing services of A_4 are assigned only to the fog orchestration control node as this is the closest deadline among all applications. Above 120 s and below 300 s, the application A_1 has a closer deadline than A_4, and therefore processing services of A_1 and one processing service of A_4 are placed on the fog orchestration control node, and the rest two processing services of A_4 are propagated to the cloud. Above 300 s, A_4 has enough time to wait for the deployment, and therefore, some services are propagated to the closest neighbor colony, and both τ and $\overline{T}^{t}_{w_N}$ affect the response time of the application A_4.

5.5.2 *Utilization of the fog landscape*

In the First Fit scenario, all sensing and actuating services are placed on the different fog cells in the fog colony, four processing services are placed at the control node, and the remaining 11 processing services are propagated to the closest neighbor colony.

In the Genetic scenario, the sensing services of applications A_1, A_3, and A_5 are placed on the fog cells, the actuating services are placed either on the fog cells or in the cloud, and the processing services are placed in the cloud. This service placement plan includes 36% of placements in the cloud, and the available resources of the fog colony are not used optimally, i.e., the fog orchestration control node has enough free resources to host more services. Many cloud placements occur because in the fitness function of each of generated chromosomes there is a considerably big penalty for deadline violations, but there are no penalties for not using the full capacities of resources of the fog orchestration control node or fog cells.

In the Optimization scenario, all ten sensing and actuating services are placed on fog cells; however, the fog orchestration control node hosts more services, i.e., six services. Another six services are propagated to the closest neighbor colony, and three are executed in the cloud. Such service placement utilizes fog resources to a higher degree, leading to a reduced cloud utilization. The utilization of the fog landscape in different scenarios is summarized in Fig. 10.

To show how τ and $\overline{T}^{t}_{w_N}$ affect the utilization of the fog landscape, we conduct separate experiments, calculate the goal function, and observe service placement by varying τ and $\overline{T}^{t}_{w_N}$. As can be seen in Fig. 11, while $\overline{T}^{t}_{w_N}$ is small, the closest neighbor fog colony performs the processing fast and therefore allows the fog orchestration control node to assign most of the services to the closest neighbor colony. Then, $\overline{T}^{t}_{w_N}$ reaches a certain value when it starts to interfere with deadlines of applications, and, therefore, there is a decrease in the goal function as more services are propagated to the

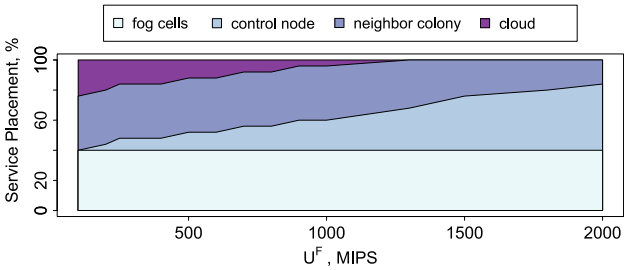

Fig. 12 Impact of resources of F on service placement

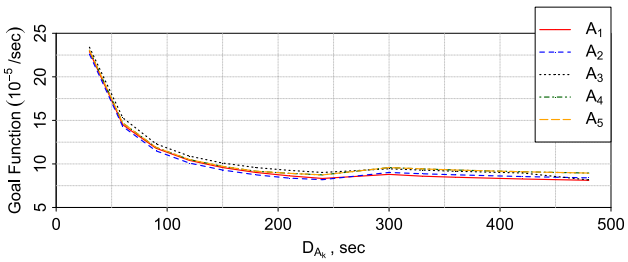

Fig. 13 Impact of application deadlines on the goal function

cloud. Also, when to increase $\overline{T}^t_{w_N}$ significantly, all the services are propagated to the cloud, and the closest neighbor colony is not sufficient any more to host any service. The same considerations are valid for the τ parameter.

Regarding variations in the fog colony's resources, the number of services placed in the fog grows as well as the goal function of the model when the capacity of the fog orchestration control node increases (see Fig. 12). In the considered scenarios, increasing the capacity of the fog cells currently does not change the goal function, because fog cells can only host sensing and actuating services, and the number of these services does not change.

The impact of variations of deadlines of applications on the goal function is shown in Fig. 13. To receive these observations, the experiment was conducted five times. In each experiment, the deadline of one application at a time has been changed from 30 s to 6 min, while all other applications remain unaltered. The results of the experiment show that when the deadline is small, the services of the considered application are placed by the fog orchestration control node as they cannot be propagated to the closest neighbor colony, because that would mean adding a further deployment time of τ plus $\overline{T}^t_{w_N}$. When the relaxed deadline is in place, the deadline becomes higher than the deadlines of other applications, i.e., starting from 3 min, which prevents the services from the considered application to be placed on the fog orchestration control node, and therefore they are propagated to the cloud. The goal function value in this case is reduced according to the coefficient $\frac{1}{D_{A_k} - r_{A_k}}$ and due to the fact that less services are placed in the fog. When the deadline becomes more relaxed comparing to the parameters τ and $\overline{T}^t_{w_N}$, i.e., starting from 5 min, services are propagated to the closest neighbor colony, and the goal function slightly increases, as more services are hosted by the fog landscape. After that, only reduction is observed due to the change of the $\frac{1}{D_{A_k} - r_{A_k}}$.

5.5.3 Cost of execution

The cost of service execution according to the Genetic scenario is \$0.22, because 9 out of 25 services are executed in the cloud. In the Optimization scenario, the execution cost constitutes 40% of the cost in the Genetic scenario, charging \$0.09 since only three services are propagated to the cloud. The execution cost received in the Optimization and Genetic scenarios is 2 and 4%, respectively, compared to the execution cost in the Cloud scenario. The results are shown in Tables 4 and 5.

To sum up the most important observations of the evaluation, the execution of the service placement plans produced in the Genetic and Optimization scenarios do not violate deadlines of applications unlike the service placement plan of the First Fit scenario. The cost of execution in the Optimization scenario constitutes only 40% of the cost from the Genetic scenario. Even though the GA solution leads to less delay if observing single service executions, the exact optimization method better utilizes the fog landscape resources.

6 Related work

As fog computing is still a recent research topic, there is a lack of concrete solutions supporting this paradigm. Nevertheless, there is some conceptual as well as fundamental work in related areas, which needs to be discussed.

First, there has been some work on fog computing architectures. In their seminal conceptual work on the topic, Bonomi et al. introduce a layered model bridging the IoT and the cloud [4]. The authors show that applications may be placed in the cloud and in the fog, spanning potentially different cloud providers. In addition, it is shown that a fog computing framework has to encompass different communication links, i.e., with the cloud, within the fog, and with IoT devices. Dastjerdi et al. [10] present a reference architecture for fog computing which follows a very similar structure if compared to the work by Bonomi et al. The reference architecture implies serving IoT requests in the local fog rather than involving the cloud. In the reference architecture, central fog services are placed in a *Software-Defined Resource Management* layer that provides a cloud-based middleware. This prevents fog colonies from acting in an autonomous way. Instead, fog cells are analyzed, orchestrated, and monitored by the cloud-based middleware. Also, fog resource provisioning and the offloading of computa-

tional tasks from the fog to the cloud are achieved through the middleware. In another discussion of basic fog features, Vaquero et al. [31] consider different concepts to realize fog architectures, including both centralized and decentralized, i.e., peer-to-peer, approaches. In particular, the authors introduce the notion of *edge clouds*, which are private fogs made up from IoT devices, resembling our notion of fog colonies.

Having discussed conceptual architectures, it is necessary to focus on service delivery models for fog computing. The first model to be discussed is stream processing. It requires the creation of a processing topology that includes IoT data sources and operators. An IoT ecosystem for stream processing called VISP is proposed in the work of Hochreiner et al. [19]. In VISP, complex network topologies are analyzed, and services are placed on resources according to QoS constraints. This work is an interesting example of a real-world testbed which can be used for implementation of a fog computing framework in the future. Giang et al. [16] introduce a DDF programming model, which is also the basic application model applied within the work at hand. This model resembles the stream processing approach and provides means to create and execute IoT applications. In the work of Barnaghi et al. [11], a structured IoT information model is proposed. This model is based on semantic annotations and is divided into (i) an entity model, which aims to establish basic physical entities and relationships in the IoT infrastructure, and (ii) a resource model, which represents software artifacts corresponding to those physical entities. The proposed IoT information model can be used, e.g., for service association discovery and monitoring, for reasoning upon semantic annotations, and for decision-making processes. A different service delivery model is Network Function Virtualization, described in a survey by Han et al. [18]. This approach influences both the conceptual architecture of the fog computing environment and the modeling of IoT applications. Virtualized Network Functions (VNF) are virtual appliances which can be placed onto physical resources. With regard to fog computing, VNFs can be considered as containers with software corresponding to fog orchestration control nodes, fog cells, and cloud-fog control middleware, which were discussed in our paper. We assume that these VNFs are already placed onto fog devices. In fact, in our work, we consider one layer above VNFs, i.e., the placement of IoT applications on top of VNFs.

The conceptual fog frameworks discussed above do not take into account the concrete needs of resource provisioning and service placement in the fog. Instead, the focus is on communication and task sharing between the different layers, i.e., cloud, fog, and IoT. In fact, the number of resource provisioning mechanisms specifically aiming at fog computing is quite limited so far. Hong et al. present a programming model including a simple resource provisioning strategy which relies on workload thresholds, i.e., if the utilization of a particular fog cell exceeds a predefined value, another fog cell is leased [20]. Aazam and Huh present a more sophisticated resource provisioning mechanism based on the prediction of resource demands [1]. In this work, dynamic allocation of resources is performed in advance during the design time of the system. This approach is based on cost optimization, and the resource allocation depends on the probability fluctuations of the demand of the users, types of services, and pricing models. In another work of Vögler et al. [32], a policy-based approach to optimize deployment topologies on the edge devices is presented. This approach offers an elastic application deployment by the means of defining a 'hot pool' of resources per each service of requested applications to enable additional scaling. In our work, we also consider runtime service placement optimization to account for dynamic infrastructural changes in a fog landscape; however, we concentrate on more concrete formalization of a system model and optimization problem.

Apart from fog-specific resource provisioning solutions, resource provisioning and service placement are major research challenges in the general field of cloud computing [7,23,40]. While these approaches offer interesting insights, there are certain differences between fog services and cloud services. These differences prevent a direct adaptation of cloud resource provisioning solutions. First, the size and type of cloud resources are very different from their counterparts in fog computing. While cloud resources are usually handled on the level of physical machines, virtual machines, or containers, fog resources are usually not as powerful and extensive. Second, fog colonies may be distributed in a rather large area and heterogeneous network topology, while cloud resources are usually placed in centralized data centers, making it more important to take into account data transfer times when developing solutions for resource allocation in the fog. This is especially important since one particular reason to use fog computing in IoT scenarios is the higher delay-sensitivity of fog-based computation [4]. Hence, resource provisioning approaches for the fog need to make sure that this benefit is not foiled by extensive data transfer times and cost.

Resource provisioning is also an important topic in Mobile Cloud Computing (MCC) [14], which integrates mobile devices and cloud resources and offers solutions for offloading tasks from mobile devices to the cloud [12]. However, MCC is mostly based on a rather simple network topology with direct communication between mobile devices and the cloud. Neither groups of devices (as in fog colonies), nor the different layers observed in fog computing are taken into account in MCC. Therefore, again, the according resource provisioning approaches offer interesting insights and ideas, but cannot be directly ported to the field of fog computing.

Our former works [29,30] consider both the structure of a fog landscape and service placement mechanisms. The

paper at hand extends those contributions offering additional service placement mechanisms (i.e., the genetic algorithm-based heuristic) and evaluation experiments.

7 Conclusion

Fog computing aims to utilize available computational, storage, and networking resources for the enactment of IoT applications close to the edge of the network. Currently, the uptake of fog computing is still at its very beginning, and thus there is a lack of theoretical and practical foundations for fog resource provisioning and service placement.

After having motivated our work with a scenario from the field of Cloud Manufacturing, we discussed a conceptual architecture for a fog computing framework and formalized an optimization problem for service placement in the fog, called the FSPP. We simulated the envisioned architecture and solved the FSPP using several approaches, i.e., a greedy first fit heuristic, a genetic algorithm, and an exact optimization method. Also, we compared all the results with the execution of the same experimental setup in the cloud. Unlike the service placement plans produced by the greedy first fit heuristic, the service placement plans of the genetic algorithm and the exact optimization method do not violate deadlines of applications. The optimization method produces a service placement plan which is more effective in utilizing the fog landscape resources, leading to lower execution cost when compared to the average service placement plan produced by the genetic algorithm (with the cost constituting only 40% of the cost of service placement plans produced by the genetic algorithm). The genetic algorithm produces solutions which on average experience a lower deployment delay by exploiting more cloud resources (on average 36% of the services have been run in the cloud).

In our future work, we aim to implement a real-world testbed based on the proposed conceptual fog computing framework and to improve the system model for resource provisioning in terms of cost of resources and reliability and availability of services. The architecture can be enhanced by fault tolerance mechanisms to account for mobility in the fog landscape. Parallel heuristic algorithms should be investigated in order to find a viable substitution for the exact optimization method, which may fail to solve the problem on the Big Data scale. Another aspect of our future work is the systematic observation of a fog landscape to obtain real-world network data to evaluate the behavior of the service placement approaches.

Acknowledgements Open access funding provided by TU Wien (TUW). This paper is supported by TU Wien research funds. This work is partially supported by the Commission of the European Union within the CREMA H2020-RIA project (Grant Agreement No. 637066).

The authors thank IBM Academic Initiative for providing an academic license for the IBM CPLEX optimizer.

References

1. Aazam M, Huh EN (2015) Dynamic resource provisioning through fog micro datacenter. In: 12th IEEE international workshop on managing ubiquitous communications and services. St. Louis, Missouri, USA, pp 105–110
2. Armbrust M, Fox A, Griffith R, Joseph AD, Katz R, Konwinski A, Lee G, Patterson D, Rabkin A, Stoica I, Zaharia M (2010) A view of cloud computing. Commun ACM 53:50–58
3. Atzori L, Iera A, Morabito G (2010) The internet of things: a survey. Comput Netw 54:2787–2805
4. Bonomi F, Milito R, Natarajan P, Zhu J (2014) Fog computing: a platform for internet of things and analytics. In: Big data and internet of things: a roadmap for smart environments. Studies in computational intelligence 546: 169–186. Springer
5. Botta A, de Donato W, Persico V, Pescape A (2016) Integration of cloud computing and internet of things: a survey. Fut Gener Comput Syst 56:684–700
6. Brent RP (1989) Efficient implementation of the first-fit strategy for dynamic storage allocation. ACM Trans Prog Lang Syst 11(3):388–403
7. Cardellini V, Grassi V, Lo Presti F, Nardelli M (2016) Optimal operator placement for distributed stream processing applications. In: 10th ACM international conference on distributed and event-based Systems (DEBS'16), pp 69–80. ACM, Irvine, California, USA
8. Chen M, Mao S, Liu Y (2014) Big data: a survey. Mobile Netw Appl 19(2):171–209
9. Dabek F, Cox R, Kaashoek F, Morris R (2004) Vivaldi: a decentralized network coordinate system. ACM SIGCOMM Comput Commun Rev 13(4):15–26
10. Dastjerdi AV, Gupta H, Calheiros RN, Ghosh SK, Buyya R (2016) Fog computing: principles, architectures, and applications. In: Internet of things: principles and paradigms, chap. 4, Morgan Kaufmann
11. De S, Barnaghi P, Bauer M, Meissner S (2011) Service modelling for the internet of things. In: Federated conference on computer science and information systems, 2011. IEEE, Szczecin, Poland, pp 949–955
12. Dinh HT, Lee C, Niyato D, Wang P (2013) A survey of mobile cloud computing: architecture, applications, and approaches. Wirel Commun Mob Comput 13:1587–1611
13. Elmroth E, Leitner P, Schulte S, Venugopal S (2017) Connecting fog and cloud computing. IEEE Cloud Comput 4(2):22–25
14. Fernando N, Loke SW, Rahayu W (2013) Mobile cloud computing: a survey. Fut Gener Comput Syst 29:84–106
15. Georgakopoulos D, Jayaraman PP, Fazia M, Villari M, Ranjan R (2016) Internet of things and edge cloud computing roadmap for manufacturing. IEEE Cloud Comput 3(4):66–73
16. Giang NK, Blackstock M, Lea R, Leung VC (2015) Developing IoT Applications in the fog: a distributed dataflow approach. In: 5th International conference on the internet of things (IoT). IEEE, Seoul, Korea. pp 155–162

17. Gupta H, Dastjerdi AV, Ghosh SK, Buyya R (2017) iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Softw Pract Exp 47:1275–1296
18. Han B, Gopalakrishnan V, Ji L, Lee S (2015) Network function virtualization: challenges and opportunities for innovations. IEEE Commun Mag 53(2):90–97
19. Hochreiner C, Vögler M, Waibel P, Dustdar S (2016) VISP: an ecosystem for elastic data stream processing for the internet of things. In: IEEE 20th International enterprise distributed object computing conference (EDOC), 2016. IEEE, pp 1–11
20. Hong K, Lillethun D, Ramachandran U, Ottenwlder B, Koldehofe B (2013) Mobile fog: a programming model for largescale applications on the internet of things. In: 1st ACM SIGCOMM workshop on mobile cloud computing. Hong Kong, China, pp 15–20
21. Huang X, Ganapathy S, Wolf T (2009) Evaluating algorithms for composable service placement in computer networks. In: IEEE International conference on communications. IEEE, Dresden, Germany. pp 2276–2281
22. Kubler S, Holmström J, Främling K, Turkama P (2016) Technological theory of cloud manufacturing. Stud Comput Intell 640:267–276 Springer
23. Leitner P, Hummer W, Satzger B, Inzinger C, Dustdar S (2012) Cost-efficient and application SLA-aware client side request scheduling in an infrastructure-as-a-service cloud. In: 5th International Conference on Cloud Computing. IEEE, Honolulu, HI, USA, pp 213–220
24. Papageorgiou A, Cheng B, Kovacs E (2015) Real-time data reduction at the network edge of internet-of-things systems. In: 11th International Conference on Network and Service Management. Barcelona, Spain, pp 284–291
25. Qu T, Lei SP, Wang ZZ, Nie DX, Chen X, Huang GQ (2016) IoT-based real-time production logistics synchronization system under smart cloud manufacturing. Int J Adv Manuf Technol 84(1):147–164
26. Rohjans S, Dnekas C, Uslar M (2012) Requirements for Smart Grid ICT-architectures. In: 3rd IEEE PES international conference and exhibition on innovative smart grid technologies. Berlin, Germany, pp 1–8
27. Schulte S, Hoenisch P, Hochreiner C, Dustdar S, Klusch M, Schuller D (2014) Towards process support for cloud manufacturing. In: 18th IEEE International enterprise distributed object computing conference. IEEE, Ulm, Germany, pp 142–149
28. Skarlat O, Borkowski M, Schulte S (2016) Towards a methodology and instrumentation toolset for cloud manufacturing. In: 1st International workshop on cyber-physical production systems, CPS week 2016. IEEE, Vienna, Austria, pp 1–4
29. Skarlat O, Nardelli M, Schulte S, Dustdar S (2017) Towards QoS-aware fog service placement. In: 1st IEEE international conference on fog and edge computing (ICFEC), 2017. IEEE, Madrid, Spain, pp 89–96
30. Skarlat O, Schulte S, Borkowski M, Leitner P (2016) Resource provisioning for IoT services in the fog. In: 9th IEEE international conference on service oriented computing and applications (SOCA), 2016. IEEE, Hong Kong, China, pp 32–39
31. Vaquero LM, Rodero-Merino L (2014) Finding your way in the fog: towards a comprehensive definition of fog computing. ACM SIGCOMM Comput Commun Rev 44(5):27–32
32. Vögler M, Schleicher J, Inzinger C, Dustdar S (2016) Optimizing elastic IoT application deployments. Trans Serv Comput PP:1–14
33. Whitley D (1994) A genetic algorithm tutorial. Stat Comput 4:65–85
34. Wu D, Greer MJ, Rose DW, Schaefer D (2013) Cloud manufacturing: strategic vision and state-of-the-art. J Manuf Syst 32:564–579
35. Xu M, Tian W, Buyya R (2017) A survey on load balancing algorithms for virtual machines placement in cloud computing. Concurr Comput e4123:1–16
36. Xu X (2012) From cloud computing to cloud manufacturing. Robot Comput Integr Manuf 28(1):75–86
37. Ye Z, Zhou X, Bouguettaya A (2011) Genetic algorithm based QoS-aware service compositions in cloud computing. In: 16th International conference on database systems for advanced applications (DASFAA) part II. Springer, Berlin, Heidelberg, Hong Kong, China, pp 321–334
38. Yoo M (2009) Real-time task scheduling by multiobjective genetic algorithm. J Syst Softw 82(4):619–628
39. Yu J, Buyya R, Ramamohanarao K (2009) Workflow scheduling algorithms for grid computing. Stud Comput Intell 146:173–214
40. Zhan ZH, Liu XF, Gong YJ, Zhang J, Chung HSH, Li Y (2015) Cloud computing resource scheduling and a survey of its evolutionary approaches. ACM Comput Surv 47(4):63