TU WIEN Informatics

Resource Provisioning in Fog Computing

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Olena Skarlat, MSc
Matrikelnummer 01429034

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Prof. Dr.-Ing. Stefan Schulte

Diese Dissertation haben begutachtet:

Josef Spillner	David Bermbach

Wien, 18. April 2023

Olena Skarlat

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Resource Provisioning in Fog Computing

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Olena Skarlat, MSc
Registration Number 01429034

to the Faculty of Informatics

at the TU Wien

Advisor: Prof. Dr.-Ing. Stefan Schulte

The dissertation has been reviewed by:

Josef Spillner	David Bermbach

Vienna, 18th April, 2023

Olena Skarlat

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Olena Skarlat, MSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 18. April 2023

Olena Skarlat

Acknowledgements

First and foremost, my sincere gratitude goes to my scientific advisor Prof. Dr.-Ing. Stefan Schulte, whose support, mentorship, and guidance have been instrumental in the completion of this doctoral thesis. I am deeply thankful to Dr.-Ing. habil. Josef Spillner of the ZHAW School of Engineering, as well as to Prof. Dr.-Ing. David Bermbach of TU Berin for reviewing this thesis and providing insightful comments. I extend my gratitude to Univ.-Prof. Uwe Zdun and to Univ.-Prof. Wolfgang Kastner for assessing my work and providing their feedback during my proficiency evaluation.

I would like to thank for the opportunity to pursue my PhD studies at the Doctoral College "Cyber-Physical Production Systems" at TU Wien. I am also very proud to be a member of the Distributed Systems Group at TU Wien. I would like to express my gratitude to Univ.-Prof. Schahram Dustdar, to the entire administration team, and to my colleagues from the DSG for their support.

I would like to express my sincerest words of appreciation to teachers and lecturers of the Lyceum "Intellect" in Kyiv, Ukraine. I am immensely thankful to Docent Dr. Oleksandr Myronets for his guiding light in my journey towards pursuing STEM education.

I feel especially fortunate that my family has always been there for me despite many years of distance between us. My dear sister Olga has been my rock and my source of motivation during the highs and lows of this journey.

This dissertation is supported by the Commission of the European Union within the CREMA H2020-RIA project (Grant agreement no. 637066), by the TU Wien University Library, and by TU Wien research funds.

Kurzfassung

Das Internet of Things (IoT) führt zu einer ständig wachsenden Anzahl an vernetzten IoT-Geräten. Diese Geräte fungieren nicht nur als Sensoren oder Aktuatoren, sondern verfügen über Rechen-, Speicher- und Netzwerkressourcen, die oft nicht weiter in Betracht gezogen werden. Da sich diese Geräte am Rande des Netzwerks befinden, können sie genutzt werden, um IoT-Applikationen verteilt auszuführen. Die Kombination von IoT-Geräten am Rande des Netzwerks und von Ressourcen in der Cloud ermöglichen neuartige IoT-Szenarien und Anwendungsfälle wie beispielsweise Smart Cities. Dieses neue Paradigma, das die nahtlose Verwendung von Infrastruktur und Rechenkapazitäten am Rande des Netzwerks bis hin zur Cloud ermöglicht, wird als *Fog Computing* bezeichnet. Während die theoretischen Grundlagen des Fog Computings bereits gelegt wurden, fehlt es an konkreten Ansätzen für die effiziente Nutzung von Fog-basierten Rechenressourcen. Eine besondere Herausforderung bei der Nutzung dieser Ressourcen in der Fog ist die geographische Verteilung der Datenquellen und die Zuverlässigkeit sowie die Fehlertoleranz von IoT-Geräten. Weitere Problemstellungen ergeben sich durch die Verzögerungsempfindlichkeit von IoT-Applikationen.

Um eine effiziente Nutzung Fog-basierter Rechenressourcen zu ermöglichen, führt diese Arbeit das Prinzip sogenannter Fog-Kolonien ein. Diese Mini-Rechenzentren bestehen aus Geräten am Rande des Netzwerks und Ressourcen in der Cloud. Eine Vielzahl von Fog-Kolonien bilden eine Fog-Landschaft, in der die Ausführung von IoT-Applikationen ermöglicht wird. Dafür stellen wir eine konzeptionelle Architektur und neuartige Ansätze für die Ressourcenbereitstellung und Dienstplatzierung in der Fog vor. Wir modellieren Ressourcen und Applikationen in der Fog und formulieren und lösen das Fog Service Placement Problem, um IoT-Applikationen effizient in der Fog ausführen zu können.

Diese Dissertation befasst sich mit den grundlegenden Problemen, die mit der Einführung von Fog Computing einhergehen. Die entwickelte konzeptionelle Architektur für Fog Computing bildet die Grundlage für das Fog Computing Framework *FogFrame* – ein System, das in der Lage ist, IoT- und Cloud-Ressourcen in einer Fog-Landschaft zu verwalten und zu überwachen und IoT-Applikationen auszuführen. FogFrame ermöglicht die Kommunikation und Interaktion von Geräten innerhalb der Fog sowie die dezentrale Platzierung und Ausführung

von Applikationen. Wir analysieren die Leistung verschiedener Ansätze zur Ressourcenbereitstellung in FogFrame. Weiters demonstrieren wir die Fähigkeit von FogFrame, Anpassungen an volatile Änderungen in der Fog durchzuführen inklusive Maßnahmen zum Ausgleich von Lastspitzen und zur Wiederherstellung von Applikationen nach dem Ausfall von Ressourcen.

Abstract

The Internet of Things (IoT) leads to an ever-growing presence of ubiquitous networked IoT devices. The computational resources of these devices, which can be used not only for collecting data but also for data processing, are often neglected. Being located at the network edge, these resources can be exploited to execute IoT applications in a distributed manner. The combination of the IoT devices at the edge and resources in the cloud opens doors to novel IoT scenarios and use cases, for example, smart cities. This new paradigm, which seamlessly considers technology, infrastructure, and computations from the edge to the cloud, is known as *fog computing*. While the theoretical foundations of fog computing have been already established, there is a lack of concrete approaches to enable the efficient exploitation of fog-based computational resources. Particular challenges of adopting these resources in the fog are the adherence to the geography of data sources, the reliability and fault tolerance of computational resources of IoT devices, and the delay-sensitivity of IoT applications.

In order to enable the efficient exploitation of fog-based computational resources, this thesis introduces fog colonies, which resemble mini data centers consisting of devices at the edge of the network and in the cloud. A multitude of fog colonies forms a fog landscape, where we aim to enable IoT application execution. For that, we introduce a conceptual architecture and novel approaches for resource provisioning and service placement in the fog. We model resources and applications in the fog and formulate and solve the Fog Service Placement Problem to efficiently distribute IoT applications over fog resources.

This thesis addresses the fundamental and critical issues that come along with the adoption of fog computing. The conceptual architecture for fog computing becomes the foundation for the fog computing framework FogFrame – a system able to manage and monitor edge and cloud resources in a fog landscape, and to execute IoT applications. We enable communication and interaction within the fog as well as provide application management, namely decentralized service placement, deployment, and execution. We show, through experiments in FogFrame, the performance of different resource provisioning approaches in a real-world fog as well as adaptations to volatile changes in the fog, balancing the workload, and recovering from failures.

Contents

List of Figures

List of Tables

Core Publications

This dissertation is originated from work published in different scientific conferences and journals. The following papers represent the core of this dissertation. Parts of these papers are included in different chapters in this dissertation in verbatim. For a full list of publications, please refer to Appendix A.

- Olena Skarlat and Stefan Schulte. FogFrame: A framework for IoT application execution in the fog. *PeerJ Computer Science*, 7:e588, 1–45, 2021. eCollection. DOI: 10.7717/peerj-cs.588.
- Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. A Framework for Optimization, Service Placement, and Runtime Operation in the Fog. In *11th International Conference on Utility and Cloud Computing (UCC 2018)*, pages 164–173, Zurich, Switzerland, 2018. IEEE. DOI: 10.1109/UCC.2018.00025.
- Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards QoS-aware Fog Service Placement. In *1st International Conference on Fog and Edge Computing (ICFEC 2017)*, pages 89–96, Madrid, Spain, 2017. IEEE. DOI: 10.1109/ICFEC.2017.12
- Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized IoT Service Placement in the Fog. *Service-Oriented Computing and Applications*, 11(4), 1–17, 2017. DOI: 10.1007/s11761-017-0219-8
- Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Resource Provisioning for IoT Services in the Fog. In *9th International Conference on Service Oriented Computing and Applications (SOCA 2016)*, pages 32–39, Hong Kong, China, 2016. IEEE. DOI: 10.1109/SOCA.2016.10

CHAPTER 1

Introduction

Cloud technologies have become a major trend in the world economy. Companies and end users nowadays have the means to lease computational assets in an on-demand, utility-like fashion [32]. Cloud computing remains means to an end to optimize and move towards achieving business continuity: the pandemic forced even more aggressive adoption of cloud computing yielding awareness towards technologies and acceleration of the public cloud infrastructure market [9]. As a second major technology trend, the arrival of the Internet of Things (IoT) leads to the pervasion of business and private spaces with ubiquitous computing devices, which are available in many forms, able to act autonomously, and often enriched with Internet connectivity [12]. These devices sense their environments, perform computations, and enact operations by working autonomously or by cooperating with other devices. Furthermore, IoT devices can expose their computing and storage capabilities.

The proliferation of the cloud and IoT technologies enables small- and large-scale smart environments and systems for various domains [146], such as smart cities [63], healthcare [37], grids [97] and manufacturing [41]. IoT data is mostly produced in a distributed way, sent to a centralized cloud for processing, and then delivered to distributed stakeholders or other distributed IoT devices, often located close to the initial data sources. This centralized processing approach is a common business model in industry, however the number of IoT devices, the great volume and the speed of produced data together result in high communication latency and low data transfer rates between IoT devices as well as the IoT devices and potential users [25, 99]. Hence from a technological point of view, the decentralized nature of the IoT does not match the rather centralized structure of the cloud. One more issue regarding the centralized processing is that computational resources of IoT devices, which can be used not only for collecting data but also for data processing, are often neglected. Typical examples of such IoT devices which

possess computational resources and are capable to host IoT applications are gateways, routers, or sensor nodes [45, 60, 141]. Enterprises are eager to employ IoT devices at the network edge to bridge the gap between end users and the cloud, providing low-latency performance independently from location, scaling on-demand, and being consumed 'as-a-service' [95, 122].

In order to implement delay-sensitive smart environments, 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 [25, 111]. For this, it is necessary to move computational and storage resources needed to process IoT data closer to the data sources and service consumers, i.e., end users or data sinks [45]. The underlying conceptual approach, i.e., the virtualization of IoT devices at the edge of the network and the subsequent usage of those virtualized resources in combination with cloud resources in order to execute IoT applications, is known as fog computing [25]. In other words, fog computing contributes a missing link in the cloud-to-thing continuum [42], where the IoT, edge and cloud, which consist of a multitude of heterogeneous networked devices, cooperate with each other in the form of a fog landscape [12, 45].

To implement fog computing, it is necessary to introduce a framework to manage and monitor the fog, consisting of different computational resources offered in the cloud and IoT devices at the edge of the network [42], with the purpose to execute distributed IoT applications efficiently. This fog computing framework has to provide also means for efficient exploitation of such a heterogeneous and volatile resource pool while satisfying defined Quality of Service (QoS) parameters and Service Level Agreements (SLAs). This cooperative resource pool becomes a fog landscape, and in order to organize it and enable cooperation within different levels of resources in the fog, we introduce the notion of a fog colony within a fog landscape. Fog colonies cover certain geographical locations, can be dedicated to certain domains, act as so-called mini data centers for cooperative execution of IoT applications, and are able to communicate with other fog colonies present in a fog landscape.

The fog computing framework has to achieve an efficient exploitation of a fog landscape. For that, mechanisms for resource provisioning need to be enabled. The state-of-the-art research on resource provisioning in fog computing originates from methods and algorithms in cloud computing. However, resource provisioning in the fog and in the cloud differ in terms of execution environment, optimization goals, infrastructure and application constraints [19, 76, 93, 123]. In the cloud, optimization is done on the cloud infrastructure level including physical computational machines, Virtual Machines (VMs) on physical machines, or containers on those VMs. In the cloud-to-thing continuum, in which the fog operates, the optimization of resources can be done on multiple levels involving different resources of the fog, i.e., at the edge of the network and in the cloud, and therefore there is a need of a more lightweight technology other than a VM,

in order to be able to accommodate capacities and capabilities of devices.

1.1 Problem Statement

Smart Environments

Fog computing has been named as an enabler to provide IoT applications in many different scenarios, especially with regard to smart systems, for example, smart cities, smart buildings, or smart factories [70, 80, 135]. By deploying IoT applications in the fog, it is possible to filter data for stream processing or to conduct IoT data processing on-site instead of relying on cloud-based computational resources [25, 72]. This leads to lower latency in IoT scenarios [116].

Research Focus

Fog computing has been a vivid field of research in recent years, and the theoretical principles of fog computing, as well as conceptual fog architectures, are already well-established [19, 76, 116, 123]. However, there is a lack of frameworks that implement fog computing in practice and answer the questions of how to setup a real-world fog landscape, how to manage and monitor the fog infrastructure, how to provision resources and accordingly deploy and execute IoT applications, and how to dynamically react to changes in computational demand and in a fog landscape itself. Therefore, in this dissertation, the research is focused on conceptualizing, abstracting resources and functionalities of the fog as well as on the practical realization of the fog that is able to execute real-world IoT applications.

In the following, we frame these discussed challenges around specific research questions.

In order to realize fog computing, we need to establish a coordinated control over the physical and virtual IoT infrastructure both at the edge of the network and with additional resources from the cloud. To achieve this, available resources of IoT devices and the cloud have to be integrated into a fog landscape:

- Resources of IoT devices at the edge of the network need to be virtualized, abstracted, and represented in the fog.
- Cloud resources need to complement the resources at the edge.

These mechanisms become essential requirements to implement fog computing.

Virtualization

Notably, as has already been mentioned, fog computing is based on common principles from the field of cloud computing. One of the main principles to consider is virtualization [39, 161]: In the cloud, a physical machine is provided in terms of a VM and containers. Fog computing employs the idea that computational resources from IoT devices can be offered in a similar manner. Since cloud VMs are resource-intensive, they are not the best virtualization approach for rather resource-constrained IoT devices [142]. IoT devices at the edge of the network are less powerful than cloud resources. Therefore, a lightweight virtualization technology is needed to abstract their functionalities and represent them as small

instances which contain instructions and execute IoT applications. At the edge, abstractions should provide different levels of functionalitites based on capabilities and capacities of IoT devices, i.e., they need to execute IoT applications as well as to provide functionalities to manage and support the underlying networked infrastructure, thus enabling fog computing. A promising solution for this issue is the utilization of containers as a virtualization mechanism for resources at the edge [25, 103, 144].

These challenges constitute our first hypothesis and Reserach Question I:

Hypothesis A fog landscape includes heterogeneous devices at the edge of the network and in the cloud. These resources are virtualized, abstracted and represented in the fog.

Research Question I

How can heterogeneous IoT devices at the edge and resources in the cloud be utilized together to enable fog computing?

Fog Computing Environment

After virtualizing IoT devices, the next step is to enable a coordinated control over the physical and virtual infrastructure in the fog. Devices at the edge of the network are dynamic, i.e., they can enter and leave a fog landscape, they can experience outages in communication, as well as failures and overloads. The challenge here is to create and maintain a volatile fog landscape enabling devices and data sources potentially entering or leaving the fog at any time, and considering that the data volume to be processed within the fog can frequently change [124, 143]. Taking into account the potentially volatile and fluctuating nature of the fog, not only it is needed to create and maintain a fog landscape consisting of heterogeneous resources at the edge of the network and in the cloud, establishing communication and interaction within the fog, but also to efficiently provision their resources for IoT application execution: distribute, deploy, and execute IoT applications onto those resources.

Resource Provisioning Goals

Resource provisioning in fog computing needs to be able to allocate resources in a fog landscape taking into account the volatile fog, to provide application runtime management, and to guarantee efficient and fault-tolerant performance. The provisioning goals in fog computing include: optimization of resource utilization in the fog, optimization by latency or by location, maximization of the usage of certain resources, and minimization of the usage of other resources, e.g., of the cloud, and minimization of the operational overhead, and saving energy [4]. Resource provisioning in fog computing has to take into account a multitude of constraints, which are regarded in cloud computing, such as adhering to time-sensitivity, QoS and SLAs, as well as constraints which are not regarded in cloud computing. Constraints can be derived from capabilities and specification of IoT

devices used at the edge of the network. Constraints can be based on types of devices, device availability, power consumption, data transfer times, adherence to certain covered locations, bandwidth capabilities, connection and networking [51]. A fog landscape has to adapt to volatile events in the network and to optimize resource provisioning, migrate and deploy applications depending on the changes in the infrastructure. All these requirements need to be implemented within resource provisioning methods to enable distributed execution of IoT applications.

These challenges become the basis for our second hypothesis and Reserach Question II:

Hypothesis Different algorithms, either exact mathematical optimization, or heuristics and meta-heuristics, which account for both a heterogeneous landscape and workload of applications, can be formalized and implemented in the context and constraints of fog computing.

Research Question II

How can the execution of IoT applications in a fog landscape be optimized for resource efficiency and adherence to QoS and SLAs?

Fog Computing Framework

In order to bring together under one umbrella of fog computing our heterogeneous and volatile resources in a fog landscape together with the efficient execution of IoT applications, we need to introduce a fog computing framework. The architecture of this framework has to provide technological means and advances in the fields of IoT resource virtualization, management of heterogeneous resources, and resource provisioning and service placement in the fog.

Within this framework, it is necessary to introduce mechanisms that provide virtualization and abstraction of fog resources, establish communication between them, introduce practical means to store and transfer data within the fog, and, last but not least, enable resource provisioning and service placement for efficient execution of IoT applications. A fog computing framework has to adapt to volatile changes in the fog and optimize accordingly its resource provisioning: distribute and deploy IoT applications on different resources, migrate and redeploy them depending on the changes in the fog.

From these challenges originate our third hypothesis and Research Question III:

Hypothesis A fog landscape can effectively operate at runtime considering the volatile and fluctuating nature of its resources and efficiently execute IoT applications.

Research Question III

How to achieve a highly available, consistent, and fault-tolerant real-world fog landscape?

1.2 Scientific Contributions

Based on the research questions discussed in Section 1.1, the following scientific contributions are achieved in this thesis:

Contribution I

Fog landscape: a conceptual architecture, artifacts and functionalities that enable resource-efficient fog computing.

In this dissertation, the decentralized processing in different environments is investigated, i.e., how to create a fog landscape that comprises of IoT resources at the edge of the network and of cloud resources. As a result, a conceptual architecture for fog computing is created. The main goal of this architecture is to abstract and enable resources of fog computing as well as to provide functionalities to efficiently exploit ubiquitous networked computing IoT devices that are close to the edge of the network and complement them with the resources in the cloud with the purpose to execute arbitrary IoT applications.

In order to efficiently exploit the fog, within this contribution, we enable resource provisioning in a fog landscape. Fog resources are heterogeneous, which means certain resources have more capabilities and computational capacities than other resources. Those more powerful resources need to manage and coordinate the less powerful resources in order to establish and maintain the fog. One of their main functionalities is the ability to perform allocation of other available resources in the fog according to needed demands in workload.

Contribution II

QoS-aware resource provisioning and service placement in the fog.

Resource provisioning and service placement in a fog landscape has to be efficient and account for different optimization goals and constraints distinctive and original to fog computing. Within this contribution, we introduce the resource model of a fog landscape and an IoT application model. Based on these two models we formulate the Fog Service Placement Problem (FSPP) and design and

develop several approaches to solve it, i.e., a greedy first-fit algorithm, a Genetic Algorithm (GA) algorithm, and exact optimization. Within the evaluation of the FSPP we observe how the workload of submitted for execution IoT applications is distributed between different resources in the fog.

Our work on resource provisioning approaches starts with simulations of different arbitrary fog landscapes in order to test the performance of those approaches. The implementation of those fog landscape simulations within different environments become an enablement layer for experiments, and thus are also part of this contribution. Working with simulated environments allows implementing algorithms faster and more dynamically perform experiments with the purpose to realize those approaches later when a real-world fog landscape is implemented.

Contribution III

IoT application execution with the fog computing framework FogFrame.

Taking into account the potentially volatile nature of a fog landscape, the fog computing framework FogFrame:

- Creates and maintains a fog landscape made up from computational resources at the network edge and in the cloud,
- Establishes communication and interaction in a fog landscape,
- Efficiently deploys and executes IoT applications by distributing the single services of IoT applications on the available fog resources.

The framework is built to provide coordinated control over the physical and virtual infrastructure of a fog landscape. Resource provisioning and service placement are the main functionalities in this framework. The framework is intended to be used by research communities to provide a testbed for experimentation with resource provisioning. The framework enables volatile IoT devices at the network edge to interact within the fog, taking into account devices that potentially enter or leave a fog landscape at any time. Within this contribution, we introduce different runtime events in the fog and the corresponding means to react to them within FogFrame enabling reentrance of services, focusing on the volatility in a fog landscape. We introduce different reactive and proactive methods to tackle the events of entering devices into a fog landscape, IoT device failures and losses of communication. When new devices appear in the fog, we implement proactive migration of parts of applications on the new devices from overloaded devices. Other events like failures and overloads trigger reactive response by the framework that aims to reenable all affected applications on other available fog resources, thus enabling reentrancy.

The implementation of FogFrame starts with the technical configuration of physical and virtual resources and establishing communication between those

resources. After a fog landscape is configured, IoT applications can be submitted for execution. In order to efficiently utilize available resources of a fog landscape and achieve optimal performance, in FogFrame several control mechanisms are developed to:

- Analyze resource utilization within the fog,
- Perform resource provisioning according to specified goals and analyzed constraints,
- Allocate resources for IoT applications,
- Perform infrastructural changes and monitoring in the fog landscape, thus tackling the problem of fog volatility.

The identified abstractions of resources and functionalities in a fog landscape provide the necessary and sufficient level of democratization for fog computing, i.e., providing usability, modularization, and interoperability within the fog.

1.3 Methodology

This dissertation provides conceptual foundations and requirements towards creating a fog landscape that efficiently executes IoT applications. A more specific focus is on the formalization of resource provisioning and service placement approaches, their implementation and evaluation within the fog.

Research Framework

This thesis follows the design science research framework [150] in implementing artifacts and advancing knowledge to answer the research questions presented in Section 1.1. The design science research framework describes the means to solve problems in Information Systems research, and provides a holistic understanding on why the implemented artifacts and acquired knowledge improve and contribute to the chosen problem. Examples of such artifacts are theories or systems, their components, approaches, algorithms or models [113]. In this dissertation, we investigate means-ends relationships to realize resource provisioning in fog computing. The overall research effort in this work follows general principles of the rationalist and technocratic paradigms of Computer Science [52]. These principles comprise from a wide spectrum of experimental and manipulative methods to evaluate research activities: the identification of functional and technical requirements of artifacts, conceptual architectures, the implementation, evaluation, and analysis of software artifacts following a prototyping approach.

Problem Space

First, we define a problem space of efficient resource provisioning in the fog, its needed artifacts, tasks, involved technologies and existing methods [98, 113]. After that, in order to identify the challenges, we aggregate existing knowledge in fog computing, cloud computing, distributed architectures, resource provisioning and service placement, and correlate this state of the art to the existing problems in the adjacent areas of the IoT [21, 110].

Research Objectives

Next, we formulate the research questions together with qualitative and quantitative objectives of the desired solutions and corresponding acceptance criteria [16]. At the very beginning of this work in 2016, the non-existence of standards and practical experiences in fog computing required to shape new approaches, which extend principles of cloud computing and the IoT. For that, it was needed to identify trends and to synthesize dependencies between fog computing, cloud computing and the IoT.

Artifacts

The next step is to specify conceptual architectures, artifacts and their intended functionalities, workflows, formal methods, and consider best practices [165]. Software artifacts need to address shortcomings of existing approaches in cloud frameworks, to meet IoT demands and implement fog computing.

Methods

In order to specify requirements for artifacts, we need to consider existing surveys, creative methods, and experiments [85]. At the beginning of the work on this thesis, fog computing has been at its origins. Only few related works have been available to identify opportunities to advance in this domain. Regarding resource provisioning, we model the fog, determining a class of related problems and applicable optimization methods.

Experimentation

We investigate empirically with simulations and experiments the value of created solutions and methods, where original objectives are compared to the received results [110, 113]. Different performance indicators are defined in advance and calculated during experiments. Improvement possibilities and limitations are identified as an option for future research.

Qualitative Evaluation

We evaluate prototypes and resource provisioning methods functionally and regarding their performance. Among quantitative metrics of performance, we consider the communication delay, makespan duration of services, execution time of IoT applications, QoS violations of user-defined deadlines for application execution, and utilization of fog resources. The evaluation involves performing experiments with different configurations of the fog and by applying different workload with IoT applications.

Research Contributions

Successful research outcomes become contributions, which enable real-world solutions as well as advance the already established knowledge with new original inputs. Here, the key is to perform rapid prototyping and at the same time to work on generalization of inputs as holistic models and principles, and to place them in the context of the declared global research goals [16]. This dissertation encompasses multiple artifacts, models, and algorithms that contribute to the research goal of realizing resource provisioning in fog computing.

1.4 Dissertation in Brief

The synopsis of the thesis is shown in Fig. 1.1. It highlights interrelations between different sections by categorizing them with informative icons. The dissertation starts with introducing foundations in fog computing in Chapter 2. This chapter ends with the specification of the requirements towards achieving the research goal of resource provisioning in fog computing. The main part of the dissertation consists of the three chapters corresponding to each of the declared contributions from Section 1.2. In Chapter 3, resource provisioning in the fog is investigated. First, the fog landscape components are abstracted and described, their main functionalities and interactions within the fog are introduced. Resource provisioning in fog computing is discussed, and a first resource provisioning model in a fog landscape is created. A heuristic first-fit placement algorithm to place workload on the available resources at the network edge according to this model is implemented and evaluated. Next, in Chapter 4, the FSPP is formalized. To solve this problem, a GA, a first-fit algorithm, and an exact mathematical resolution of the model are implemented and evaluated. The described architecture of the fog and requirements to realize resource provisioning in fog computing are implemented within the fog computing framework FogFrame in Chapter 5. The framework and the performance of the resource provisioning approaches are evaluated by the means of a real-world testbed for fog computing. In Chapter 6, the state-of-the-art research is discussed and compared to the contributions of this dissertation. The overview of the related work is devoted to the two main research directions: resource provisioning in the fog and fog computing frameworks. Finally, in Chapter 7, the contributions of this dissertation are summarized as well as future research advances are proposed.

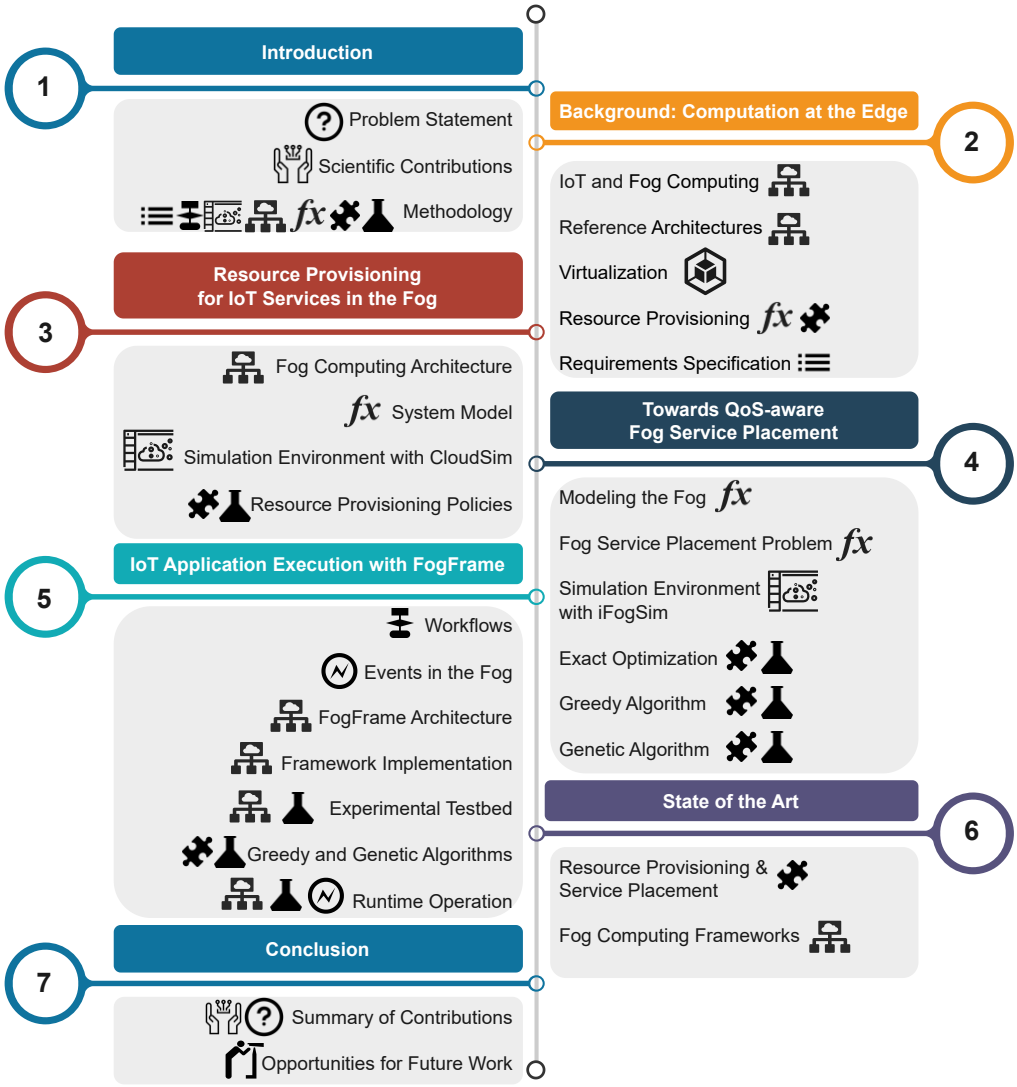

Figure 1.1: Synopsis of the thesis.

CHAPTER 2

Background: Computation at the Edge

Fog computing mirrors the structure of the IoT, where a multitude of heterogeneous connected devices cooperate. The centralized processing of high-volume, high-velocity and high-variety of IoT data in the cloud incurs high delays and accordingly low speed of data processing, which are unfavorable for IoT applications and services [95]. Fog computing promises to solve this problem by utilizing available computational, storage, and networking resources for the enactment of IoT services close to the edge of the network in combination with cloud resources [122]. Therefore, while mirroring the IoT, fog computing also aims to provide the benefits of cloud computing, i.e., virtualization, orchestration, manageability, and efficiency.

Fog computing employs the idea that computational resources from IoT devices can be offered in a similar manner as resources in the cloud. For rather resource-constrained IoT devices at the edge of the network, a promising solution for this issue is to utilize containers as a virtualization mechanism [21, 103]. However, the deployment and execution mechanisms at the edge of the network and in the cloud are different [117]. This chapter aims to provide a foundation and understanding of the basic concepts and features of the IoT, edge computing and cloud computing. These foundations and elemental mechanisms become a basis for the definition of principles of fog computing.

In the fog computing community, great attention is devoted to existing standards and Reference Architectures (RAs) to reuse the already obtained experience. Hence, the RAs for fog computing are also in focus of this chapter. We investigate how the ETSI MEC framework [54, 55, 56, 57], the OpenFog RA [42, 82], and

the FORA RA [115] provide communication and control mechanisms necessary in the fog.

Service deployment and execution mechanisms at the edge of the network and in the cloud are different, since the hardware used in these environments differs [117]. Therefore, we provide an overview of existing virtualization technologies and discuss suitable technologies for the fog computing environment.

In order to enable efficient resource provisioning in a fog landscape, networking issues, i.e., considering throughput, bandwidth and latency, as well as networking topologies and different types of hardware need to be taken into account [88, 124]. In this chapter, we overview the processes and methods to enable resource provisioning. This is achieved through environment modeling and finding feasible solutions through either optimization approaches or heuristic algorithms and meta-heuristics.

The final part of this chapter is devoted to summarizing requirements needed to realize resource provisioning in fog computing based on the foundations from the fields of the IoT, cloud and edge computing. We will use those requirements to provide necessary context and argumentatively discuss the contributions in the remaining chapters of this dissertation.

2.1 Internet of Things

Advances in the IoT

The IoT is a disruptive paradigm that has been developed for more than 20 years [146]. It leads to an ever-growing presence of ubiquitous networked computing devices in public, business, and private spaces, which are able to act autonomously and provide network connectivity [28]. Significant research in the IoT has been devoted to the system aspects of the IoT, i.e., networking, middleware, programming models, and cloud aspects [64]. IoT devices do not simply act as sensors, but feature computational, storage, and networking resources, which they can expose for free or under incentives. The main features of the IoT are the distribution of devices over large geographical areas, heterogeneity and the size of devices, their redundancy to accommodate uninterrupted measurements or failures. The mentioned communication-related features together with the IoT application-related features form a backbone of all the challenges to be solved when applying the IoT in practice, i.e., the latency-sensitivity of IoT applications [18, 27]. The IoT aims to provide communication, interaction, data transfer, and control between IoT devices at the edge of the network, e.g., sensors, actuators, mobile devices. It has been projected that at the end of 2022 the number of connected IoT devices reached 16.4 billion[1], and more than 30 billion devices will be installed and connected to the Internet until 2025. New technologies, e.g.,

[1]Online; Accessed: Apr. 2023 https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/

the roll-out of 5G, provide new networking environments, which accelerate even more the proliferation of the IoT. One particular area of the IoT that aims to enable industrial applications for collecting sensor data, actuating and monitor devices is the Industrial Internet of Things (IIoT). The difference between IoT and IIoT is in the requirements towards real-time processing, resource capabilities, connectivity [170]. And the aim of the IoT and IIoT is to accommodate those new technologies, to provide networks for the multitude of devices for sharing data with other devices or systems to provide sensing of environmental data in smart cities [63], monitoring of health conditions of people in healthcare [37], monitoring of production processes in industry and manufacturing [41], and various other small- and large-scale applications.

Challenges

The IoT highly depends on the underlying technologies available at the edge of the network. The development of these technologies dictate the challenges of the IoT data processing. Major challenges are [28, 61, 154]:

- Adhering to the size of devices and their capabilities,
- Enabling a high level of security,
- Achieving fault tolerance,
- Need of low latency and available bandwidth,
- Interoperability.

Other challenges include the ease of IoT application deployment, real-time processing, and mobility. Latency-sensitivity can be named as one of biggest challenges. The data transmission time between sensors and data sinks needs to be efficiently minimized. This can be achieved either by minimizing the distance between sensors and data sinks or by increasing the speed of data transmission.

IoT and Cloud Computing

IoT data is sourced in a distributed way, i.e., the sensors deployed in the IoT generate a large volume of data with great velocity and veracity. This data is then sent to a centralized cloud for processing, and delivered to the distributed stakeholders or other distributed IoT devices, often located close to the initial data sources. This centralized processing approach results in high link delays and low data transfer rates between IoT devices as well as IoT devices and potential stakeholders [26]. Therefore, Big Data processing becomes an important challenge of IoT systems and requires specialized techniques in transferring data from data sinks to the cloud, in enabling high-performance processing and in efficient storage [125].

IoT and Edge Computing

Pre-processing of data generated by IoT devices close to data sources becomes a crucial issue in IoT scenarios, as well as the integration of data analytics solutions to enable real-time analysis of received data [125]. Edge computing provides computational capabilities locally in the network and, therefore, reduces considerable computational stress on the centralized cloud [76]. The data transfer between IoT devices and data sinks becomes reduced due to edge computing, where data sinks become part of the network. The data is either processed at

the edge or pre-processed at the edge and then off-loaded to the cloud or to remote data sinks. In both cases, the latency is considerably decreased with edge computing.

2.2 Fog Computing

Concept

While edge computing provides needed localization and narrows the scope and time of processing enabling low latency and context awareness, the cloud provides global centralization, where data processing widens in scope, coverage, and time [25] (see Figure 2.1). Many applications require both localization and globalization, particularly for Big Data analytics. In order to implement delay-sensitive smart environments, 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 [21, 25, 111].

Fog Landscape

Fog computing has been in research for ten years, when Bonomi et al. [26] in 2012 published their seminal conceptual work on this topic, coined the term fog computing, and introduced a layered model bridging the IoT and the cloud. The authors discussed that IoT applications are naturally distributed and may be placed in the cloud and at the edge of the network. Ubiquitous IoT devices generate data at an exponential rate. This imposes new requirements towards computations, performance, and network congestion at the edge of the network, parallel computing enablement together with the problem of low capacities of computing resources at the edge [97]. Therefore, in Big Data, wide geographical distribution becomes one of key characteristics that promotes moving of data processing closer to the data [89]. Figure 2.1 shows such a new hierarchical organization of data processing based on different levels in the network and different capabilities of computing devices.

This combination of the edge- and cloud-based computational resources that cooperatively enable decentralization and execute IoT applications is also known as fog computing. Together, these resources form a fog computing environment, or a so-called *fog landscape* (see Figure 2.2).

Mirroring IoT and Cloud

Fog computing mirrors the basic structure of the IoT, where a multitude of heterogeneous, networked devices cooperate [12, 45]. Single IoT devices coordinating a group of other IoT devices and providing virtualized resources, are located close to the network edge. These devices allow executing IoT services close to the data sources and sinks, instead of involving the cloud. This leads to decreased delays, as well as to the better utilization of already available computational, storage, and networking resources in the fog [143]. Potential use cases for fog computing include typical IoT scenarios, e.g., data pre-filtering in Big Data use cases [40, 89] or pre-processing of data streams from sensor nodes [74]. While mirroring the IoT, fog computing also aims to provide all the benefits of cloud computing, i.e., containerization, virtualization, orchestration, manageability, and efficiency. The

difference between edge and fog computing is that the fog is hierarchical, includes the interactions with the cloud, and tackles the issues with networking, resources and control, while the edge is limited and excludes the cloud [42].

Principles

Fog computing is based on common principles from the field of cloud computing, most importantly virtualization [39, 161]. In the cloud, physical machines are provided in terms of VMs. Fog computing employs the idea that computational resources from edge devices can be offered in a similar manner. However, since VMs are resource-intensive, they are not the best virtualization approach for rather resource-constrained edge devices [142]. A promising solution for this issue is the utilization of *containers*, for example, Docker containers, as a virtualization mechanism for edge resources [25, 103, 158]. Accordingly, in order to provide a practical framework for fog computing, it is necessary to introduce mechanisms both to manage a fog landscape and to execute distributed IoT applications in the fog using virtualized resources. This requires mechanisms for resource allocation, decentralized service placement, deployment, and execution in a fog landscape.

OpenFog

In 2017, the notions of fog computing were brought together into the OpenFog RA, which later evolved into the IEEE Standard for Adoption of OpenFog Reference Architecture (IEEE 1934–2018) [82]. The definition of fog computing per IEEE 1934–2018 [82] is as follows:

> "Fog computing is a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum."

In this thesis, we adopt this definition.

Requirements

Before diving into fog computing implementation, it is necessary to understand fundamental requirements towards a fog landscape and its operation. In the work of Byers [33], a description of conceptual imperatives for fog computing is provided. Al-Qamash et al. [8] provide a comparison of non-functional requirements between cloud computing and fog computing paradigms. These requirements also are the basis of the fog computing pillars as described in IEEE 1934–2018 [82] and aim to provide a road-map to design and implement a fog landscape. In the following paragraphs, the requirements are grouped according to their purpose and reflection in the implementation of a fog landscape.

Communication The first conceptual imperative to be considered when building a fog landscape is *communication*. This principle requires to address delay-sensitivity and bandwidth problems in a fog landscape. IoT applications are delay-sensitive. Therefore, in order to make fog computing feasible for the IoT, it is necessary to provide communication delays between devices in a fog landscape measurable in tens of milliseconds. With regard to the bandwidth, data transfer to the cloud from devices at the edge of the network becomes insufficient for IoT

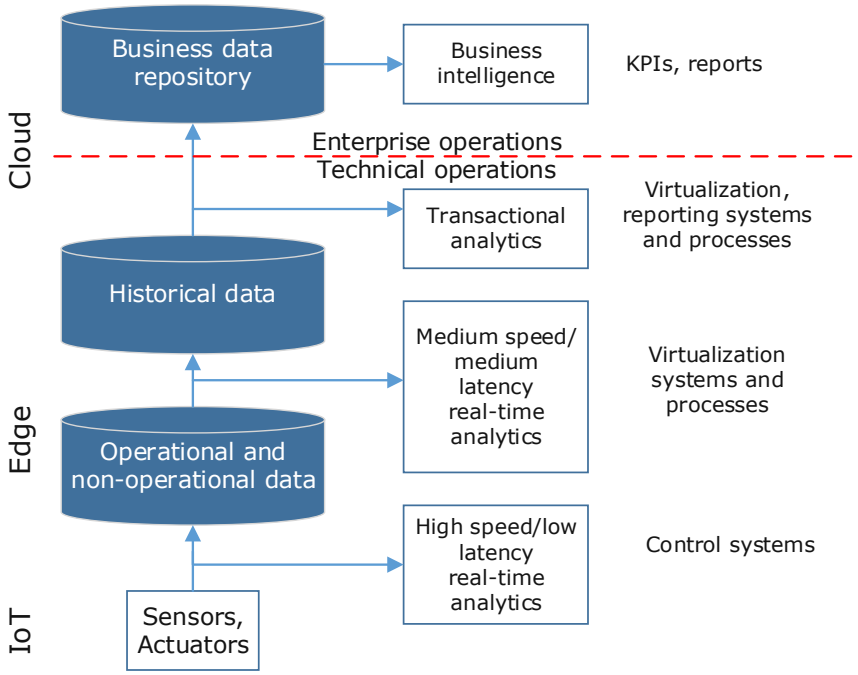

Figure 2.1: Application scope in fog computing. Adapted from [25].

applications. Therefore, a fog landscape aims to reduce stress on data transfer by providing computing power to process data explicitly near data sources and sinks.

Geography Another imperative is *geography*. This principle requires a fog landscape to control the information flow based on physical and logical boundaries, and tells that there is no valid reason to send data beyond those boundaries. In other words, data has more value close to its source rather than at a larger vicinity. Boundaries to process data in the fog landscape have to be clearly defined based on the location of devices at the network edge.

Quality The third conceptual imperative is *quality* and includes reliability and stability principles. These principles require to maintain uninterrupted operation in a fog landscape. This necessitates adaptive approaches to migrate applications in response to runtime events in the fog landscape by applying fault tolerance and redundancy scenarios at different levels of the fog landscape. Redundancy in the fog ensures the integrity of deployments of large applications providing reliability at scale.

Performance The next group of principles belongs to the *performance* conceptual imperative, which includes agility and scalability in a fog landscape. Agility requires fast reaction on runtime events and on changes in the fog landscape.

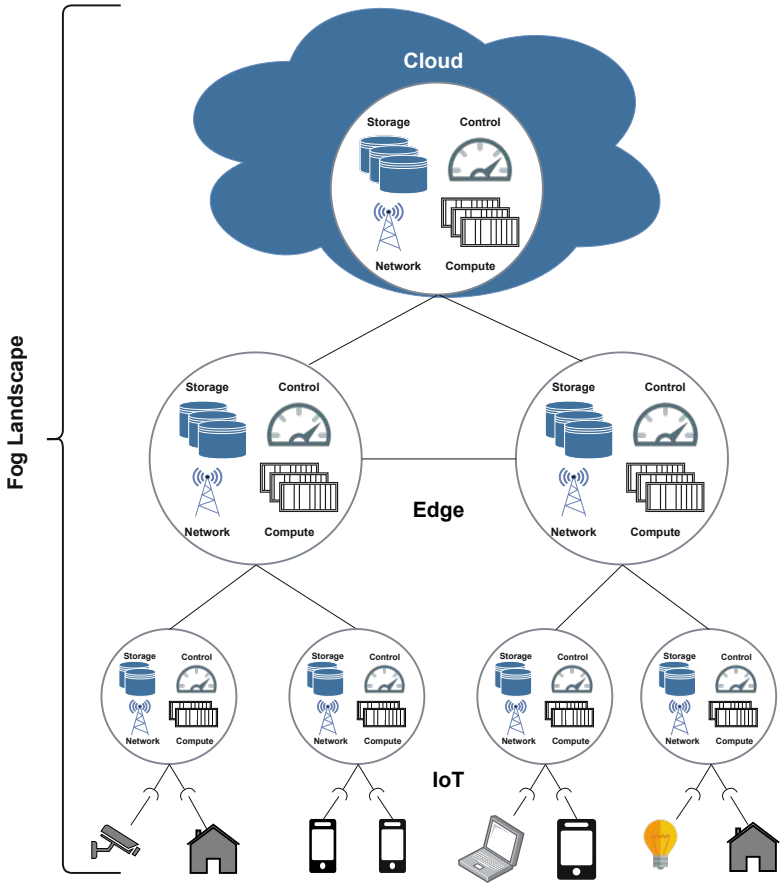

Figure 2.2: An overview of fog computing.

Scalability is considered from multiple points of view, e.g., of physical devices and software. The fog landscape is expected to grow continuously, and therefore its architectural design has to be adaptive to such growth.

Virtualization One more imperative is virtualization. It requires a fog landscape to have a pool of resources represented as virtual resources. These resources have to be orchestrated according to workload demands and currently available devices in a volatile and fluctuating fog landscape.

Hierarchy The *hierarchy* imperative considers the hierarchical topology of a fog landscape and the modularity of its resources. Within its architecture, the fog landscape has to consider interoperability and mobility of its resources, easily adding new resources according to workload demands. The hierarchy in fog computing supports autonomy of fog devices, and different fog devices at different levels of the fog hierarchy can perform different decision making activities. This brings the intelligence from the cloud closer to the network edge and to IoT

devices. This intelligence can facilitate device discovery in the fog on different fog levels, provide means for orchestration of services and resource provisioning, and enable local decision making thus ensuring reliability and eliminating the cloud as a single point of failure.

Openness The last group of principles to be considered is *openness*. Openness is important to enable fog computing for IoT applications. Reducing risks of vendor lock-in and standardization has to be of ultimate importance to the fog computing community. Fog computing should support portability of applications, modularity and dynamism of deployments. This can be achieved by development of testbeds, creating unified standards and providing open source solutions. Open source solutions and networks have to lay a pathway towards industry-wide adoption of fog computing.

Security A separate imperative is *security*. It requires establishing access control and ensuring integrity of every device in a fog landscape, accounting for data privacy, enabling threat and data breaches detection. Fog computing introduces complex networking topologies of different resources. Therefore, a chain of security and trust needs to be enabled from one fog layer to another, and all the way to the cloud. Since the nature of the fog is volatile, meaning devices may appear and disappear in the fog landscape, the hardware and software of the devices must be verified to enable full trustworthiness. If such a verification is impossible, devices cannot become members of the fog landscape. The security group of principles is out of the scope in this dissertation and is addressed only partly by establishing a secure connectivity between devices in a fog landscape in FogFrame.

Challenges While the basic idea and theoretical foundations of fog computing are already well-established, there is still a lack of concrete solutions to implement fog computing in practice. Apart from the question of how to abstract and virtualize resources offered by IoT devices, another major barrier for the uptake of fog computing is the question how to distribute IoT applications onto available fog resources accounting for the discussed imperatives and constraints. Particular challenges of adopting fog computing are the adherence to geographical distribution of IoT data sources, delay-sensitivity of IoT applications, and the potentially very large volumes of data emitted by IoT devices. The fog landscape needs to be adaptive and scalable, and to account for the geographical distribution of its computational resources. The fog landscape has to provide an operating environment for the execution of IoT applications. Therefore, resource provisioning approaches need to account for those aspects to achieve an efficient fog landscape. Having built such an architecture, the focus in the work has to be on the development and experimentation with different resource provisioning approaches. Resource provisioning and service placement are major research challenges in the field of cloud computing [35, 94, 166]. While these approaches offer interesting insights, there are certain differences between the edge and the cloud. These differences prevent a direct adaptation of cloud resource provisioning solutions.

Differences to Cloud Computing

First, the size and type of cloud resources is very different from their counterparts in fog computing. While cloud resources are usually handled on the level of physical machines, VMs, or containers, fog resources are usually not as powerful and extensive. Second, a fog landscape 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 provisioning in the fog. This is especially important since one particular reason to use fog computing in IoT scenarios is the higher delay-sensitivity of computations at the edge of the network [21]. Hence, resource provisioning approaches for the fog need to make sure that this benefit is not foiled by extensive data transfer times and cost. Having in mind that storage, computational resources, and network demands are volatile, resource provisioning for fog computing needs to account for peak resource demands and utilization gaps. The basis for such a smart resource provisioning that eliminates over-provisioning lays in the characteristic of the elasticity of the cloud, i.e., minimizing cost, latency and delays, while maximizing QoS. A lot of work on the elasticity in the cloud has been introduced [75], however, elasticity in fog computing environments remains a challenging topic in research.

Differences to MCC

Resource provisioning is also an important topic in Mobile Cloud Computing (MCC) [59], which integrates mobile devices and cloud resources, and offers solutions for offloading tasks from mobile devices to the cloud [50]. However, MCC is mostly based on a rather simple network topology with direct communication between mobile devices and the cloud. Neither grouping and cooperation of devices, nor the different layers observed in fog computing are taken into account in MCC. Therefore, again, these resource provisioning approaches in MCC offer interesting insights and ideas, however cannot be directly ported to the field of fog computing.

Delivery Models

Having discussed general fog computing principles and challenges in resource provisioning, it is necessary to focus on service delivery models for fog computing.

VISP The first model to be discussed is stream processing. It requires the creation of a processing topology that includes IoT data sources and operators. In an IoT ecosystem for stream processing called VISP [74], complex network topologies are analyzed, and services are placed on resources according to QoS constraints. VISP is an interesting example of a real-world testbed which can be used and adapted to the needs of fog computing.

DDF The Distributed Data Flow (DDF) programming model [64] is a basic application model which resembles the stream processing approach, and provides means to create and execute IoT applications. An application according to the DDF model is a directed acyclic graph where vertices represent the single steps of applications to be executed in the flow, i.e., corresponding to services, and edges between vertices represent data transmissions and control flows between those

services [91]. A more structured IoT information model [46] can be used to enrich DDF, e.g., for service association discovery and monitoring, for reasoning upon semantic annotations, and for decision-making processes. It is based on semantic annotations and divided into an entity model, which aims to establish basic physical entities and relationships in the IoT infrastructure, and a resource model, which represents software artifacts corresponding to those physical entities.

NVF A different service delivery model is provided by the Network Function Virtualization (VNF), where physical network infrastructure is decoupled from its functions, and where services are split into Virtualized Network Functions (VNFs) [67]. This approach influences both the conceptual architecture of the fog computing environment and the modeling of IoT applications. VNFs can be considered as containers with software corresponding to resources on different levels in the fog. The eventual integration of the fog with NFV is assured by the need for flexibility and scalability providing fog-enabled NVF [119].

Edgeflow A more recent approach towards providing a methodology to develop and deploy IoT applications is presented in the framework EdgeFlow [13]. It is aimed to assist developers to divide IoT application functionalities into multiple parts following Flow-based Programming (FBP), i.e., single services, and to assign to those services needed QoS and resource-consumption parameters.

Delivery models are crucial to enable IoT application distribution in a fog landscape. They aim to provide the methodology and assist in splitting an IoT application into single cooperative services, which wrap sensing and processing functionalities, and can be independently placed and executed in the fog. Delivery models provide specifications of workloads for resource provisioning. In this dissertation, we follow the DDF model [64].

Cloud Workflows Recent cloud workflow modeling approaches can also be applicable to fog computing. In the survey of Menaka and Kumar [102] an overview of different levels of workflows is considered when distributing an application in the cloud. Similarly, such levels can be introduced when considering the fog and IoT applications.

2.3 Reference Architectures

In this section, we provide an overview of how the ETSI Mobile Edge Computing (MEC) framework, the OpenFog RA, and the Fog Computing for Robotics and Industrial Automation (FORA) RA consider the communication and control mechanisms in a fog landscape.

2.3.1 Mobile Edge Computing Framework

Standard

The MEC [56] set of standards provides necessary definitions, principles, functional and technical requirements, a RA, and an application lifecycle in MEC. It is

necessary to take into account these standards when talking about fog computing, as these documents provide an established groundwork and document all necessary interactions of MEC. In the following, we reflect on how the principles identified in MEC can be applied to communication and control mechanisms in fog computing.

Reference Architecture

For communication mechanisms, ETSI MEC [55] provides the mobile edge host level management, which aims to handle the management of mobile edge-specific functionality. The reference points to describe this functionality include platform configuration, management of virtualization infrastructure, request handling, and relocation management of applications. These points are mentioned in the MEC technical requirements specification standard [54]. In general, MEC deals with the specification of requirements for the application lifecycle management, i.e., the lifecycle of applications running inside MEC, rather than on the details of how to establish communication within the infrastructure of MEC.

2.3.2 OpenFog Reference Architecture

Vision

The OpenFog RA [42] for fog computing has been introduced in 2016 and is aimed to close the gap in the cloud-to-thing continuum, which is not considered by MEC, i.e., the gap between the cloud and IoT devices at the edge of the network. The OpenFog RA uses the benefits of cloud computing and provides a certain level of independence and reliability at the edge of the network. Fog computing is considered as an umbrella paradigm, which includes cloud computing, edge computing or MEC, and the IoT. The RA explicitly is focused on an execution environment for fog computing. The hierarchy pillar of the OpenFog RA considers different IoT end devices to be present in a fog landscape, and provides monitoring and operational support in the fog. A hierarchy in computational resources and systems is not necessary in the OpenFog RA's vision of fog computing. However, exactly because of the logical hierarchy that can be established by IoT devices, cloud computing can be amplified and eventually substituted by a fog computing architecture.

Enablers

The OpenFog RA (see Figure 2.3) represents various requirements and components that on different levels contribute to the fog. It consists of four main enablers:

- Fog landscape hardware marked in red in Figure 2.3,
- Fog node management marked in blue,
- Application management in gray,
- Application services in white.

Requirements towards the fog hardware infrastructure include environmental issues, hazardous safety, and modularity to enable configurable hardware capabilities, which is not in the focus of this dissertation. Fog node management enables general functionalities, configurations, and management of a node and its communications with other nodes. It forms the software backplane that is required to enable virtualization and execution of any application on the node,

and provides node-to-node communication. Application management includes resource provisioning and management of the application lifecycle, i.e., resource allocation, source image management, deployment, execution, and verification. Last but not least, application services are containerized instances, i.e., single services of IoT applications that are executed within the fog. Supporting such services means providing ordinary duties during service execution, such as their registration and cleanup, data transfer, and logging.

Standard

The breakthrough of this architecture was its adoption by the IEEE Standards Association which resulted in the IEEE 1934–2018 Standard for Adoption of OpenFog Reference Architecture for Fog Computing [82]. This document establishes the OpenFog RA as an industrial standard. This standard establishes a necessary basis and a blueprint for fog computing, specifically for industries enabling the implementation of fog computing applications and infrastructures. This standard provides acceleration, innovation, and market growth in the IoT and cloud computing under the umbrella of fog computing. It clearly defines terminology developed in the OpenFog communities, and the relationships between the entities of fog computing with regard to the IoT applications along the things-to-cloud continuum. This document also introduces a high-level taxonomy of these concepts and constructs. This standard enables the fog computing community both in industry and academia to follow the same terminology and principles and to eliminate subjective interpretation of any of the fog computing terms or concepts.

Lifecycle Management

Compared to the MEC framework, which is focused mainly on application lifecycle management, the IEEE 1934–2008 standard as well as the OpenFog RA consider the management lifecycle of each device in the fog landscape. The management lifecycle of a device according to the RA consists of four phases:

- The pre-life phase when the device is being instantiated, i.e., basic settings are adjusted, identification is set up, security certificates are generated, and adherence to the specified Application Programming Interface (API) is checked;
- The early-life phase when the device is added to the fog landscape;
- The functional-life phase when the device is used to execute applications;
- The end-of-life phase when the device is decommissioned, i.e., cleaned of all application deployments.

Relation to the Thesis

In this dissertation, we follow the OpenFog RA in terms of its concept of a cloud-to-thing continuum. First, the focus is on the pre-life and early-life phases of device management in the fog. Next, the functional-life phase is considered and application management mechanisms are described and implemented. One more point which goes beyond the OpenFog RA is that we discuss latency sensitivity and location awareness. However, within the OpenFog RA this is considered on the design level within the hierarchy pillow, and in this thesis we focus on the

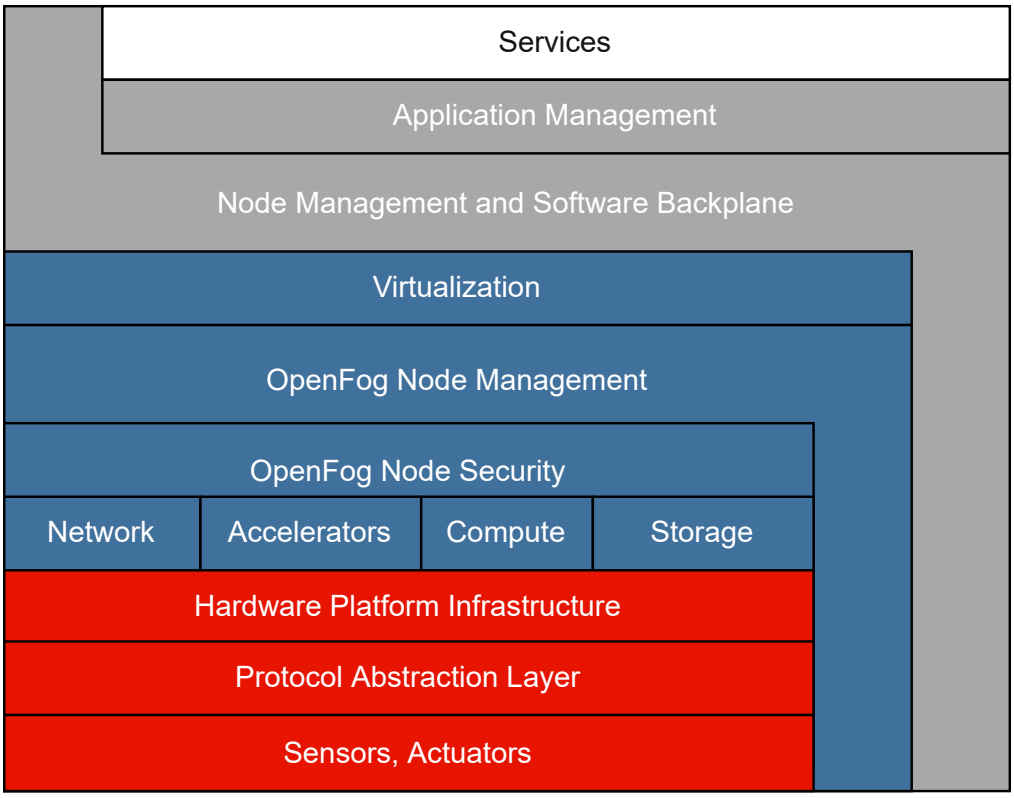

Figure 2.3: OpenFog reference architecture. Adapted from [42].

resource models and optimization, dynamically tackling latency and delays with resource provisioning.

2.3.3 FORA Fog Computing Platform

Reference Architecture

FORA provides a RA for fog computing targeting the IIoT. It has been specified within the Horizon 2020 research and innovation programme FORA – Fog Computing for Robotics and Industrial Automation [115]. The FORA fog computing platform is built upon the principles of deterministic virtualization and execution, deterministic networking and communication, interoperability, security and fault tolerance. It adheres to open standards such as Time-Sensitive Networking (TSN) and the Open Platform Communications Unified Architecture (OPC UA).

Perspectives

The FORA RA covers multiple perspectives: devices in the fog landscape, or so-called *fog nodes*, resource management and orchestration, and services. FORA considers different classes of fog nodes which operate on different levels and have different functionalities and resources. Fog nodes of Class-1 are located in close proximity to sensors, control real-time tasks, and operate with Machine-to-Machine (M2M) protocols, e.g., MQTT, Constrained Application Protocol

(CoAP), or Open Mobile Alliance (OMA). Class-2 fog nodes perform non-critical real-time data collection and analysis. Class-2 nodes' runtime is on a factory shop floor level. They possess more computational power than the resource-constrained Class-1 nodes. Fog nodes of Class-3 aim at the most resource-intensive tasks such as data analytics, long-term data preservation, and communication with cloud resources. They have considerable computational power and require high network throughput. Each of the classes work with the hypervisor technology and operate with VMs.

Runtime FORA provides a runtime environment to orchestrate applications, allocate resources on-demand, taking into account network balancing and self-management. The main goal of the FORA fog computing platform is to ensure the balanced usage of fog resources. Therefore, applications to be executed within this platform are analyzed based on their requirements, and distributed between the network edge and the cloud. Monitoring of the fog nodes on all levels recognizes the volatility of the fog computing environment, and provides fault tolerance mechanisms and describes augmentation of fog resources with other technologies.

Enabled Services FORA enables platform services for resource monitoring, safety and security monitoring, and edge analytics in the fog computing platform. Individual fog nodes and the network need to be monitored providing insights into usage of all the resources. New challenges come together with the fog computing paradigm, where computational resources and networking are not isolated any longer thus making the infrastructure vulnerable to attacks. Another type of services addressed here are edge analytics services. As the IIoT generates high volumes of data and due to the bandwidth constraints, processing and industrial automation in the cloud for Industry 4.0 may become difficult for the existing solutions from, e.g., Google or Microsoft. The RA divides the data processing between fog nodes, and aims to move from hierarchical architecture towards completely distributed processing, for example, for predictive maintenance use cases.

2.4 Virtualization Technologies

Lightweight Virtualization A prerequisite of the elasticity of distributed environments, e.g., cloud computing, is virtualization [32, 157]. Service deployment and execution mechanisms at the edge of the network and in the cloud are different, since the hardware used in these environments differs. Therefore, this section provides an overview of existing virtualization technologies. Benefits and drawbacks of the technologies are considered and compared in order to justify using lightweight container technology in a fog computing environment. In order to implement a fog landscape and to enable an easy deployment of services, a lightweight virtualization technology is necessary [144]. Virtualization in the cloud can be used within a fog landscape when resources at the edge of the network are exhausted. For a fog landscape itself, using cloud VMs becomes unacceptable due to high start-up times of VMs

and their resource consumption. Therefore, a lightweight virtualization concept needs to be applied, i.e., containers or unikernels [103].

Unikernels One option for virtualization, which recently has gained more attention, are *unikernels* [73, 103]. Unikernels are independent specialized cores of an operating system and they represent a solution that aims to minimize complexity of virtual resources [73]. However, each unikernel is compiled manually for a single process and requires very specific expertise [65]. Unless usability tools are developed to popularize this concept and reduce engineering efforts when using unikernels, it stays practically impossible to justify using unikernels in large-scale scenarios such as fog computing. Kuenzer et al. [92] introduce an operating system called Unikraft that aims to reduce engineering of unikernels. Unikraft provides modular abstractions together with a convenient API to ease any needed customizations of unikernels. Considering the architecture of Unikraft as described in [92], it splits the classical operating system functionalities into micro components that can operate only within certain API calls.

Docker A promising solution to tackle the issue of usability of unikernels is the utilization of lightweight *containers*, e.g., Docker containers, as a virtualization mechanism for resources at the network edge [25, 103]. Compared to unikernels, containers operate by the means of a convenient API, which allows using this technology in any domain. Multitude solutions proposed in the state of the art are based on the Kubernetes system for containerized applications [124, 114, 152]. In such a system, a centralized Kubernetes[2] server performs the management, scaling, orchestration of containers and storing of resources. However, such a system is heavyweight and requires high maintenance efforts for the deployment in a resource-constrained fog landscape. The particular challenge is to address various environments of a fog landscape. It is necessary to introduce a fog computing framework that provides a decentralized solution, unlike the centralized Kubernetes cluster, and enables communication and interaction between different virtualized resources, as well as ensures service execution in those environments. Some approaches have been introduced to enable federated Kubernetes scheduling for the IIoT, which is a multi-resource environment with multi-connections and delay sensitivity. For example, in the work of [170], a scheme of scheduling with Kubernetes for the IIoT is introduced. However, this scheme is executed within a centralized master node of the Kubernetes cluster. This master node upon a user request performs the screening of the connected environment, among the filtered nodes does the scoring by adherence to resource constraints, and finally chooses the node with the highest score. With regard to this thesis, this approach could be implemented on the fog colony level, however, it would not allow hierarchy within single fog colonies.

Interest towards container technology is also boosted by the ever-growing demand towards technologies that appear in big flagship enterprises, e.g., Google, which

[2]Online; Accessed: Apr. 2023 `https://kubernetes.io`

made the Docker container technology together with Kubernetes a very popular open source project [22]. Docker containers can execute only Linux processes, however, they can run within various operating systems. Docker containers enable efficient software development and deployment.

Java Containers Among other technologies available with the similar concept of containers, there are autonomous applications based on Java containers, i.e., Wildfly and the Spring Boot technology. These technologies allow executing containerized Java applications without the necessity to have a dedicated Java environment [140].

LXC Containers LXC is another container platform. LXC containers are created by the means of the same tools as VMs, however they enable high performance during the execution of workloads [128]. Compared to Docker containers, the management of LXC containers does not need any additional orchestration system. LXC is very close to the full operational environment of hypervisors of VMs [43, 112]. It provides virtualization of network interfaces and data storage interfaces. Inside an LXC container, it is possible to launch several Docker containers.

OpenVz Containers OpenVz[3] is one of the oldest and still usable container platforms for launching multiple full-scale operating systems on one physical server to ensure security and isolation of different applications. The isolation means that those containers have root access, users, IP addresses, memory, processes, files, applications, system libraries and configuration files. However, since these containers share the core of the host system, OpenVz containers are much faster and more efficient than traditional hypervisors [112].

Rocket The application container engine Rocket (or rkt[4]) is a security-minded container technology introduced in the CoreOS platform to facilitate integration between different production systems and to isolate vulnerabilities [43].

App Container In 2014, CoreOS published the App Container or appc[5] specification to stimulate innovation in the container space that spawned a number of open source projects [43].

OCI Initiative The Opensource Container Initiative (OCI) standard[6] has been founded in 2015 by Docker and CoreOS. It is intended to provide specifications for container technologies for the industry [43]. This initiative is supported by RedHat, Google, AWS, VMware and CoreOS, thus ensuring compatibility with Rocket.

[3] Online; Accessed: Apr. 2023 `https://openvz.org`
[4] Online; Accessed: Apr. 2023 `https://www.redhat.com/en/topics/containers/what-is-rkt`
[5] Online; Accessed: Apr. 2023 `https://github.com/appc/spec`
[6] Online; Accessed: Apr. 2023 `https://www.opencontainers.org`

CRI-O Project The consequence of the implementation of the standard OCI was the project CRI-O[7], which is a lightweight container runtime for Kubernetes. From the user point of view, both Docker and CRI-O implement Kubernetes Container Runtime Interface (CRI) and enable functionalities of loading and launching containers.

Windows Containers Windows Containers[8] work in unison with Docker to ensure the seamless operation of the Docker container management tools in Microsoft infrastructure [31].

Concluding this overview, using lightweight containers to enable fog computing is a necessity. In this thesis, Docker containers are chosen as suitable wrappers around services both on the fog entity level and IoT application level. The democratization and usability of the Docker technologies has been one of the decisive factors. In the future, fog computing can certainly benefit from using the unikernels virtualization technology provided that the enterprise usage and tools will be developed for it.

2.5 Resource Provisioning

Goals in Cloud Computing

In cloud computing, resource provisioning enables the allocation of virtualized resources for users according to needed workloads adhering to user-defined QoS requirements and agreed SLAs. Cloud computing offers on-demand rapid access to CPU, RAM, and storage resources shielding all the management overhead of acquiring and maintaining those resources for cloud users. In cloud computing, resource provisioning addresses the topics of selection, deployment, management, and runtime operation of a resource pool of hardware to ensure the desired operation of needed software for cloud users. The main feature of cloud computing is elasticity, i.e., the cloud is capable to adjust to the fluctuating workload by up- and down-scaling the needed capacities [19, 157]. Resource provisioning techniques may have different goals from the perspective of cloud users and cloud providers [79]. From the point of view of cloud users, the goal is to save cost of leasing computational resources and to provide rapid scalability. On the other hand, cloud providers aim to balance between multiple goals:

- Maximize their profits by efficiently using their available pool of resources taking into account needed SLAs,
- Minimize their risks by aiming at fault-tolerant resource provisioning and reducing SLAs violations, and
- Minimize energy consumption to tackle green computing challenges [51, 130].

[7] Online; Accessed: Apr. 2023 `https://cri-o.io`

[8] Online; Accessed: Apr. 2023 `https://docs.microsoft.com/en-us/virtualization/windowscontainers/about`

Resource provisioning can be on-demand when resources are provided reactively based on current workloads, and proactive, when resources are analyzed, workload is approximately predicted and resources are reserved in advance [93].

Static and Dynamic Provisioning

In cloud computing, different resource provisioning techniques have become best practices to achieve the desired QoS values for applications, i.e., static and dynamic provisioning [93]. Static methods tackle unchanged demand where the workload is known based on historical data or predictable in advance before the start of applications [118]. Static provisioning means a pre-defined number of resources (i.e., VMs) based on agreed SLAs, and uses fixed prices. Such provisioning aims at over-provisioning to guarantee a certain level of QoS; however, it has high operational costs and is exposed to shortages, i.e., underprovisioning, because of unmitigated changes in workloads. Dynamic methods resolve on-demand workloads providing automated scaling based on current demand, the available resource pool and needed parameters of applications [118]. The drawback of static provisioning is the 'bottleneck' of underprovisioning and overprovisioning due to specified SLAs. Dynamic provisioning addresses the issues of overprovisioning and underprovisioning of resources, however, it introduces challenges to predict future workloads of resources, to perform adaptations and to enable migrations of VMs accordingly [118]. Dynamic provisioning is proven to be NP-hard due to the combinatorial nature of choosing between available resources and needed constraints [7]. Resource provisioning approaches for the cloud are in the focus of many surveys, e.g., [19, 51, 76, 79, 124, 130, 157, 166].

2.5.1 Distributed Applications

Novel Use Cases

Emerging use cases with IoT and Big Data require using distributed applications, which evolved from very simple single client-server applications to more complex ones with multiple clients and several servers to collaborate. The goal of distributed computing environments is to provide a virtual computational layer between such complex applications and physical hardware. It is a commonly accepted concept that applications that require high performance and real-time processing are served by sharing the virtual computational layer of resources in a controlled manner. Applications formed from cooperating services, which exploit such a virtual computational loosely-coupled environment, are called distributed applications. In this dissertation, a distributed application is represented according to the DDF model [64], as discussed in Section 2.2.

Distributed Execution

One challenge of executing distributed applications is to establish an efficient mapping between services of distributed applications and virtual computational resources available in the resource pool (see Figure 2.4). In the figure, we observe services, i.e., sensing, processing, and storage services, distributed in the fog landscape and communicating with each other. The distribution of applications can be performed because of different quantitative and qualitative reasons, e.g., to achieve high performance by exploiting more computational resources or to

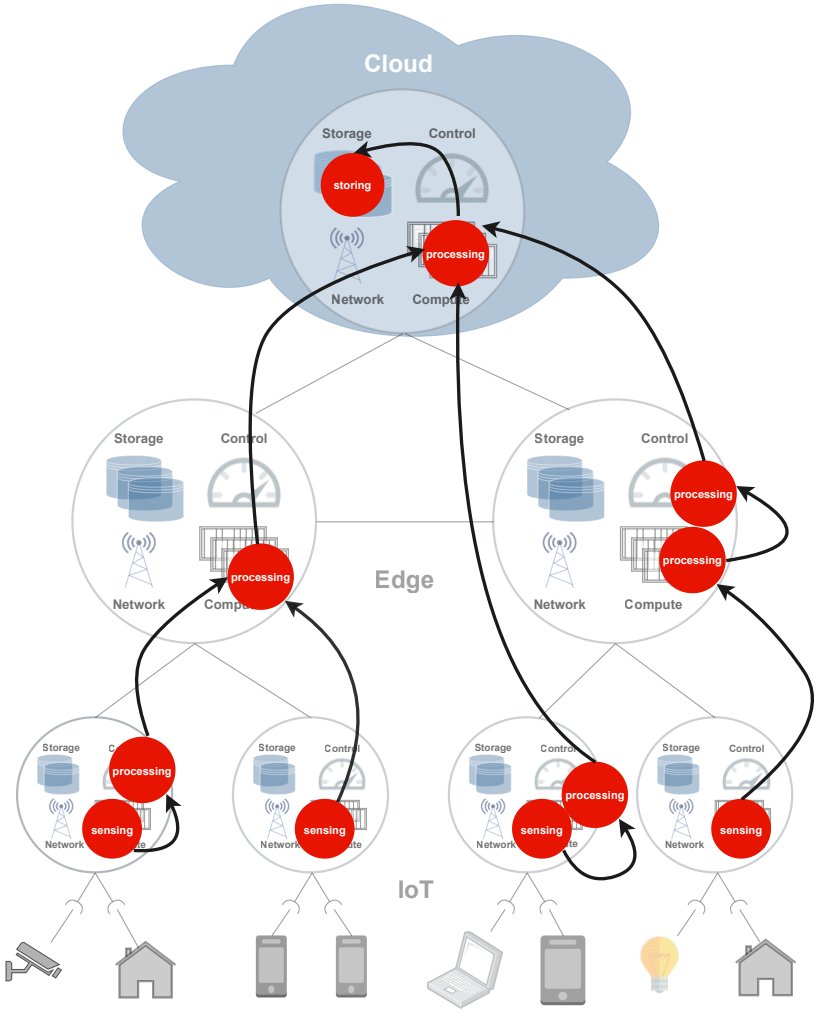

Figure 2.4: An example of a distributed data flow in a fog landscape.

use certain software or hardware resources that cannot be provided by local infrastructure [96]. To enable distributed applications, efficient application management needs to be established, including communication means and providing a holistic and unified abstraction of distributed computational resources.

2.5.2 Resource Provisioning in the Fog

New Environments

The advent of IoT-related use cases and the emergence of different distributed environments, e.g., edge computing and fog computing, makes it necessary to investigate novel mechanisms for resource provisioning. In all distributed environments there is a concept of a pool of computational resources. This

pool of computational resources can be represented as nodes, where each node may represent a collection of CPU, RAM and storage resources available for exploitation by applications. An example of a node can be a VM, which is constituted from certain CPU, RAM and storage resources taken from the available resource pool. The resource pool can be static when the number of nodes and available resources do not change, or dynamic as is often the case in fog computing, when the resource pool is fluctuating and volatile. In fog computing, resources can enter or leave the fog landscape or also change their capacities and performance. However, independently of the type of a distributed environment, the explicit assignment of services of distributed applications onto a distributed pool of resources has to be provided. This assignment needs to select available resources, i.e., match them according to defined constraints of resources and services. This process is tackled with resource provisioning approaches.

Fog Landscape

Following the basic structure of fog computing as presented in [25, 45], resource provisioning approaches are necessary to identify how services can be distributed in the entire fog landscape. Therefore, resources in the fog landscape have to be provided with means to select and allocate resources and schedule service executions. Apart from resource provisioning, the fog landscape needs to address infrastructural changes in the fog meaning monitoring overall resource utilization and reacting on resource volatility and mobility.

Provisioning Overview

Fog resource provisioning assumes the efficient usage of available virtualized resources in a fog landscape. For this, a system model has to be formalized aiming, e.g., to minimize delays arising from the transfer times between the network edge and the cloud, and to maximize resource utilization of existing resources. As in the field of cloud resource optimization, manifold goals for resource provisioning optimization are possible, e.g., time, cost, and energy efficiency optimization [1, 111, 141]. Together, these goals may form the foundation for a multi-criteria optimization problem. Based on the solution to this optimization problem, i.e., a resource provisioning plan, services are instantiated and executed on particular fog resources.

2.5.3 Problem Definition and Modeling

Operations Research

Resource provisioning problems belong to a certain kind of decision-making problems from the area of operations research [19, 79, 127, 155]. Operations research applies mathematical methods in order to justify decisions in an activity domain. The main characteristics of such problems are:

- A problem deals with a certain arrangement that pursues a concrete goal;
- In the problem a set of conditions, or so-called constraints, are given which characterize the context of the problem;
- Within the framework or confinement of these conditions, it is needed to make a decision in a way that the problem arrangement would be most

beneficial [134].

Stochastic and Deterministic Problems

An operation in this context is a set of mutually conformed arrangements aimed at achieving a concrete goal. An operation is always controllable, meaning that by performing such an operation, a set of parameters can be identified or chosen. These parameters are called controllable variables or so-called decision-making variables. A concrete set of these parameters becomes a solution of the problem. The parameters that form a solution are called the solution elements. Let x denote a collection of elements that form a solution. Operations can be characterized as well by uncontrollable parameters. Uncontrollable variables are the factors which cannot be controlled, e.g., weather conditions, and are often represented as pessimistic, measured, historical values. If uncontrollable variables are undefined, the model becomes stochastic, i.e., indeterministic, when systems evolve according to probabilistic laws [148]. If uncontrollable variables are known, then the model is deterministic.

Optimal Solution

An optimal solution is a solution that achieves the goal of the operation. In any operations research problem, constraints are given upfront and may consider financial, materialistic, technological, and human operation aspects in the domain. Constraints form a pool of feasible solutions for the considered problem. Let X denote such a pool of feasible solutions. The main task of solving the problem means to find such a solution x within the pool of feasible solutions X that from certain points of view is more efficient or more preferred than any other solution. In order to compare different solutions in X, a quantitative criteria of efficiency is needed to reflect the goal of the operation, i.e., the goal function. When the efficiency of a solution is estimated based on several criteria, the problem becomes a multi-criteria decision problem.

2.5.4 Research Stages

Formalization

The work on a problem starts with its identification, when the problem and goals are meaningfully formulated [113]. At this step, the inherent requirements of the considered system and environment are determined.

Models

The next step is to build a mathematical model based on the received formulation of the problem and requirements. During this step, the following questions need to be answered:

- For identification of which parameters is the model built?
- What is the goal of the operation?
- Which constraints have to be applied to the variables to mitigate the conditions of the modeled domain?

To describe a resource provisioning problem as a mathematical model, a set of formal relations, equations and domain definitions need to be specified. They link the controlled and uncontrolled parameters and describe the domain, environment,

applications, and their interactions as well as efficiency criteria. These equations contain a goal function for resource provisioning and multiple constraints. A goal function is a representation of the efficiency criteria, i.e., business or technical objectives like cost, time, or profit. Constraints are mathematical expressions of limitations on different aspects of the considered environment, applications, behavior, runtime operation, filters and relevant assumptions. These constraints cannot be changed and are imposed from previous knowledge or domain definitions. In this dissertation, resource provisioning models are deterministic.

Methods The third step is choosing a mathematical method to solve the defined model. In operations research, there is no single unified method of solving all mathematical models that appear in practice. The choice of the mathematical method is to be determined by the type and complexity of the formalized mathematical model. Some mathematical models can be so complex that it is impossible to find any solution with the available optimization methods. In this case, the researchers use imitation or heuristic methods [127].

Solutions The next step is finding solutions of the specified problem. In practice, it is impossible to solve a mathematical model in a so-called closed form, i.e., when a formula for calculation of optimal variables is derived. Mathematical methods of solving models consider algorithms that are iterative. This means that the problem is solved consequentially: with each iteration, a solution is received that gradually improves and eventually becomes an optimal solution.

Performance The fifth step is checking model performance, i.e., how adequate is the evaluated model [110]. This step is finished by applying the received results in practice.

2.5.5 Integer Linear Programming

Problem Types Practical decision-marking problems in resource provisioning belong to a certain class of Integer Linear Programming (ILP) problems or also boolean linear programming problems. These problems are characterized by certain conditions when fraction values of variables make no sense, i.e., problems with indivisibility. The solution elements can have either 0 or 1 values, i.e., the values of decision variables. Indivisibility problems can be knapsack problems or assignment (placement) problems [139]. In the following, we describe these problems, which are applicable in resource provisioning in fog computing and where indivisible services need to be assigned onto fog resources.

Knapsack Problem The knapsack problem can be described as follows [134]: A traveler is packing for a hike and wants to pack a certain number of items from 1 to n. The weight p_j and volume v_j for an item j is known, $j \in \{1..n\}$. The overall weight of the knapsack must not exceed P, and the overall volume must not exceed V. Based on subjective estimations of usefulness of items, each item has a coefficient of being useful in the trip c_j. The goal is to identify which items and how many should be taken in order to maximize the usefulness of the knapsack. The formulation

of the knapsack problem is given in equations (2.1)–(2.4).

$$\max \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

$$\sum_{j=1}^{n} p_j x_j \leq P \tag{2.2}$$

$$\sum_{j=1}^{n} v_j x_j \leq V \tag{2.3}$$

$$\forall x_j \geq 0 \tag{2.4}$$

A more specific definition of the knapsack problem is when items can be taken only as a whole, i.e., the item can be either put into the knapsack or not. In this case one more domain definition has to be added as $\forall x_j \in \{0, 1\}$.

Assignment Problem

In the assignment problem, there are n working places R_i and m workers W_j, $i, j \in \{1..n\}$. For each assignment $W_i \rightarrow R_j$ the cost c_{ij} of performing work is known. The goal is to distribute m workers on n places in a way each worker performs only one job, each job is performed by only one worker, and the sum of costs are minimal. The formulation of the assignment problem is given in equations (2.5)–(2.8).

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} \tag{2.5}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \ , \forall i \in \{1..n\} \tag{2.6}$$

$$\sum_{j=1}^{m} x_{ij} = 1 \ , \forall j \in \{1..m\} \tag{2.7}$$

$$\forall x_{ij} = \begin{cases} 0, \text{ if } W_j \text{ is assigned to } R_i \\ 1, \text{ if } W_j \text{ is not assigned to } R_i \end{cases} \tag{2.8}$$

Complexity

The resource provisioning problem has been shown to be NP-hard [7]. For this problem, an analogy towards the multiple knapsack problem can be performed [10]: different fog resources are knapsacks, single services of IoT applications are items to be inserted into knapsacks, the weight of knapsacks corresponds to available resources of devices in the fog, such as CPU, RAM and storage resources, and the cost of the knapsack is given by the defined QoS parameters. For service placement, the objective function is to maximize the utilization of devices at the edge of the network while satisfying the QoS requirements of applications, namely, satisfying deadlines on application deployment and execution time. With the assignment problem, the analogy stands with the constraints that services need

to be assigned only once to a certain resource, i.e., knapsacks. The complexity of a multiple knapsack problem is proven to be $O(n^2 + nm)$ [49], where n is the number of services to be placed and m is the number of fog resources available.

2.5.6 Solving Resource Provisioning Problems

Optimization

The mathematical methods of solving discrete programming problems can be divided into two main categories: exact mathematical methods and approximation methods. The exact mathematical methods include methods of the implicit enumeration algorithm [84], the branch and bound algorithm [137], and the dynamic programming algorithm [139].

Approximation

Approximation methods can be split into several groups [131, 151]:

- Greedy algorithms,
- Random search,
- Local search,
- Heuristics, and
- Meta-heuristics.

The random search algorithm selects a solution randomly from a pool of feasible solutions. Local search uses a common search space and chooses each element of the solution based on neighborhood relationships. The greedy algorithm composes a solution element by element by choosing the next element being the most obvious choice and eliminates considering already used elements or resources. Heuristic algorithms, meta-heuristics and their combinations are also very popular to find solutions for efficient resource usage and performance [79]. Heuristic algorithms contain a set of rules and constraints that can be based on certain rational and reasonable assumptions in the problem domain. These rules and constraints are rigorously constructed to receive a feasible solution within certain time limits. Meta-heuristic algorithms are general-purpose methods that can be applied onto any models, for example, a GA, swarm intelligence algorithm, and ant-colony optimization are meta-heuristic algorithms. These algorithms have a large space of solutions and require different resolution times, however are proven to provide efficient solutions. The combination of heuristics and meta-heuristics, i.e., hybrid algorithms, brings improvement in computational time.

Solvers

Linear programming has been popularized in many industries to find the optimal solution for any domain problem. Apart from the Excel Basic Solver Add-In, there are open source tools such as OpenSolver[9], SolverStudio[10], PuLP[11], Pyomo[12],

[9] Online; Accessed: Apr. 2023 https://opensolver.org
[10] Online; Accessed: Apr. 2023 https://solverstudio.org
[11] Online; Accessed: Apr. 2023 https://coin-or.github.io/pulp/index.html
[12] Online; Accessed: Apr. 2023 https://pyomo.readthedocs.io/en/stable/index.html

Java ILP [13], or lpsolve[14], and commercial tools such as Gurobi[15], MOSEK[16], AMPL[17], FrontlineSystem's Analytic Solver[18], or IBM Ilog CPLEX Optimization Studio[19]. These tools are meant to be used to produce exact mathematical solutions. For approximation methods, it is important to understand how to represent the considered problem. Such algorithms use existing open source or proprietary libraries. However, it order to apply these libraries it is needed to represent the model within the frameworks of different algorithms, which is a challenging task. In this dissertation, in order to solve different resource provisioning models, exact optimization methods, heuristic and meta-heuristic methods are implemented.

Limitations

A challenge in resource provisioning in fog computing is that software needs to be executed on different devices in the fog landscape, meaning that both internal management services and applications submitted for execution have to be adapted or reimplemented according to the processor architecture of the hardware. For example, popular devices used in research of fog computing are single-board computers, like the Raspberry Pi [83]. Raspberry Pis are built on the foundation of the ARM processor architecture. The ARM architecture is used in mobile phones, smart phones, and smart TVs. It is implemented in nearly 60% of all mobile devices [160]. Therefore, the is conducted under the assumption of using Raspberry Pis as representative devices, however, in this case the volatility of a fog landscape need to be setup within experiments. Open source and proprietary solvers which would be useful in resource provisioning exact optimization methods, for example, IBM CPLEX solver, Java ILP, or Gurobi, do not provide library distributions runnable on devices with the ARM processor architecture so far. This problem is not specific to this thesis, but is a common problem when using fog infrastructure. Therefore, other methods apart from exact mathematical methods of solving resource provisioning need to be investigated.

2.6 Identified Requirements

To sum up the foundations and concepts presented in this chapter, we can derive the following requirements that need to be fulfilled in order to realize fog computing. We divide these requirements into three main directions dealing with:

- Fog landscape,
- Resource provisioning, and

[13] Online; Accessed: Apr. 2023 https://sourceforge.net/projects/javailp/
[14] Online; Accessed: Apr. 2023 http://lpsolve.sourceforge.net/5.5
[15] Online; Accessed: Apr. 2023 https://www.gurobi.com
[16] Online; Accessed: Apr. 2023 https://www.mosek.com/products/mosek
[17] Online; Accessed: Apr. 2023 https://ampl.com
[18] Online; Accessed: Apr. 2023 https://www.solver.com
[19] Online; Accessed: Apr. 2023 https://www.ibm.com/products/ilog-cplex-optimization-studio

– Fog computing frameworks.

Fog Landscape Requirements A fog landscape is a representation of a cloud-to-thing continuum in a certain domain enabling fog computing. In order to realize a fog landscape, it is necessary to abstract and model its functionalities. In the following, we specify requirements that need to be fulfilled in order to create a fog landscape:

1. To realize fog computing, entities of a fog landscape and their functionalities should be defined and abstracted, and a fog computing architecture should be created.
2. A fog landscape should establish coordinated control over available cloud and edge resources. It should exploit ubiquitous networked computing devices that are close to the edge, as well as complement these devices with more powerful cloud resources, thus establishing a cloud-to-thing continuum.
3. To provide such a coordinated control, a fog landscape should have means to virtualize, instantiate, and utilize those ubiquitous heterogeneous resources at the edge of the network and in the cloud.
4. Those resources of a fog landscape should have different purposes according to their capabilities. Some resources, i.e., which are close to the edge of the network and are less powerful, should be able to execute certain services, e.g., sensing and actuating services, depending on the underlying IoT devices and their capabilities. Other more powerful resources should in addition to the execution of certain services manage those less powerful resources.
5. A fog landscape should provide means to orchestrate the available resource pool in an efficient manner with the purpose to execute arbitrary IoT applications. The cloud should complement resources at the edge of the network in the case when the latter are not enough for execution of needed workloads.

Resource Provisioning Requirements A fog landscape needs to have means to select and allocate its available resources and to perform service placement and execution. For that, resource provisioning need to be enabled. Therefore, we derive the following resource provisioning requirements in a fog landscape:

6. A fog landscape should provide resource provisioning in order to efficiently distribute workload among resources in the fog.
7. A resource model in a fog landscape should represent all available resources at the edge of the network and in the cloud and their capabilities and relationships.
8. A distributed IoT application to be executed in the fog should also be modeled.
9. The resource model and the IoT application model should be used for the definition of resource provisioning in the fog, i.e., of a service placement

problem.

10. The resource provisioning should adhere to the desired goals and account for fog landscape constrains.

Framework Requirements The specification of a fog landscape and resource provisioning provides theoretical foundations for realizing fog computing. To apply this in the real world, it is necessary to create a fog computing framework that abstracts functionalities of the fog and provides fog computing-specific software. It is one of the goals of this dissertation that this framework may be selectively changed by additional modules in the future, thus providing a tool for researchers to continuously work on resource provisioning approaches in an arbitrary fog landscape. In the following, we derive the requirements for realizing a fog computing framework for efficient IoT application execution and for implementing a real-world testbed by the means of this framework.

11. A fog computing framework should be able to create and maintain a fog landscape made up from computational resources at the edge of the network and in the cloud.
12. The framework should establish communication and interactions in the fog.
13. It should efficiently perform service delivery, deployment and execution of IoT applications.
14. For a resource- and QoS-efficient execution of the applications, resource provisioning should be implemented.
15. IoT application execution may be requested at any time. The framework should accordingly react to the workload and trigger resource provisioning. The available resource pool should be shared and optimized among submitted IoT applications.
16. The IoT application execution as well as the resource pool should be monitored at runtime. The monitoring should provide necessary data for the evaluation of resource provisioning constraints.
17. The framework should dynamically react to the volatility of the fog. Corresponding countermeasures should be performed to tackle the volatility, i.e., when devices lose connection or get overloaded, as well as when new resources appear in the fog.
18. The framework should provide means to overview execution results and calculate different performance metrics both of the framework runtime and resource provisioning, e.g., deployment and execution time, number of successfully executed applications, failure rate, QoS of single applications, or utilization of resources.

The identified requirements provide context for the contributions of this dissertation. In the following chapters, we discuss how modeling of a fog landscape, resource provisioning, framework architecture and its implementation in FogFrame reflect these requirements.

CHAPTER 3

Resource Provisioning for IoT Services in the Fog

In this chapter, we introduce a fog landscape model aiming to exploit ubiquitous networked computing devices that are close to the edge of the network with the purpose to execute arbitrary IoT services. We consider how to abstract and use the resources offered by IoT devices, as well as how to allocate those available resources to execute arbitrary IoT services. These findings become the foundation for a conceptual fog computing architecture.

Based on this conceptual architecture, we are able to orchestrate resources at the edge and to provide a suitable resource provisioning approach, i.e., a solution on how to distribute workload and data among resources in the fog. For this, we formalize a system model which aims to minimize delays arising from the transfer times between the fog and the cloud, and to maximize resource utilization of existing fog resources. For evaluation of the proposed system model, we apply different scenarios and resource provisioning policies within a simulated fog. The goal of the evaluation is to identify the best provisioning policy by comparing suitable metrics, i.e., round-trip time, delay, makespan, and cost.

It has to be noted that this part of this thesis has been developed and published in early 2016. At that time, research on different workload allocation approaches for the fog was rather limited and this work was one of the first ones. Later, this work has been extended in manifold ways to address different aspects of fog computing. In subsequent chapters of this thesis, the concept and content of this paper is extended.

3.1 Overview

The fog mirrors the basic structure of the IoT, where a multitude of heterogeneous, networked devices cooperate [12, 45] as has already been discussed in Section 2.2. Potential use cases for fog computing include typical IoT scenarios, e.g., data prefiltering in Big Data scenarios [40] or preprocessing of data streams from sensor nodes [74]. Single IoT devices coordinating a group of other IoT devices, located close to the edge of the network and providing virtualized resources, are called *fog cells*. These cells allow executing IoT services close to the data sources or sinks, instead of involving the cloud. This leads to decreased delays, as well as a better utilization of already available computational, storage, and networking resources in the fog.

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 how to distribute IoT services on available fog resources. To answer this question, we apply the concept of a *fog colony*. 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, task requests and data can be distributed and shared between the single cells. The operational purpose of fog colonies is the cooperative execution of arbitrary IoT services. Thus, fog colonies facilitate to move from centralized cloud-based data processing to a decentralized processing network that includes networked IoT devices, cloud offloading and multi-cloud deployment.

There has been some work on fog computing architectures, which provides foundations and concepts into understanding functional and technical requirements towards implementation of fog computing.

In their seminal conceptual work on the topic, Bonomi et al. [25] introduce a layered model bridging the IoT and the cloud. The authors show that applications might be placed in the cloud and in the fog, spanning potentially different cloud providers. In addition, it is shown that a fog computing framework needs to enable the communication between the cloud and the fog, inside the fog, and between the fog and IoT devices. Dastjerdi et al. [45] present a RA for fog computing which follows a very similar structure if compared to the work by Bonomi et al. The RA implies serving IoT requests in the local fog rather than involving the cloud. In the RA, central fog services are placed in a software-defined resource management layer, which provides a cloud-based middleware. Notably, this prevents that fog colonies act 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 computational tasks from the edge to the cloud are achieved through the middleware. In another discussion of basic fog features, Vaquero et al. [141] consider different concepts to realize fog architectures, including both centralized and decentralized, i.e., Peer-to-Peer (P2P), approaches. Notably, the authors introduce the notion of edge clouds,

which are private fogs made up from IoT devices, resembling our notion of fog colonies.

The RAs need to take into account the concrete needs of fog resource provisioning. In the discussed RAs, the focus is on the communication and task sharing between the different layers, i.e., the cloud, the edge, and the IoT.

The number of resource provisioning mechanisms specifically aiming at fog computing was quite limited at the time when we started this research. Hong et al. [77] 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 [77]. Aazam and Huh [1, 2] present a more sophisticated resource provisioning mechanism, which is based on the prediction of resource demands. Dynamic allocation of resources is performed in advance during the design time of the system. This approach is based on cost optimization, and resource allocation depends on the probability fluctuations of the demand of the users, types of services, and pricing models. Cost function parameters are set up during the time the contract between the user and provider is negotiated. The approach also takes user incentives and encouragement mechanisms into account. In contrast, our work provides runtime resource provisioning, i.e., accounts for dynamic infrastructural changes in a fog colony.

Apart from fog-specific resource provisioning solutions, resource allocation and service scheduling are major research challenges in the general field of cloud computing [94, 130, 166]. While these approaches offer interesting insights, there are certain differences between fog services and cloud services. These prevent a direct adaptation for the use in the work at hand. First, the size and type of cloud resources is very different from its counterparts at the edge. While cloud resources are usually handled on the level of physical machines, VMs, or containers, fog resources are usually not as powerful and extensive [141]. 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 and cost in the fog [143]. This is especially important since one particular reason to use fog computing in IoT scenarios is the higher delay-sensitivity of fog-based computation [125]. Hence, resource provisioning approaches for the fog need to make sure that this benefit is not foiled by extensive data transfer times and cost.

Therefore, in this chapter, we introduce a conceptual fog computing framework. We apply the mentioned above concept of fog colonies. Based on this framework, we are able to orchestrate fog resources and to provide a suitable resource provisioning approach, i.e., a solution on how to distribute task requests and data in the fog.

The remainder of this chapter is organized as follows: In Section 3.2, we describe a fog landscape, and propose a conceptual architecture to enable resource provi-

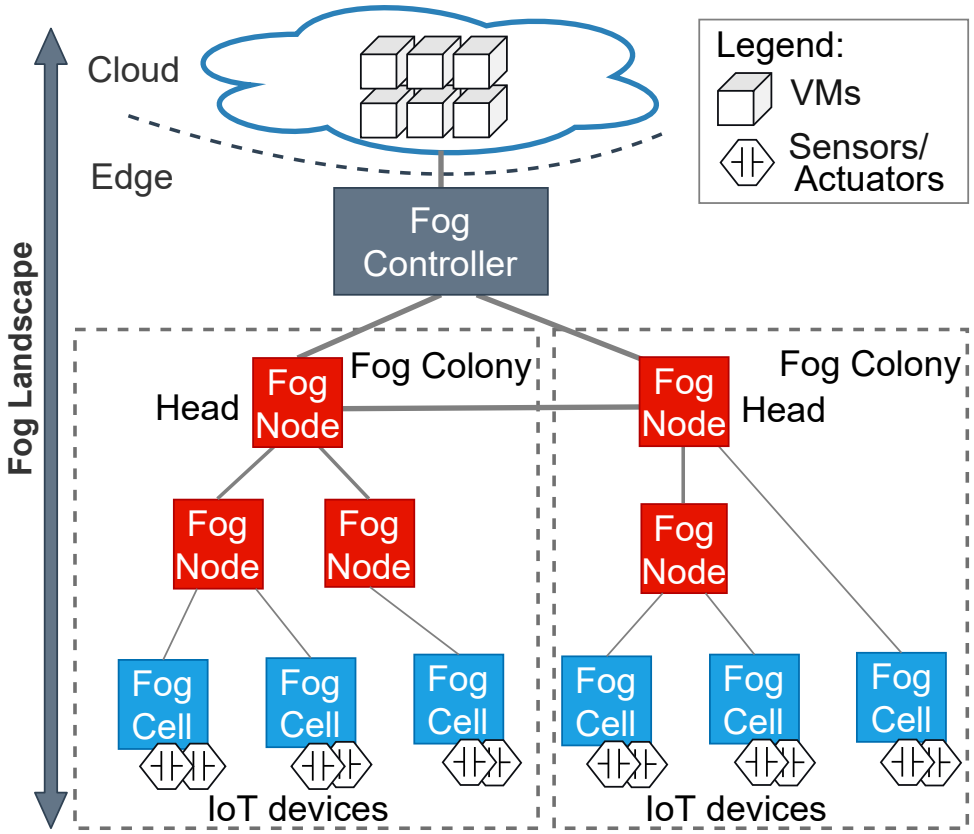

Figure 3.1: An overview of a fog landscape.

sioning. Next, in Section 3.3, we formalize a resource provisioning model with the goal of maximally utilizing resources at the edge in fog colonies, while also complementing fog colonies with the cloud resources. We evaluate the model in Section 3.4, demonstrating the efficiency of our resource provisioning approach regarding different QoS and cost metrics comparing to time- and space-shared provisioning policies and to the execution in the cloud.

3.2 Fog Computing Architecture

In this section, we present a conceptual fog landscape as depicted in Figure 3.1 and a fog computing architecture. The architecture enables the execution of IoT services in an arbitrary fog landscape. This architecture allows optimizing resource provisioning in the fog, as discussed in Section 3.3.

Resources Before describing the typical functionalities of fog computing, it is necessary to discuss general characteristics of a fog landscape. For this, we follow the notion of a fog landscape as a cloud-to-thing continuum [82]. A fog landscape consists of

a combined pool of computational and storage resources in the cloud and at the edge [45, 80]. The combined pool of resources enables support of decentralized processing of data on IoT devices in combination with the benefits of cloud technologies and virtualization [25, 111]. The organization of the fog can be both, flat or hierarchical [86]. In this dissertation, a hierarchical structure of the fog landscape is considered (see Figure 3.1). Different resources of the fog landscape are organized as layers taking into account different computational capabilities of devices and different workloads on those devices. For example, computations to perform resource provisioning and service placement are resource-intensive, and need to involve different computational and memory capabilities compared to a measurement service sending measurements from a sensor of an IoT device.

In the following, we use the notion of *task requests* for computational duties which need to be accomplished using cloud or fog resources. The actual software instances executing these task requests are called *services*. Possible examples of such services include stream processing, MapReduce applications, or distributed data storage.

3.2.1 Fog Landscape

Components

Following the basic structure of fog computing as presented in [25, 45], we consider resource provisioning and orchestration in both the cloud and fog. To achieve this, a *fog controller* is introduced, which controls fog cells. As discussed above, IoT applications should also be executable without any involvement of the cloud, as the cloud complements fog colonies. Hence, another level of control is necessary, which needs to run exclusively at the edge of the network. For this, we introduce *fog nodes*, which are a specific kind of fog cells. A fog node manages a number of fog cells or other fog nodes connected to it. A fog node is itself an extended fog cell, which has got the capabilities to not only host services, but to perform management activities, such as service placement and deployment. As it has been already mentioned in Section 3.1, we call such structures fog colonies. In our architecture, we support a hierarchy of fog colonies which are managed by a fog controller. The further layers of the hierarchy are the fog nodes, the fog cells, and finally the IoT devices at the very bottom of the hierarchy (see Figure 3.1). Fog cells and fog nodes are two specific types of *fog devices*.

Fog Colony As it can be seen in Figure 3.1, a hierarchy of fog devices forms a fog colony:

- Sensors and actuators attached to fog cells and
- Fog cells connected to a fog node, which becomes a parent to those fog cells.

Having a hierarchical structure allows establishing a coordinated control over application deployments in a fog colony, for example, a fog colony may become a domain-specific execution environment or it can cover a certain location.

Therefore, fog colonies may be considered as micro data centers made up from an arbitrary number of fog cells as we have briefly mentioned in Section 3.1. As in a cloud data center, within a fog colony, workloads and data can be distributed and shared between the single fog cells. The operational purpose of fog colonies is the cooperative execution of arbitrary IoT services. Thus, fog colonies facilitate to move from centralized data processing in the cloud to a network of IoT devices for decentralized processing and complemented with cloud offloading. In each fog colony, there is exactly one head fog node that performs service placement. Other fog nodes can be present in a fog colony. They perform computations in the same manner as fog cells, and as well are responsible for the communication and data flow between the connected fog cells and other fog nodes.

Fog Controller Since a fog landscape consists of computational resources both from the cloud and at the edge of the network, it is necessary to mediate between fog colonies and the cloud. In the fog computing architecture, a *fog controller* is forseen for this. The fog controller establishes communication between fog colonies and the cloud, and within and between fog colonies. The head fog nodes of fog colonies communicate with the fog controller in the case when additional cloud resources are needed for application execution. However, the latter may also act autonomously if the fog controller is not available.

Neighbor Colonies Fog colonies do not only interact with the cloud. The colonies also need to interact with each other in order to propagate (i.e., delegate) the execution of applications from one fog colony to another. For instance, if one fog colony does not have enough computational resources to execute an application, then it may delegate the corresponding application to a neighbor fog colony. In order to enable the delegation of the application execution, fog colonies are connected to each other via their corresponding head fog nodes.

3.2.2 Fog Controller

Purpose The fog controller is the central unit that manages the execution of task requests in the cloud, and supports the underlying fog landscape. The fog controller performs cloud resource provisioning for task requests that are not delay-sensitive or cannot be executed in the fog, e.g., very resource-intensive Big Data analysis task requests. If necessary, the fog controller performs the global optimization of underlying fog colonies by restructuring them. For this, the fog controller 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 could be used to deploy fog cells, or to recover after faults or damage of fog cells. Importantly, the fog controller can overrule fog nodes in fog colonies, but the latter may also act autonomously in the case that no fog controller is available.

3.2.3 Fog Cells

Purpose

Fog cells are software components running on fog devices. Fog cells serve as access points to control and monitor connected IoT devices, e.g., sensor nodes. In order to do so, the fog cells are able to receive task requests, perform data analysis, allocate their own computational resources if available, or propagate task requests to upper layers of the fog colony hierarchy, i.e., to the fog nodes (as discussed below). Cells may interact with an arbitrary number of IoT devices, however, in practice, the number of devices to be controlled by a fog cell is limited by the cell's computational resources.

Components

Each fog cell consists of the following components (see Figure 3.2): The *listener* receives task requests from other fog cells or IoT devices. The *monitor* observes service executions in the compute unit. The *database* stores data about received task requests, the current system state of the fog cell, i.e., available computational and storage resources, and monitoring data. The *fog action control* performs actions according to the provisioning plan produced by the fog node, e.g., to deploy and start a particular service (see Section 3.2.4). Fog cells access the distributed *storage* to share data, e.g., service implementations, in a colony. This component enables faster data access compared to data storage in the cloud. The *compute unit* provides the actual computational resources for the deployment and execution of services.

APIs

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

3.2.4 Fog Nodes

Purpose

A fog node's main task is to support a fog colony by orchestrating the involved fog cells. Each fog colony features exactly one head fog node. The fog node is itself a (powerful) fog cell, which manages the resources offered by subordinated fog cells and performs resource provisioning to execute task requests. Also, the control node is able to propagate task requests to the fog controller or to other fog colonies (via their fog nodes), if task requests cannot be handled by the current fog colony. For this, a resource management mechanism for vertical scalability is necessary to identify how task requests can be delegated in the entire fog landscape. Apart from resource provisioning, fog nodes

- Perform infrastructural changes in the fog colony,
- Analyze resource utilization within the colony,
- Create a provisioning plan to allocate resources for task requests, and

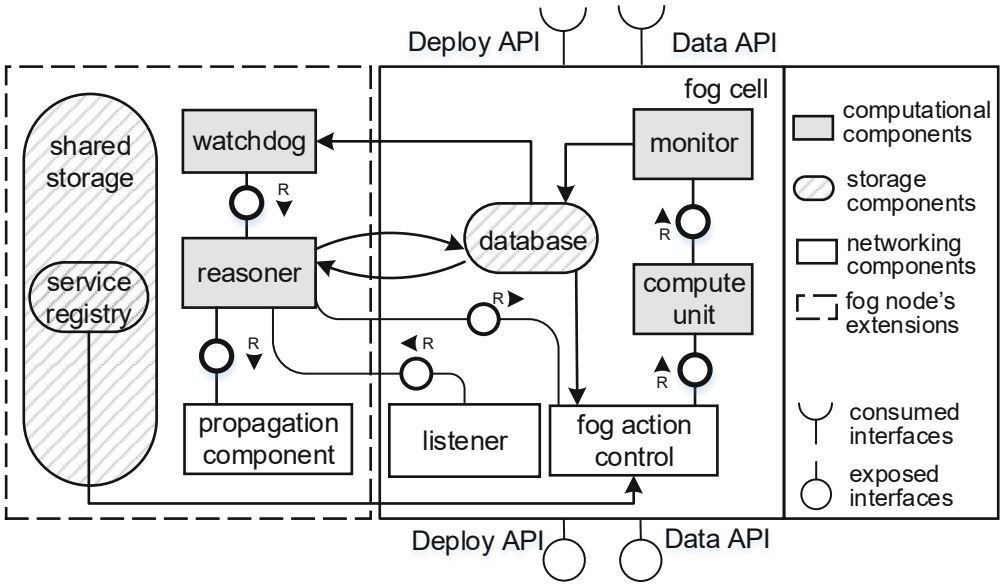

Figure 3.2: Architecture of a fog cell and a fog node

– Monitor IoT devices and fog cells.

An approach to optimize task request distribution and resource allocation is described in Section 3.3.

Components As previously mentioned, fog nodes are extended fog cells. On the left-hand side of Figure 3.2, the extensions needed for fog nodes are depicted. The *reasoner* component produces a provisioning plan for the colony's resources to execute received task requests. The provisioning plan determines which services should be used to fulfill task requests, and where these services should be deployed, i.e., what fog cell to use. The reasoner also gets the information about the system state, i.e., available fog colony resources, and controls the connected fog cells, i.e., plans infrastructural changes in the fog colony, if necessary. The *database* additionally stores the resource provisioning plan produced by the reasoner. After the reasoner produces a provisioning plan, the *fog action control* performs the orchestration of fog cells according to the plan. If there are no sufficient resources in the considered fog cell, or further processing is needed, such task requests are separated and delegated to the other fog colonies by the *propagation component* via the fog node. The *watchdog* component features means to receive up-to-date information about the utilization of the connected fog cells. It observes monitoring data in the database and compares it to the expected QoS level, i.e., measures the consumption of the cell's computational resources as well as QoS parameters, e.g., the execution time. This information influences the decision-making in the reasoner component. The *service registry* component hosts service implementations and enables the fog action control to search for services and to

deploy them on the fog cell compute units. As storing service implementations is resource-consuming, the service registry is located in the storage unit of the control node.

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 head fog 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 to maintain a fog colony, but decentralized regarding the execution of applications in this work. However, we still foresee that another fog node becomes the head fog node in a fog colony, if necessary, e.g., in case the original head fog node fails.

3.3 Resource Provisioning in the Fog

As discussed in Section 3.2, it is necessary to provide the fog node as well as the fog controller with means to allocate resources for service requests. For this, the fog node needs a complete overview of the system state of a fog colony. With this system state as input, the reasoner is able to compute a service placement plan and to schedule service requests onto services running in fog cells. As stated above, we assume that a fog colony is autonomous, i.e., the fog controller is only involved if a fog colony needs additional cloud resources.

Goals

As in the field of cloud resource optimization, manifold goals for fog resource provisioning are possible, e.g., time, cost, or energy efficiency optimization [1, 111, 141]. In the following, the goal of the presented resource provisioning approach is the optimization with regard to the utilization of fog cells and the minimization of delays. First, this means that the computational resources offered by fog cells should be utilized as much as possible, because using cloud resources leads to higher overall cost. Second, data needed and sourced within a particular fog colony should be handled by that particular fog colony, if possible. This is done in order to avoid an increase of delays by propagating data to the cloud or to other fog colonies. Together, these goals form the foundation for a multiple-criteria optimization problem, as is presented in the upcoming sections. The system model is formalized, and its goal is to provide delay-sensitive utilization of available computational resources in the fog. This model is implemented and then evaluated to show the benefits of this contribution.

3.3.1 System Model

Entities

The basic entity for optimization in our system model is a fog colony with a head fog node F. F has n subordinate fog cells f^j, i.e., fog cells are part of the same fog colony and controlled by F: $\forall f^j \in F^C$, $j \in \{1..n\}$. The CPU utilization of the fog node is indicated by U^F, for a fog cell by U^j. Analogously, for RAM

utilization, the parameters M^F and M^j are used. There are different types of fog cells (T^F for fog nodes, T^j for fog cells), which indicate the sets of services that can be run on this particular cell, i.e., indicate different IoT devices. The bandwidth and link delay between the fog node and a particular fog cell j are indicated by b^j and d^j.

Task Requests The fog colony controlled by F receives m task requests. Each request r_i, $\forall r_i \in R$, $i \in \{1..m\}$ is characterized by two time points: t_s and t_f, where t_s is the time when a task request enters the system and t_f is the time when the task request is fulfilled. Each r_i is characterized by its CPU μ_i^{CPU} and RAM μ_i^{RAM} demand, and the type of service μ_i^{TYPE} needed to fulfill the task request.

System State The current system state is indicated by the period τ and its start time t^τ. The head fog node produces the service placement plan S for the period $\tau + 1$ with the start time $t^{\tau+1}$ calculated by adding a certain time ϵ corresponding to the desired time between two consequent runs of resource provisioning: $t^{\tau+1} = t^\tau + \epsilon$.

Cardinality We define the cardinality function, i.e., length of a service placement plan S, which consists of a sequence of assignments for fog cells $S(F^C)$, for the fog node itself $S(F)$, and for not assigned task requests $S(0)$, for all $i \in \{1..m\}$, i.e., $S = S(F^C) \bigcup S(F) \bigcup S(0)$.

Delegation We assume for a fog cell f^j the assigned task requests r_k^j, $\forall r_k^j \in R^j$, $k \in \{1..p^j\}$, for the fog node r_q^F, $\forall r_q^F \in R^F$, $q \in \left\{1..u^F\right\}$. If for some task requests there are no resources in the fog colony, or the task request needs further resources which cannot be offered by the colony itself, the service placement plan S is still produced, but unassigned task requests $R \setminus (\bigcup_{j=1}^{n} \bigcup_{k=1}^{p^j} (r_k^j)) \setminus \bigcup_{q=1}^{u^F} (r_q^F)$ are propagated to a higher layer in the fog colony. This higher layer could either be another fog node (in the case when several fog nodes are present in the fog colony) or the fog controller.

Objective Function The objective function (3.1) aims to maximize the number of possible assignments while decreasing the propagation of task requests to the higher layer as much as possible.

$$|S(F^C) \bigcup S(F)| \rightarrow max \tag{3.1}$$

Variables The binary variable x_i^j decides whether a task request r_i is assigned to a fog cell f^j as defined in (3.2). For fog nodes, x_i^F is defined analogously.

$$x_i^j = \begin{cases} 1, & \text{if } r_i \text{ is assigned to} f^j, \\ 0, & \text{if } r_i \text{ is } \textit{not} \text{ assigned to} f^j, \end{cases} \tag{3.2}$$

Based on this, the goal function can be rewritten as (3.3).

$$\sum_{i=1}^{m} \Big(\sum_{j=1}^{n} (x_i^j) + x_i^F \Big) \rightarrow max \tag{3.3}$$

Constraints on Request Type

Constraints As a first constraint, we assume that each fog cell of a specific type T^j can execute certain types of task requests, i.e., the type of a task request must be checked. If the task request is assigned to a fog cell, meaning $x_i^j = 1$, then the type of a task request must conform to the type of the fog cell, i.e., $\mu_i^{\text{TYPE}} \in T^j$. And, correspondingly, if the task request is assigned to the head fog node, i.e., $x_i^F = 1$, then $\mu_i^{\text{TYPE}} \in T^F$.

Constraint on Assignments

Second, each task request r_i can be assigned only to one specific fog cell f^j or F as defined in (3.4):

$$\sum_{j=1}^{n} (x_i^j) + x_i^F \leq 1 \ , \forall i \in \{1..m\} \tag{3.4}$$

Priority by Delays

Next, we choose a fog cell with minimum estimated delay between a fog cell and the fog node. For that, the set of all fog cells is sorted according to the value d_j. Therefore, the assignment of task requests is prioritized to fog cells with lesser delay.

Constraint on Resources

As defined in (3.5) and (3.6), assigned task requests must not exceed the resources of a fog cell. The equations consider an assumed percentage γ of system needs that must be free to maintain a corresponding fog cell. In (3.5) and (3.6), we multiply approximate CPU and RAM resources needed to execute task requests and decision variables corresponding to assignments of a task request to a certain resource and find their sums. The constraints limit the number of assignments, ensuring that the calculated sum of according CPU and RAM resources does not exceed the desired availability of CPU and RAM fog cells $\forall j \in \{1..n\}$.

$$\sum_{i=1}^{m} (\mu_i^{\text{CPU}} x_i^j) \leq U^j (1 - \frac{\gamma}{100}) \ , \forall j \in \{1..n\} \tag{3.5}$$

$$\sum_{i=1}^{m} (\mu_i^{\text{RAM}} x_i^j) \leq M^j (1 - \frac{\gamma}{100}) \ , \forall j \in \{1..n\} \tag{3.6}$$

3.4 Evaluation

In the following evaluation, we show the efficiency of our resource provisioning approach regarding the round-trip time per task requests, execution delays and makespans of tasks, as well as the cost of execution, compared to a baseline

approach and to the execution in the cloud. For this, we extend the well-known *CloudSim* modeling and simulation framework [34] with the means to simulate a fog landscape.

3.4.1 Evaluation Environment

CloudSim Extensions

In order to reflect the structure of a fog landscape, CloudSim needs to handle fog colony hierarchies as introduced in Section 3.2. For this, the original *Datacenter* class of CloudSim is extended by *FogDatacenter* that features specific methods of a fog colony, i.e., linking different fog cells to a fog node. CloudSim's *DatacenterBroker* class is extended by *FogBroker* that mirrors the behavior of fog nodes or the fog controller depending on the simulated environment. The FogBroker class implements the system model presented in Section 3.3.1. CloudSim's class responsible for task requests, *Cloudlet*, is extended by *FogCloudlet* to include the issuer parameter. This is necessary to track the origin of task requests. To calculate delays (see Section 3.4.3), the class *DelayEntity* is added. This class transforms data from CloudSim's class *NetworkTopology*, and is used for metric calculations. The main class of the simulation creates a fog landscape and the cloud environment, sets up needed services, generates task requests in fog cells, and, finally, runs the simulation. The discussed extensions are presented in Figure 3.3.

3.4.2 Experimental Setup

Fog Colony

As fog landscape in our evaluation, we consider a fog colony of 100 fog cells (each running on a separate IoT device) and a head fog node, which is the head of the fog colony. The fog colony is linked to a public cloud in the case that cloud resources are necessary for the fulfillment of task requests. Of these 100 fog cells, 10 are simultaneously issuing 1,000 task requests to the fog colony (more precisely: to the fog node) assuming these fog cells do not have enough own resources for processing the task request. This leaves 90 fog cells to provide computational resources to handle the 1,000 task requests. If these fog cells are not sufficient, cloud-based computational resources are used to fulfill the task requests. Each fog cell executes different services according to the available computational power. For the simulation, we do not restrict the service types to be hosted by the single fog cells, i.e., each cell is able to respond to every task request.

Simulation Settings

CloudSim calculates the execution time of a service needed to fulfill a particular task request by taking into account the number of instructions necessary to execute the task request and the number of instructions a compute unit is able to process per second. Within our evaluation, a task request needs 0.04 million of instructions per second (mips), and has 300 MB of incoming and 300 MB of outgoing data. Fog cells possess compute units which are able to handle 250 mips. For modeling the fog network capacities, i.e., the network topology, the

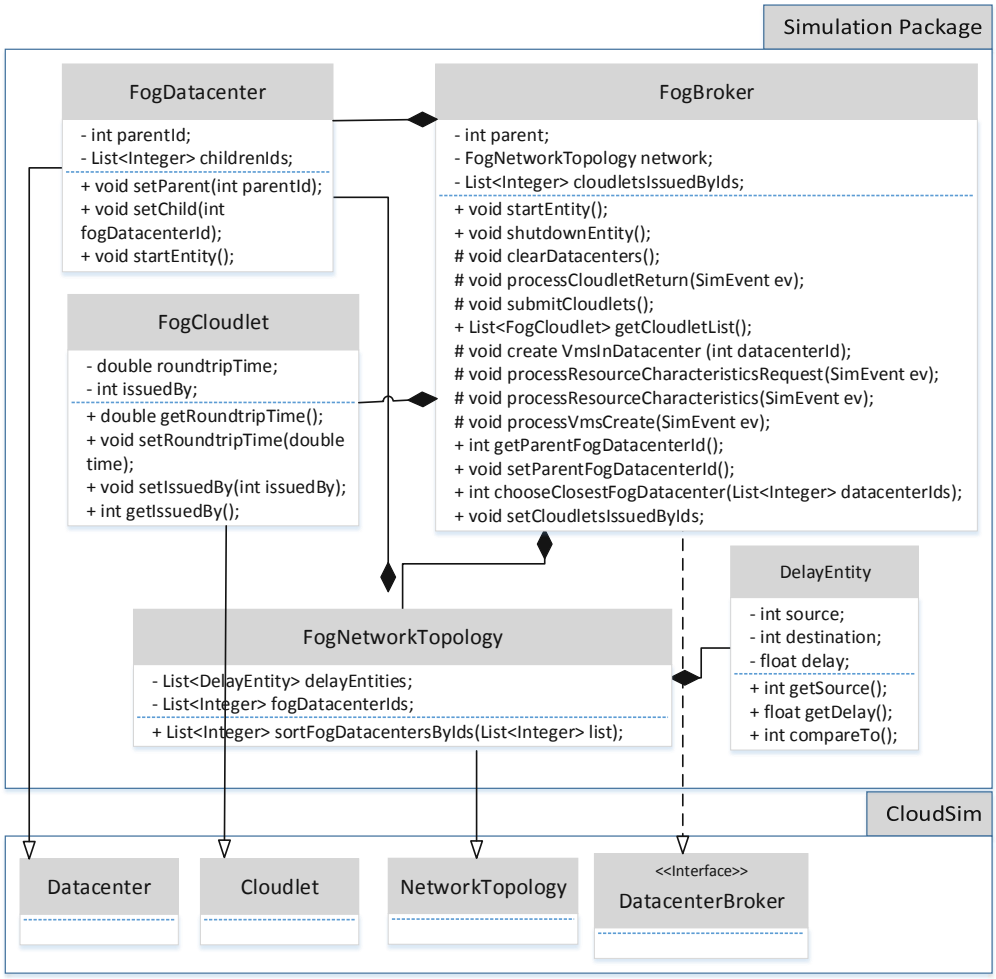

Figure 3.3: Extensions to CloudSim.

BRITE Internet topology generator [101] is used. The one-way delays between fog cells and the fog node are generated by BRITE according to the physical distance between generated nodes, and belong to the interval $(0, 1]$ seconds. The cloud-based computational resources are set to be twice as powerful as fog cells' compute units, i.e., VMs are able to handle 500 mips. The cost per processing in the cloud is set to $\$0.30$ per Billing Time Unit (BTU[1]), i.e., one hour. The delay between the fog node and the cloud is set to 2 seconds. The bandwidth between the fog node and fog cells is set to 6 Mbit/s, and between the fog controller and the fog node to 10 Mbit/s.

Scenarios

For evaluating the efficiency of the fog landscape, we apply the resource provi-

[1]BTUs were used in 2016 at the time of publication of this work. Currently, BTUs are usually not defined as full hours and vary for each service

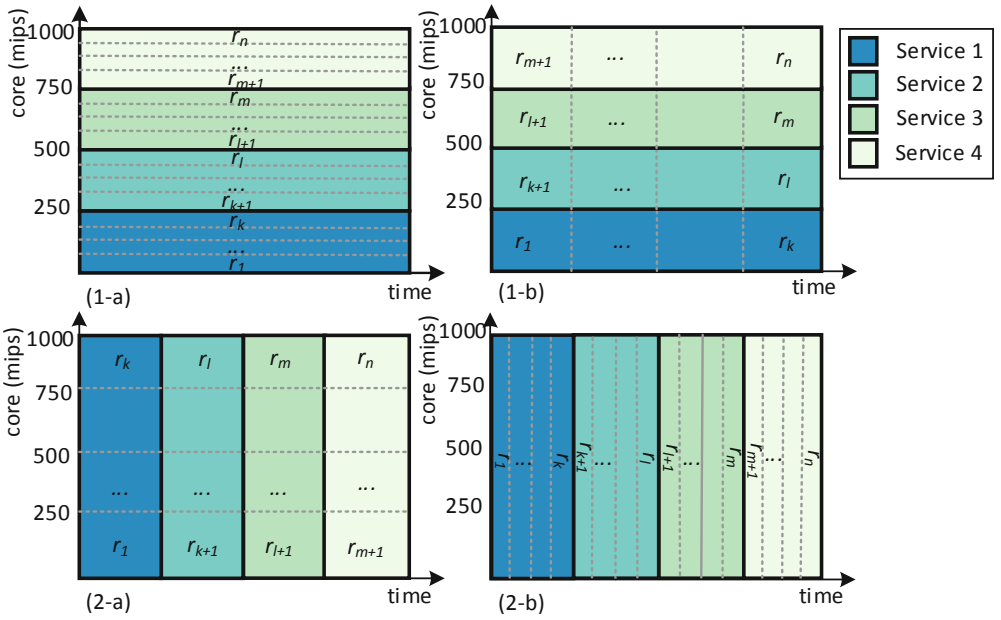

Figure 3.4: Provisioning policies.

sioning policies offered by CloudSim [34] (called the 'baseline' scenario) and our system model combined with those policies (called the 'optimization' scenario) in the fog landscape. Furthermore, we execute all task requests in the cloud via the fog controller for the case when the fog landscape is not available (called the 'cloud' scenario).

Provisioning Policies

Originally, CloudSim introduced *time-shared* and *space-shared* provisioning policies for both computational resources and services running on those resources. Time-shared provisioning means that the shared resource is simultaneously divided among assigned entities, i.e., the entities run in parallel. Space-shared provisioning means that the computational power of the resource is shared, and the resource is provisioned to an entity sequentially, i.e., the next entity starts running only when the previous entity is finished. Applied to fog computing as discussed in this thesis, we distinguish four basic policies, as depicted in Figure 3.4: (1-a) time-shared provisioning of fog cells for services and time-shared provisioning inside services for task requests; (1-b) time-shared provisioning of fog cells for services and space-shared provisioning inside services for task requests; (2-a) space-shared provisioning of fog cells for services and time-shared provisioning inside services for task requests; (2-b) space-shared provisioning of fog cells for services and space-shared provisioning inside services for task requests.

3.4.3 Metrics

Chosen Metrics

To assess the efficiency of the fog landscape in the baseline, optimization, and cloud scenarios, we calculate the average round-trip time per each task request (3.7)–(3.8), total delays for the execution of all task requests (3.10)–(3.11), and the total makespan, i.e., the time between the entering of the first task request into the fog landscape and the fulfillment of all task requests. Additionally, for the execution in the cloud we calculate the total cost (3.12).

Round-trip Time The round-trip time t_i is the time a task request spends in the fog landscape from the moment of issuing until getting back the results of the processing. In Table 3.1, we show the average round-trip time per task request along with the standard deviation of the results. The round-trip time of each task request is calculated as the sum of the duration of uploading data to the fog node's host and to the destination fog cell's host for execution along with delays, execution time, and downloading data back to the issuer of the task request:

$$\begin{aligned} t_i = & t_{up}^{\{src_{r_i}, F\}} + d^{\{src_{r_i}, F\}} + t_{up}^{\{F, dest_{r_i}\}} + d^{\{F, dest_{r_i}\}} + \\ & t_{exec}^{i} + t_{down}^{\{dest_{r_i}, F\}} + d^{\{dest_{r_i}, F\}} + t_{down}^{\{F, src_{r_i}\}} + d^{\{F, src_{r_i}\}} \end{aligned} \tag{3.7}$$

Upload and Download Time Uploading and downloading times are calculated according to CloudSim's basic methods, i.e., dividing incoming and outgoing storage capacities of task requests by a corresponding bandwidth of the used network link:

$$t_{up}^{\{start,\ end\}} = \frac{size_{up}^{r_i}}{b^{\{start,\ end\}}} \tag{3.8}$$

$$t_{down}^{\{start,\ end\}} = \frac{size_{down}^{r_i}}{b^{\{start,\ end\}}} \tag{3.9}$$

Execution Time We assume that the execution time includes the actual task request execution time in a service, and processing times needed by the fog node and the destination fog cell.

Total Delay Taking into account the number of assigned task requests from (3.3), the delay $d^{\{src_{r_i}, F\}}$ between the issuer of the task request and F, the delay d^j between F and each fog cell, and the delay d^C between F and the cloud, we calculate the total delay in the fog:

$$\begin{aligned} delay_{fog} = & 2\sum_{i=1}^{m} d^{\{src_{r_i}, F\}} + 2\sum_{i=1}^{m}\sum_{j=1}^{n}(x_i^j d^j) + \\ & 2\Big(m - \sum_{i=1}^{m}\Big(\sum_{j=1}^{n}(x_i^j) + x_i^F\Big)\Big)d^C \end{aligned} \tag{3.10}$$

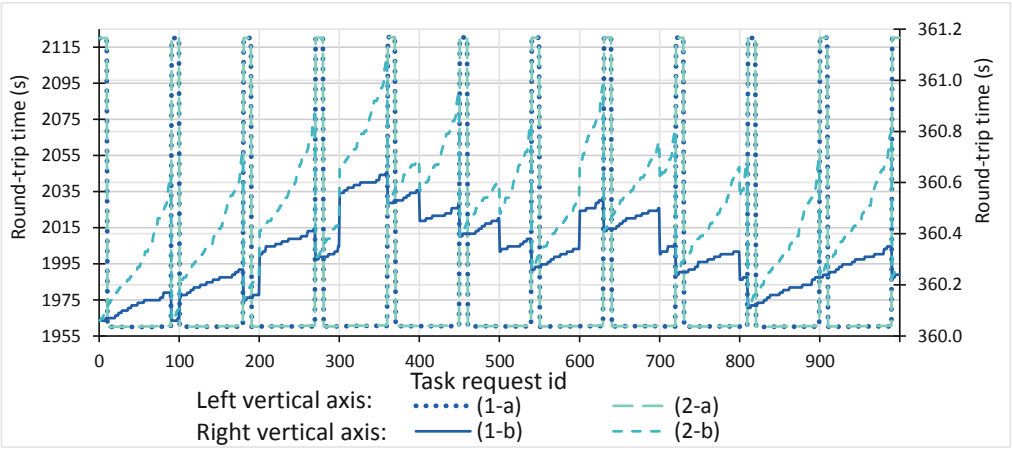

Figure 3.5: Round-trip times for task requests in the optimized fog landscape.

In the case of execution of task requests in the cloud, the delay to the destination equals to d^C for all m task requests, and the total delay equals to:

$$delay_{cloud} = 2\sum_{i=1}^{m} d^{\{src_{r_i},F\}} + 2md^C \tag{3.11}$$

Total Makespan In our experimental setup, the task requests are issued simultaneously, but have different start times of execution according to the policy applied. The total makespan is calculated as the maximum value of an array of summed values of start time of the execution with the round-trip time of the corresponding task request.

Cost We assume ownership of the fog and that these resources are available. Therefore, the fog cost can be neglected. The cost is calculated for the cloud execution according to (3.12), where c_{cloud} is the cost per processing second in the cloud and v is the amount of seconds in 1 BTU:

$$cost_{cloud} = \frac{\sum_{i=1}^{m} t^i_{exec} c_{cloud}}{v} \tag{3.12}$$

3.4.4 Results and Discussion

Round-trip Evaluation

Even though the provisioning inside a service for task requests has no influence on the delay minimization between fog cells (see Figure 3.4), it still influences the start and finish times of task request executions, and consequently affects the round-trip time per task request. The purpose of Figure 3.5 is to demonstrate visually which policy shows better round-trip times according to the optimization scenario. According to Figure 3.5, policy 1-b shows the least round-trip times for each task request in the simulated fog landscape, which is also supported by the results shown in Table 3.1.

Table 3.1: Average round-trip time metric comparison.

Policy	Average round-trip time (s)		
	Baseline	Optimization	Cloud
(1-a)	1979.69 σ=44.49	1979.48 σ=51.97	1056.04 σ=25.96
(1-b)	360.53 σ=0.20	360.32 σ=0.14	246,46 σ=1.35
(2-a)	1979.65 σ=51.99	1979.65 σ=51.92	1056.04 σ=25.96
(2-b)	360.50 σ=0.19	360.49 σ=0.20	246.46 σ=1.35

Table 3.2: Delay and makespan metrics comparison.

Policy	Baseline	Optimization	Cloud
	Total delay (s)		
(1-a)	530.82	323.68	6454.50
(1-b)	530.96	323.90	6455.00
(2-a)	492.78	490.63	6454.50
(2-b)	493.03	490.91	6455,00
	Total makespan (s)		
(1-a)	2123.07	2122.82	1129.05
(1-b)	2122.61	2122.54	1126.22
(2-a)	2129.96	2129.54	1129.05
(2-b)	2129.71	2129.28	1126.22

Evaluation of Delays

The default CloudSim time-shared VMs provisioning policy assigns VMs to data centers in the sequence these data centers are described in the network topology. When optimizing according to the system model, we compare the hosts of fog cells according to their link delays to the fog node, and prioritize fog cells with lesser delays. The conducted simulations confirm that the resource provisioning inside a service has no influence on the delays. Figure 3.6 shows the change in delays for policies 1-a and 1-b, and Figure 3.7 respectively for policies 2-a and 2-b. In the figures, the delays drop down because already allocated fog cells become free after fulfilling the assigned task requests, and afterwards other task requests are assigned to those fog cells due to the lesser link delay constraint.

Summary

As seen in Table 3.2, policies 1-a and 1-b decrease the total delay in the optimized fog landscape by 39% compared to the Baseline, and policy 1-b has also 82% less average round-trip time compared to policy 1-a. The standard deviation for the average round-trip time for the policies 1-b and 2-b is relatively small, because all task requests are executed sequentially in the services, and release

Table 3.3: Cost of cloud simulation.

Policy	Cloud Cost ($)
(1-a)	74.13
(1-b)	6.67
(2-a)	74.13
(2-b)	6.67

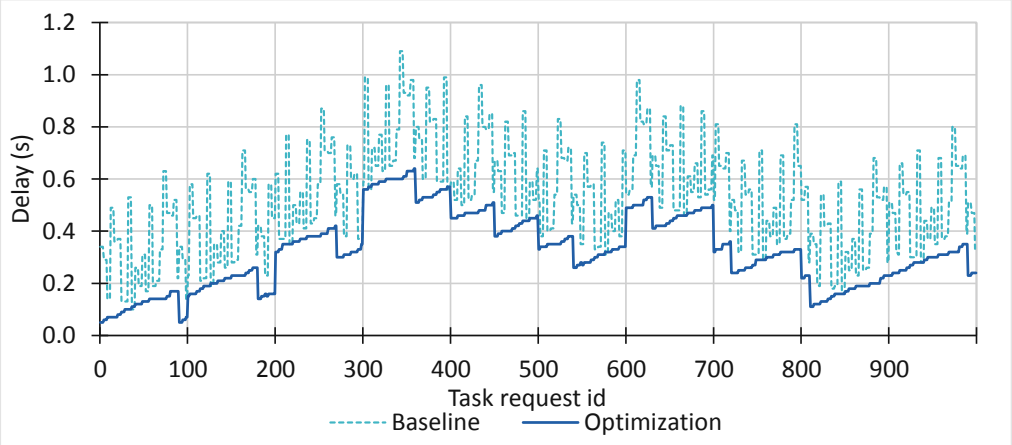

Figure 3.6: Baseline and optimization scenarios for the provisioning policies 1-a and 1-b.

resources immediately after the execution. Therefore, the round-trip time is affected more by the corresponding delays. In contrast, policies 1-a and 2-a execute task requests in parallel, which make the execution slower. The system model optimization along with policies 2-a and 2-b brings improvements, however they are marginal. The total makespan periods of all fog landscape scenarios are relatively the same, however policy 1-b shows better results as well because of reduced delays.

Cloud Scenario The cloud scenario shows smaller round-trip time and total makespan because of higher bandwidth, i.e., the data is uploaded quicker, compared to the fog landscape where the data is transferred between fog cells. However, the delay between the fog node and fog controller was set to the relatively high value of two seconds (see Section 3.4.2), and the execution in the cloud shows 95% higher delay compared to the optimization scenario in the fog. In reality, the link delay between the fog node and the cloud depends on the physical distance to the cloud. Additionally, the cloud scenario has cost for the execution (see Table 3.3). These results compensate the benefit from the smaller round-trip time and makespan.

Evaluation Outcome In total, the evaluation (see Table 3.2) demonstrates that the system model applied with the time-shared provisioning of fog cells for services along with

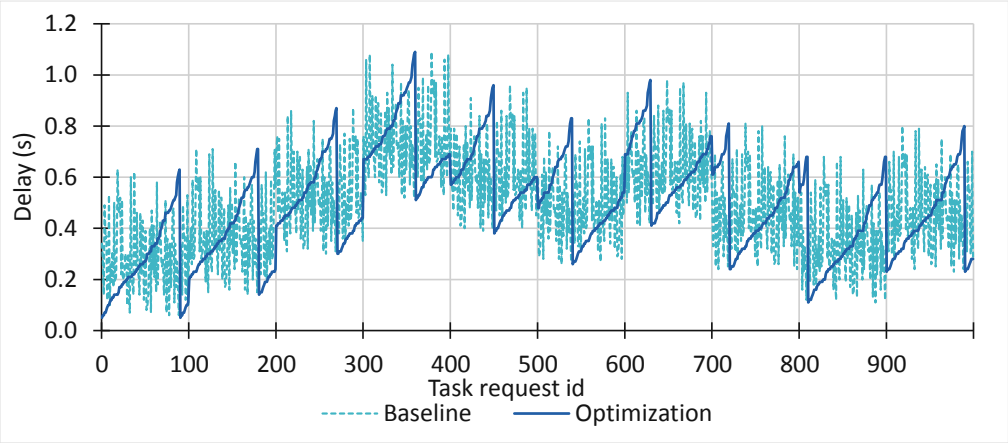

Figure 3.7: Baseline and optimization scenarios for the provisioning policies 2-a and 2-b.

space-shared provisioning inside services for task requests, i.e., the optimization scenario for policy (1-b), reduces delays in the fog landscape by 39%. It also shows less average round-trip time per each task request compared to the other policies. Lesser round-trip time is crucial if there are any time constraints for task requests executions, and especially in delay-sensitive fog scenarios, as discussed before.

3.5 Summary

The centralized processing of multiplicative IoT data in the cloud incurs high delays and accordingly low speed of data processing which are unfavorable for IoT applications and services. Fog computing promises to solve this problem by utilizing available computational, storage, and networking resources for the enactment of IoT services close to the edge of the network.

We have defined a fog computing architecture according to the specified requirements that aims to enact IoT services in an arbitrary fog landscape. A fog landscape is combined from computational and storage resource pools of cloud and edge resources. The main purpose of the architecture is to establish coordinated control over those resources and enable resource provisioning. A hierarchical structure of the architecture has been discussed together with its main entities: fog controller and fog colonies. Afterwards, the components of fog colonies that enable application execution, i.e., fog nodes and fog cells, have been introduced and discussed. This conceptual fog computing architecture becomes the foundation for the fog computing framework FogFrame, which is presented in Chapter 5.

In this chapter, we have introduced a system model for resource provisioning in a

fog landscape. To evaluate the efficiency of the proposed approach, we simulated the architecture. We showed that the optimization according to the system model combined with the time-shared provisioning of fog cells for services along with space-shared provisioning for task requests decreased delays by 39% and yielded shorter round-trip times and makespans.

This chapter constitutes Contribution I as defined in Section 1.2 and covers the first five requirements from Section 2.6:

1. To realize fog computing, entities of a fog landscape and their functionalities should be defined and abstracted, and a fog computing architecture should be created.
2. A fog landscape should establish coordinated control over available cloud and edge resources. It should exploit ubiquitous networked computing devices that are close to the edge, as well as complement these devices with more powerful cloud resources, thus establishing a cloud-to-thing continuum.
3. To provide such a coordinated control, a fog landscape should have means to virtualize, instantiate, and utilize those ubiquitous heterogeneous resources at the edge of the network and in the cloud.
4. Those resources of a fog landscape should have different purposes according to their capabilities. Some resources, i.e., which are close to the edge of the network and are less powerful, should be able to execute certain services, e.g., sensing and actuating services, depending on the underlying IoT devices and their capabilities. Other more powerful resources should in addition to the execution of certain services manage those less powerful resources.
5. A fog landscape should provide means to orchestrate the available resource pool in an efficient manner with the purpose to execute arbitrary IoT applications. The cloud should complement resources at the edge of the network in the case when the latter are not enough for execution of needed workloads.

CHAPTER 4

Towards QoS-aware Fog Service Placement

As discussed in Chapter 3, the head fog node of each fog colony has a reasoning component that aims to process submitted requests for IoT application executions and deploy single services of IoT applications onto suitable resources in the fog. So far, we have considered a rather simple resource provisioning optimization within the head fog node that is targeted to maximally utilize fog colonies. The constraints that have been considered are resource capacities of fog cells and fog nodes, assignment limits per services, and latency.

In this chapter, we advance with modeling the fog, we formalize more constraints on resources. We also focus on modeling IoT applications within the fog and their demands. These models provide foundation for the definition of the FSPP. Among manifold goals for resource provisioning, we focus on the goal function to minimize total QoS metric of application execution while maximizing the utilization of the fog. QoS metrics considered here are deadlines on the execution time of IoT applications.

The FSPP can be solved by a multitude of approaches that provide different optimal or near-optimal solutions. In this chapter, we consider several different approaches to solve the FSPP and compare their performance. Based on the solutions to the FSPP, the head fog node either instantiates services on particular fog cells, or delegates requests for specific service execution to the closest neighbor colony or to the cloud.

4.1 Overview

In order to formally define a problem for efficient resource provisioning and service placement in the fog, it is necessary to model resources involved in a fog landscape as well as to model an IoT application to be executed onto those resources, its demands and characteristics. In this chapter, we define a formal resource model for a fog landscape. This model represents the computational resources according to the conceptual architecture of the fog landscape, it defines which computational resources take part in a fog landscape, specifies their capabilities to execute certain services and provides resource capacities.

Having specified a fog landscape, we focus on the IoT application model. In this dissertation, we implement the DDF model, i.e., we model a sense-process-actuate application [66], which consists of sensing, processing, and actuating services as vertices in the directed acyclic graph of the DDF model, and communication between those services become edges in the graph. Such an IoT application consists of a set of services, which interact by means of tuples. We specify resource demands of each service and its estimated makespan duration for execution. On the IoT application level, we specify goal QoS metrics, e.g., a deadline on its execution. The DDF model and its alternatives have been discussed in detail in Section 2.2. In our modeled IoT application, sensors emit data, which is then processed. Based on the result of the processing actuators are engaged and perform certain actions.

This foundation enables us to define the FSPP, its domain definition, goal function, and constraints. We present the FSPP that places IoT services on virtualized fog resources while taking into account QoS constraints, i.e., deadlines on the execution time of applications. Their definition is used within constraints of the FSPP to calculate total execution times of IoT applications and within the goal function to minimize total QoS metric while maximizing the utilization of the fog. Based on the described formal resource model for a fog landscape and a model for IoT applications, 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. We implement the FSPP as an ILP problem as has been introduced in Section 2.5.

The remainder of this chapter is organized as follows: First, we elaborate on the problem statement and resource and application models in Section 4.2. Next, in Section 4.3 and in Section 4.4, we discuss the FSPP and formalize the optimization model. We evaluate the model extensively and discuss results in Section 4.5. After that, to solve the proposed optimization problem, we apply a different approach, namely a GA, in Section 4.6. We compare the results to the exact optimization method and a classical approach that neglects fog resources and runs all services in a centralized cloud in Section 4.7. The goal of the evaluation is to identify the best approach to solve the proposed optimization problem in

terms of resulting QoS (i.e., application response times), QoS violations (i.e., application deadline violations), and cost.

4.2 Modeling the Fog

In order to determine QoS-aware fog service placement, we need a suitable representation of the involved entities. For this, we provide a model of an IoT application which comprises the related set of IoT services, as well as QoS requirements of the application in Section 4.2.1. Then, in Section 4.2.2, we describe computational resources which collectively compose a fog landscape and communication links between them. Compared to the definitions in the system model presented in Section 3.3, here we separate application and resource models from the optimization problem. We introduce deadlines of applications and makespans of services, which provide the means to calculate deployment times. Regarding the fog, we introduce more properties to consider of fog cells and fog nodes within fog colonies and cloud resources, as well as corresponding relationships and delays between different reosurces.

The application model contains all the definitions related to applications and services and their characteristics, while the resource model provides descriptions of the fog landscape resources together with their capabilities. Finally, in Section 4.3, we formulate the FSPP as a mapping between applications and resources that takes into account QoS requirements. For the sake of clarity, in Table 4.1 we summarize the notation used in this section. The goal function in the system model in Section 3.3 aims at maximizing the utilization and assignments at the edge of the network, while in FSPP this goal is enhanced by also prioritizing submitted IoT applications which are closer to their defined deadline. Constraints also differ, since the FSPP accounts for more QoS-related constraints and models explicitly the delegation of services to a neighbor fog colony.

4.2.1 Application Model

Applications and Services

An IoT application consists of several services, which run on virtualized IoT devices (i.e., fog cells) and interact with each other to provide a well-defined functionality, i.e., the sense-process-actuate application [66], which aims to utilize computing infrastructures across the fog and resembles the stream processing approach. Hence, an application is deployed in a fog landscape by placing the services onto particular fog cells. Let $A = \cup_{i=1}^{n} A_i$ be a set of applications to be placed in a fog landscape. Each application consists of a set of services. A service $a \in A_i$ is characterized by its demands for computational resources, i.e., CPU c_a^C, RAM c_a^M, storage c_a^S, and by a service type T_a. This specification considers only static services with fixed amounts of CPU and RAM. It is clear that this is rather a limitation contrasting real-world behavior of IoT applications. Nevetheless, most cloud platforms do not consider dynamic load service descriptions and offer

fixed capacities of their services. A solution to tackle this issue in the future may be to regularly update the application model, e.g., daily, to adjust the load according to operation measurements of services.

The type T_a specifies the requirement of a service to be executed on a specific kind of resource, e.g., equipped with a specific sensor. This was not considered in Chapter 3. Without loss of generality, we consider three types of services: sensing, processing, and actuating services. These types of services account for most possible scenarios in the IoT [64]: sensing and actuating services require immediate action and, therefore, need certain equipment in IoT devices, whereas processing services may require more computational resources and can be executed either at the edge of the network or also in the cloud. A service a has an estimated makespan duration m_a.

QoS Parameters

An application A_i is characterized by a user-defined deadline D_{A_i} which is the maximum amount of time that is allowed for the application execution. The parameter w_{A_i} keeps record of the current deployment time for an application A_i. The application response time r_{A_i}, with $A_i \in A$, depends on the makespans of its services $a \in A_i$, on the communication delays among the services, and on the deployment time. The deployment time w_{A_i}, with $A_i \in A$, comprises the time needed to compute and enact the service placement plan.

4.2.2 Resource Model

The resource model is based on the conceptual architecture of the fog landscape presented in Section 3.2. The resource model reflects the fog resources, i.e., fog cells, fog nodes, cloud resources, their capabilities and relationships between them to be further used for resource provisioning and service placement.

Modeling of Fog Cells

A *fog cell* f is a virtualized single IoT device coordinating a group of other IoT devices, i.e., sensors and actuators. Fog cells run on IoT devices with computational, network, and storage capacities, e.g., gateways or a Cyber-Physical System (CPS), while 'pure' sensors and actuators are IoT devices without such resources. A fog cell is a software environment that provides access, control, and monitoring of an underlying physical IoT device. Computational resources of a fog cell are less powerful than those provided by the cloud. A fog cell is characterized by its capacities CPU C_f^C, RAM C_f^M, and storage C_f^S. There is no direct communication between fog cells, instead fog cells communicate via fog nodes in the colony, as has been discussed in Section 3.2.

Modeling of a Fog Colony

Fog cells are organized in a fog colony. Each colony is identified and managed by its head fog node F. We denote $Res(F)$ the set of fog cells which are managed by F. A colony acts as a micro data center with resources scattered in a certain geographical area, i.e., each fog cell belongs to the resources of a colony $f \in Res(F)$. The communication link between a fog cell f and the head fog node F is characterized by a not negligible delay. Under the assumption of symmetric

Table 4.1: Problem notation.

	Parameter	Description
Time	t	Current time
	τ	Interval between two solutions of the FSPP (in sec)
Applications	A	Set of applications to be executed
	A_i	Application
	D_{A_i}	Deadline for an application (in sec)
	w_{A_i}	Current deployment time of an application (in sec)
	r_{A_i}	Response time of an application (in sec)
	a	Service in an application
	c_a^C	CPU demand of a service (in mips)
	c_a^M	RAM demand of a service (in MB)
	c_a^S	Storage demand of a service (in MB)
	m_a	Makespan of a service (in sec)
	T_a	Type of a service
Fog Landscape	R	Cloud
	F	Head fog node
	N	Closest neighbor fog node
	C_F^C	CPU capacity of F (in mips)
	C_F^M	RAM capacity of F (in MB)
	C_F^S	Storage capacity of F (in MB)
	$Res(F)$	Fog cells in a colony
	f	Fog cell
	C_f^C	CPU capacity of f (in mips)
	C_f^M	RAM capacity of f (in MB)
	C_f^S	Storage capacity of f (in MB)
	$Res^a(F)$	Fog cells that can host a service a
	d_f	Link delay between F and f (in sec)
	d_R	Link delay between F and R (in sec)
	d_N	Link delay between F and N (in sec)

links and considering that a fog cell f can communicate only with the head fog node F of its own colony, we denote d_f as the communication link delay between f and F with $f \in Res(F)$. Colonies communicate with each other via their head fog nodes.

Modeling of Fog Nodes

A fog node is a more powerful fog cell as has been already introduced in Section 3.2, but, unlike a fog cell, it does not directly manage IoT devices. An example of such a fog node can be a proxy server or a gateway. Usually, fog nodes are supplemented with faster and more expensive computational resources than fog cells, however, at the same time, slower and cheaper than cloud resources. The head fog node F is characterized by its capacities CPU C_F^C, RAM C_F^M, and

storage C_F^S. Each fog node has to perform resource provisioning by exploiting the computational resources of the colony for the placement of services. Apart from resource provisioning, the purposes of fog nodes are to:

- Perform infrastructural changes in their fog colonies,
- Analyze resource utilization in the colonies, and
- Monitor fog cells.

Modeling of the Cloud

A cloud R has theoretically unlimited resources. The interaction between the fog node F and the cloud R is performed via the fog controller. The logical link between the fog node F and the cloud R has a not negligible delay d_R. The fog controller is a central unit that manages the execution of services in the cloud and supports the underlying fog landscape. Importantly, the fog controller is able to overrule fog nodes in fog colonies, but the latter may also act autonomously in the case that no fog controller is available.

4.3 FSPP: Fog Service Placement Problem

Goals

Based on the application and resource models, we are now able to define the FSPP. The FSPP aims to determine an optimal mapping between IoT applications and computational resources with the objective of optimizing the fog landscape utilization while satisfying QoS requirements of applications. In the FSPP, we consider a proactive scenario of service placement, where the placement of services is calculated and applied periodically. Such a proactive scenario fits the volatility of the IoT (with regard to data to be processed and also with regard to IoT devices leaving and entering the system) better than static resource provisioning approaches [62]. To account for QoS requirements of applications, i.e., deadlines, historical data about deployment time in fog colonies has to be preserved. This data is used to calculate an upper bound (i.e., worst-case estimation) of the response time of applications and to prioritize applications which are closer to the deadline.

Problem Statement

We define the FSPP as a decentralized optimization problem, i.e., when an IoT application request is submitted to a head fog node F, the latter solves an instance of the FSPP and performs the service placement. For that, every fog node considers a local view on the colony and accounts for the closest neighbor colony and the cloud. The fog node F places the application services on the computational resources available in its colony (i.e., fog cells, the fog node itself). If there are not enough resources, it sends the services either to the closest neighbor colony, or to the cloud. In the following sections, we describe mechanisms not only to choose the closest neighbor colony, but also to account for the most effective colony among the neighbors. With respect to the fog node F, we denote N as the head fog node of the closest neighbor colony and d_N as the communication link delay between F and N.

Assignments

Specifically, the fog node F manages fog cells which are resources of the colony $Res(F)$. Recalling the application model, we observe that a service $a \in A_i$ cannot be placed on every fog cell $f \in Res(F)$ because of its type T_a that may require specific resources (e.g., sensors). Therefore, for each $a \in A_i$ we consider a subset of fog cells, where it can be deployed, i.e., $Res^a(F) \subseteq Res(F)$.

To sum up, F determines the following subsets of placements:

- Which services have to be executed in its colony?
- Which services have to be executed locally on F's own resources?
- Which services have to be delegated to the closest neighbor colony N?
- Which services have to executed in the cloud R?

Afterwards, the fog node is able to perform placement of services on the identified fog resources. The concrete model on how services are placed on fog resources is presented in the next subsection.

4.4 Formal Definition of the FSPP

The FSPP is solved by the fog node F periodically, every τ time units, i.e., seconds. The application requests submitted between $t - \tau$ and t are scheduled for placement in t. We provide a formulation of the FSPP considering t as the current time. After defining the preliminaries and problem notation, we formulate the FSPP as an ILP problem.

Variables The variables of the optimization model are binary and indicate a placement of a specific service on specific fog resources. The binary variable $x_{a,f}$ indicates whether a service a is placed on a fog cell f in the current colony. y_a defines that a service a is placed on the fog node. The binary variable z_a defines whether a service a needs to be executed in the cloud, and n_a indicates whether a service a is delegated to the neighbor colony. Variables $x_{a,f}$ implicitly account for the types of services and types of IoT devices through $Res^a(F)$. Analogously, variables y_a and n_a indicate that the head fog node and the neighbor fog colony represented by its head fog node N can host a service a.

Goal Function The goal function of the optimization model is expressed by (4.1) and maximizes the utilization of the fog landscape while satisfying application requirements.

$$\max \sum_{A_i}^{A} \Big(\frac{1}{D_{A_i} - w_{A_i}} \sum_{a}^{A_i} \Big(\sum_{f}^{Res^a(F)} x_{a,f} + y_a + n_a \Big) \Big) \tag{4.1}$$

To account for deadlines, the goal function prioritizes those applications which have a closer deadline. For that, we use the coefficient $\frac{1}{D_{A_i} - w_{A_i}}$ which depends on the deadline and the deployment time of an application. This coefficient

becomes more influential when the deadline is close or the deployment time of the application is significant.

Constraint on Assignment

Constraints As a first constraint, each service a has to be placed either in the fog or in the cloud:

$$\sum_{f}^{Res^a(F)} x_{a,f} + y_a + z_a + n_a = 1,\ \forall a \in A_i, \forall A_i \in A \tag{4.2}$$

Constraints on Resources

Second, placed services must not exceed the capacities of CPU, RAM, and storage of a corresponding fog resource. Equations (4.3)-(4.4) guarantee that the services placed on fog cells or on the fog node do not exceed a given percentage μ of available resources.

$$\sum_{a}^{A_i} c_a^\gamma x_{a,f} \leq \mu C_f^\gamma,\ \gamma = \{C, M, S\},\ \forall f \in Res(F) \tag{4.3}$$

$$\sum_{a}^{A_i} c_a^\gamma y_a \leq \mu C_F^\gamma,\ \gamma = \{C, M, S\} \tag{4.4}$$

Constraint on QoS

As a third constraint, we account for the response time r_{A_i} of an application, so that it does not violate the application deadline as in (4.5).

$$r_{A_i} \leq D_{A_i},\ \forall A_i \in A \tag{4.5}$$

Response Time Calculation

The response time has to account for the time until the next optimization period and for an average deployment time of services in the closest neighbor colony. The calculation of r_{A_i} is done from the central point of view of the fog node in the colony. The application response time r_{A_i} is defined in (4.6) and depends on the overall makespan m_{A_i} and the overall deployment time w_{A_i}.

$$r_{A_i} = m_{A_i} + w_{A_i},\ \forall A_i \in A \tag{4.6}$$

Makespan Calculation

The overall makespan m_{A_i} accounts for the time needed to transfer services to computational resources, execute them, and retrieve the results, considering that the communications are coordinated by the fog node. The makespan m_{A_i} depends on the execution time of each service according to its placement as in (4.7)–(4.11). We define m_{A_i} as:

$$m_{A_i} = \sum_{a}^{A_i} \Big(\sum_{f}^{Res^a(F)} d(a,f)x_{a,f} + d(a,F)y_a + d(a,R)z_a + d(a,N)n_a \Big) \tag{4.7}$$

where $d(a, f)$, $d(a, F)$, $d(a, R)$, and $d(a, N)$ represent the makespan of a when it is executed on the fog cell f, the fog node F, the cloud R, and the neighbor colony N, respectively. These variables are defined in (4.8)-(4.11):

$$d(a, f) = 2d_f + m_a, \forall a \in A_i, \forall f \in Res^a(f) \tag{4.8}$$

$$d(a, F) = m_a, \forall a \in A_i \tag{4.9}$$

$$d(a, R) = 2d_R + m_a, \forall a \in A_i \tag{4.10}$$

$$d(a, N) = 2d_N + m_a, \forall a \in A_i \tag{4.11}$$

Deployment Time Calculation

The overall deployment time w_{A_i} depends on the already experienced deployment time at the time $t - \tau$, and the possibly additional time if any $a \in A_i$ is delegated to the closest neighbor colony for execution. We define w_{A_i} in 4.12:

$$w_{A_i} = w^{\tau}_{A_i} + n_{A_i}\tau + \overline{T}_{w_N} n_{A_i} \tag{4.12}$$

where the first term $w^{\tau}_{A_i}$ is the already experienced deployment time of A_i, the second term $n_{A_i}\tau$ considers that if a service is delegated to the closest neighbor colony, the placement will be performed in the next round of the FSPP, i.e., the application has to wait at least τ time units before its execution. The term $\overline{T}_{w_N} n_{A_i}$ models the additional deployment time spent in the closest neighbor colony before the application execution. The calculation of the real value of the average deployment time in the closest neighbor colony $\overline{T}_{w_N}$ requires to look ahead in time. To simplify the formulation, we rely on the estimation of $\overline{T}_{w_N}$ based on historical data. To this end, we define $\overline{T}_{w_N}$ as a moving average of parameter α of T_{w_N}, i.e., the sampled deployment time per each service delegated to N:

$$\overline{T}_{w_N} = \alpha T_{w_N} + (1 - \alpha)\overline{T}^{\tau}_{w_N}, \alpha \in [0, 1] \tag{4.13}$$

where $\overline{T}^{\tau}_{w_N}$ is the average deployment time in the closest neighbor colony in the previous round of the FSPP.

Delegation to the Neighbor Colony

Finally, we introduce variables n_{A_i} that charge a supplementary deployment time per application A_i if at least one of its services $a \in A_i$ is delegated to the closest neighbor colony N given that $|A_i|$ is the cardinality of A_i:

$$n_{A_i} \leq \sum_{a}^{A_i} n_a \tag{4.14}$$

$$n_{A_i} \geq \frac{\sum_{a}^{A_i} n_a}{|A_i|} \tag{4.15}$$

Example 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 $A_1 =$

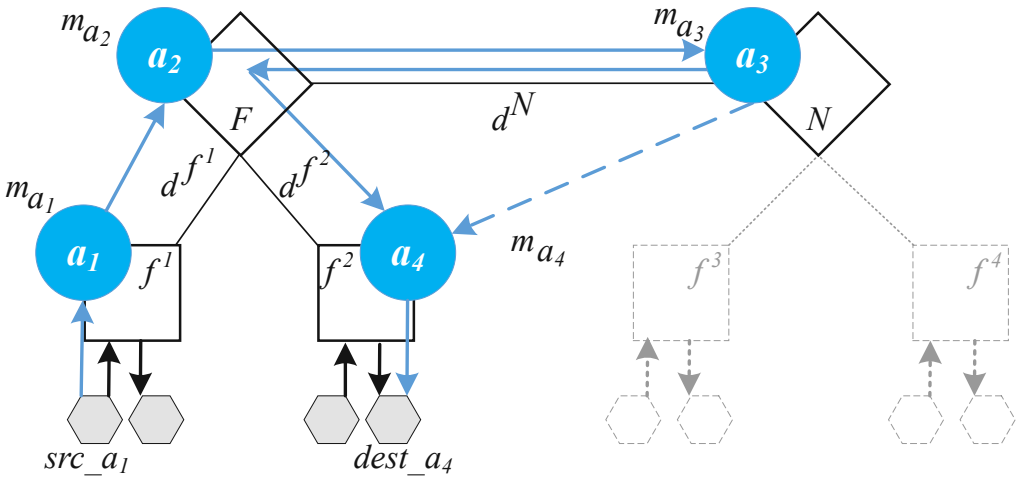

Figure 4.1: Example of service placement.

$\{a_1, a_2, a_3, a_4\}$ 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 delegated to the closest neighbor colony N, and a_4 is placed on f^2 (Figure 4.1). 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 fog node N the average deployment time is $\overline{T}_{w_N}(t)$. The response time r_{A_1} is calculated according to (4.6)–(4.11) as shown in (4.16). To be able to calculate estimations of the factors of data transfers between services in an application, as necessary for (4.8)–(4.11), we use the notion of triangular inequality in the network [44], which helps to find an approximate distance and corresponding delay between two network locations, which is assumed to be bigger than the direct delay between two locations (see Figure 4.1).

$$\begin{aligned} r_{A_1} =& d^{f^1} + m_{a_1} + m_{a_2} + d^N + m_{a_3} + \\ & d^N + d^{f^2} + m_{a_4} + \overline{T}_{w_N}(t) + \tau \end{aligned} \tag{4.16}$$

Domain Definitions Finally, (4.17)–(4.21) specify the domain definitions for the optimization model:

$$x_{a,f} \in \{0,1\},\ \forall a \in A_i,\ \forall A_i \in A,\ \forall f \in Res^a(F) \tag{4.17}$$

$$y_a \in \{0,1\},\ \forall a \in A_i,\ \forall A_i \in A \tag{4.18}$$

$$z_a \in \{0,1\},\ \forall a \in A_i,\ \forall A_i \in A \tag{4.19}$$

$$n_a \in \{0,1\},\ \forall a \in A_i,\ \forall A_i \in A \tag{4.20}$$

$$n_{A_i} \in \{0,1\},\ \forall A_i \in A \tag{4.21}$$

4.5 Evaluation of the FSPP Optimization

In the following evaluation, we show the benefits of the FSPP in terms of processing cost, response times, and deadline violations with respect to a baseline approach and to the execution in the cloud. For this, we use and extend *iFogSim*, which is a modeling and simulation toolkit for IoT and fog computing environments [66].

4.5.1 Evaluation Environment

iFogSim Extensions

We extend iFogSim with the means to account for the structure of the fog landscape introduced in Section 4.2. The *Application* and *AppModule* classes are modified to account for deadlines and deployment times of applications. The *FogDevice* class is extended by *FogLandscapeDevice* to act as a fog cell and a fog node. This class includes means to account for the utilization of a colony's resources, to store average deployment times and types of services which can be executed by fog cells and control nodes, to calculate the closest colony from all neighbor colonies for the purpose of service delegation, and to calculate a moving average of deployment times of applications in a neighbor colony, which is needed to receive an upper bound estimation of response times of applications (as described in Section 4.4). The *ModuleMapping* class provides the mapping from a fog node's name to a list of instances to be launched on that fog node. The *ModulePlacementOptimization* is a created API to ease the implementation of various resource provisioning approaches, to unify the access to the simulation environment, and to interpret and activate the obtained placement results. The baseline scenario is represented by the *ModulePlacementFirstFit* class, which implements this API and performs service placement by searching a first-fit fog resource. The discussed extensions are presented in Figure 4.2.

Extension for the Model

The optimization model is implemented by means of the IBM Ilog CPLEX Optimization Studio[1] and the open source Java ILP library[2]. The implemented *FogSolverCPLEX* class performs the mapping from Java ILP to IBM CPLEX. The *ModulePlacementExact* class implements the API and solves the model. It has to be noted that in order to use IBM CPLEX to solve the described optimization problem, the constraints are transformed to their canonical form.

4.5.2 Experimental Setup

Scenarios

To evaluate the efficiency of the FSPP, we apply a first-fit placement (called the 'baseline' scenario) and the optimization model of the FSPP as introduced in Section 4.4 (called the 'optimization' scenario) in the fog landscape. Furthermore, we perform an execution of all applications in the cloud for the case the fog landscape is not available (called the 'cloud' scenario).

[1] Online; Accessed: Apr. 2023 https://www.ibm.com/products/ilog-cplex-optimization-studio

[2] Online; Accessed: Apr. 2023 https://sourceforge.net/projects/javailp/

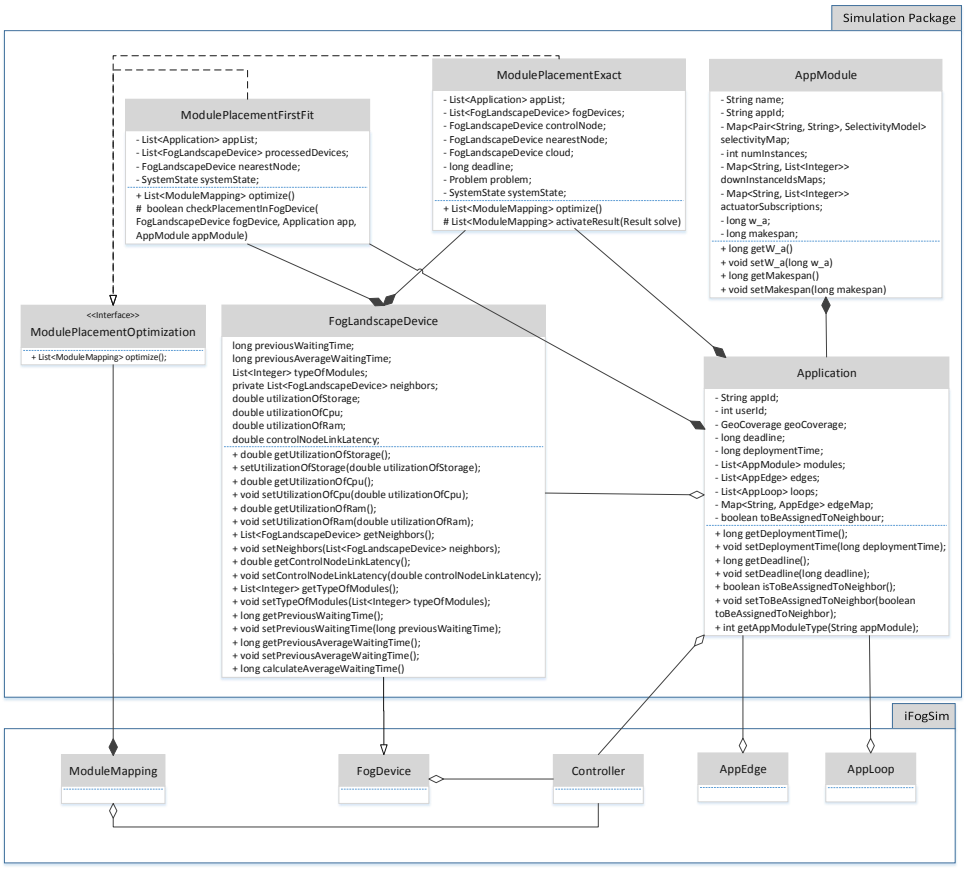

Figure 4.2: Extensions to iFogSim.

Application Settings Four different sense-process-actuate application requests are submitted for execution in the fog colony. Each fog cell is connected to four different sensors and actuators corresponding to each of the applications, i.e., motion, video, sound, and temperature sensors and actuators. Each application consists of several services. A summary of the setup is provided in Table 4.2.

Fog Landscape Setup As fog landscape, we consider several fog colonies. One colony is the main colony, where service placements are observed. From the remaining neighbor colonies, the closest neighbor is identified according to the least communication link delay (see Section 4.2). The fog node can host processing and actuating services. The neighbor colony can host only processing services, and the cloud can host all services. The communication link delay between the fog node and the cloud is set to 2 seconds. The cost per processing in the cloud is set to $ 0.30 per BTU, i.e., one hour. The communication link delay between the fog node and the neighbor fog node is set to 0.5 seconds. The average deployment time of applications in the

Table 4.2: Application details.

Services	c_a^C (mips)	c_a^M (MB)	m_a (sec)
Application A_1, $D_{A_1} = 300$ sec, $w_{A_1} = 0$ sec			
SenseMotion	50	30	0.10
ProcessMotion1	100	10	0.10
ProcessMotion2	100	20	0.10
ProcessMotion3	200	30	0.08
ActuateMotion	50	20	0.80
Application A_2, $D_{A_2} = 240$ sec, $w_{A_2} = 60$ sec			
SenseVideo	50	30	0.90
ProcessVideo1	200	100	0.10
ProcessVideo2	200	20	0.10
ProcessVideo3	100	30	0.25
ActuateVideo	50	20	0.50
Application A_3, $D_{A_3} = 360$ sec, $w_{A_3} = 60$ sec			
SenseSound	50	30	0.40
ProcessSound1	100	10	0.10
ProcessSound2	200	20	0.07
ProcessSound3	200	30	0.35
ActuateSound	50	20	0.40
Application A_4, $D_{A_4} = 360$ sec, $w_{A_4} = 0$ sec			
SenseTemperature	50	30	0.40
ProcessTemperature1	200	10	0.70
ProcessTemperature2	200	20	0.10
ProcessTemperature3	200	30	0.90
ActuateTemperature	50	20	0.30

neighbor colony is set to 3 minutes, the average deployment time of applications in the neighbor colony of the previous optimization period is set to 2 minutes. The fog node in a colony has 500 mips of CPU power, 512 MB of RAM, and 8 GB of storage. 10 fog cells are connected to the fog node. These fog cells are of less computational and storage power than the fog node, i.e., 250 mips of CPU power, 256 MB of RAM, and 4 GB of storage. These fog cells can host sensing and actuating services. The communication link delay between fog cells and the fog node is set to 0.5 seconds.

4.5.3 Results and Discussion

Baseline

This experiment aims to show how the model behaves depending on various deadlines and deployment times of the given applications. In the baseline scenario, for applications A_1, A_3, and A_4 sensing and actuating services are placed on different fog cells, and processing services are delegated to the closest neighbor

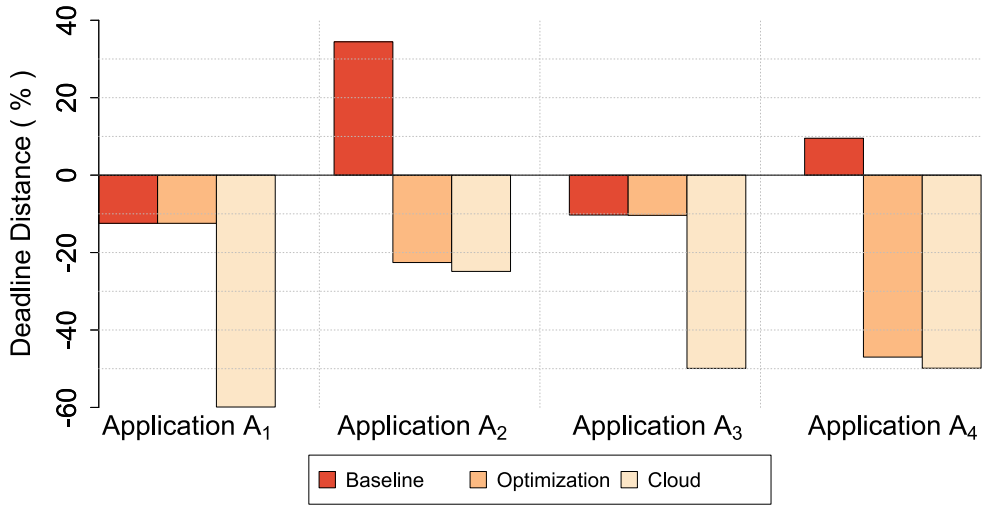

Figure 4.3: Deadline distance.

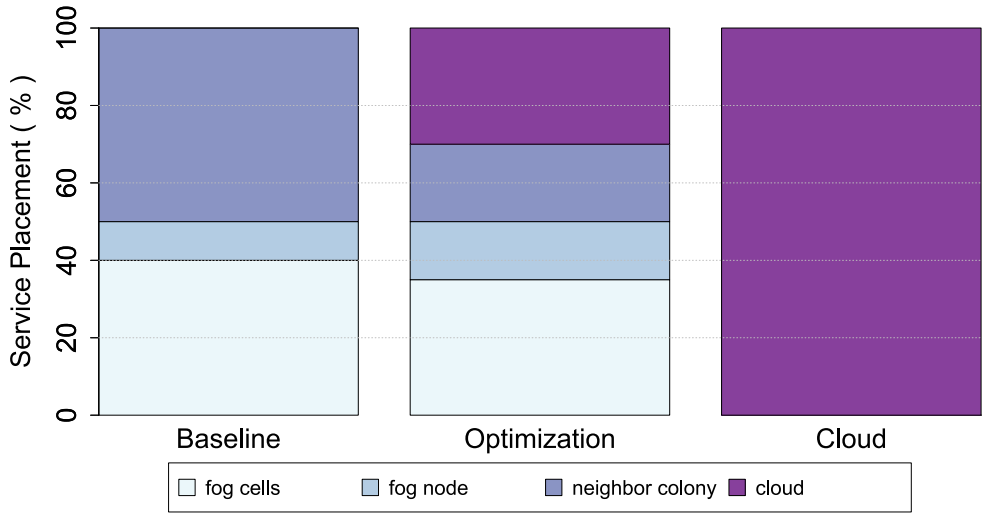

Figure 4.4: Service placement.

colony. For application A_2, two processing services are placed on the fog node, and one processing service is delegated to the neighbor colony. In the course of the experiment, the baseline scenario results in deadline violations for A_2 and A_4 by 82.65 and 22.87 seconds respectively.

Optimization In the optimization scenario, if it is not advantageous for an application to wait for the deployment in the closest neighbor colony because of the close deadline, the optimization model places the services in the cloud. If the deadline is not

Table 4.3: Scenario comparison: Baseline vs. Optimization vs. Cloud.

Metrics	Baseline	Optimization	Cloud
Application A_1, $D_{A_1} = 300$ sec, $w_{A_1} = 0$ sec			
Response time (sec)	262.66	262.66	120.31
Deadline distance (sec)	-37.34	-37.34	-179.69
Application A_2, $D_{A_2} = 240$ sec, $w_{A_2} = 60$ sec			
Response time (sec)	322.65	185.82	180.31
Deadline distance (sec)	82.65	-54.18	-59.69
Application A_3, $D_{A_3} = 360$ sec, $w_{A_3} = 60$ sec			
Response time (sec)	322.90	322.66	180.31
Deadline distance (sec)	-37.10	-37.34	-179.69
Application A_4, $D_{A_4} = 360$ sec, $w_{A_4} = 0$ sec			
Response time (sec)	262.87	126.24	120.31
Deadline distance (sec)	22.87	-113.76	-119.69
Service placement on resources (% of all services)			
Fog cells	40	35	00
Fog node	10	15	00
Neighbor colony	50	20	00
Cloud	00	30	100
Cost of execution ($)	0.000	0.069	0.107

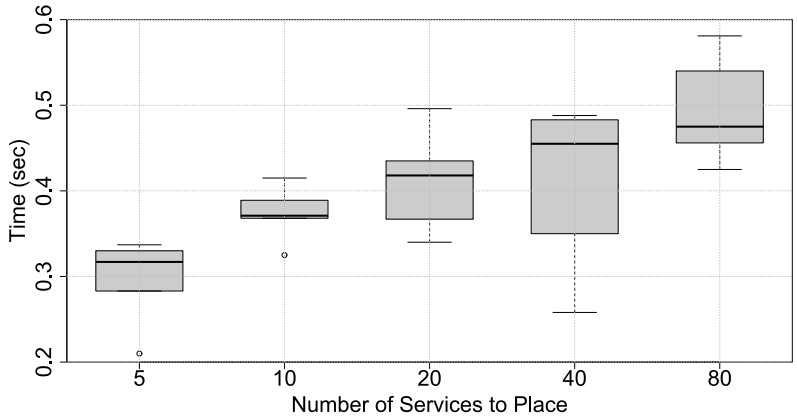

Figure 4.5: Computational time of the FSPP.

critically close, and the estimated response time of the application conforms to the constraints of the optimization model, the services are delegated to the closest neighbor colony. For application A_2, the optimization model places the sensing service on the fog cell, two processing and one actuating services onto the fog node, and one processing service in the cloud. The placement of any service from A_2 to the closest neighbor colony would violate the deadline because of

the additional deployment time that appears in the neighbor colony. In the case of A_4, the sensing and actuating services are placed on one fog cell and three services are delegated to the cloud because the fog node is already busy with A_2. For A_1 and A_3, sensing and actuating services are placed on separate fog cells, and processing services are delegated to the closest neighbor colony. To summarize, the service placement in the optimization scenario depends on the time interval τ between two subsequent FSPP optimizations in the system and on the average deployment time $\overline{T}_{w_N}$ in the closest neighbor colony, and the solution of the FSPP of the optimization scenario circumvent deadline violations of the applications. If τ and $\overline{T}_{w_N}$ are small enough, the FSPP optimization model delegates the services to the closest neighbor colony. The deadline distances for the three scenarios are depicted in Figure 4.3.

Utilization The utilization of the fog landscape in the three scenarios is shown in Figure 4.4. In the baseline scenario, services are placed only on fog resources, i.e, 40% of services are placed on fog cells, 10% of services are placed on the fog node itself, and 50% services are delegated to the closest neighbor colony. In the optimization scenario, these percentages are respectively 35%, 15% and 20%, and also 30% of all services are executed in the cloud. As a result, the cloud resource is utilized only by 30%. The use of the fog node's own resources is performed more efficiently in the optimization scenario. More services with lesser resource demands are placed on the fog node rather than less services with bigger demands compared to the baseline scenario. Such behavior is provided by the prioritization of applications in the goal function according to the deadlines and deployment times, and strict adherence to the deadlines in the constraints of the optimization model.

Cost Evaluation We assume ownership of the fog, hence the cost of execution in the fog landscape can be neglected. The cost of execution in the cloud is calculated as the sum of cost per processing second in the cloud of each service divided by the amount of seconds in 1 BTU. In the optimization scenario, services are either delegated to the closest neighbor colony, or executed in the cloud, yet at the same time satisfying deadlines of applications. Therefore, execution cost accrue. Cost is around 35% less than the cost of execution in the cloud scenario, yet the optimization scenario still satisfies the application deadlines. A summary of all evaluation results is shown in Table 4.3.

Computational Time Additionally, we conducted another experiment to calculate the *computational time* of producing a solution of the FSPP depending on the amount of services to be placed. The amount of services affects the amount of decision variables and constraints in the model, and therefore, influences the computational time. In Figure 4.5, the distribution of the computational time is shown based on minimum and maximum values, first and third quartiles, and the median of the computational times. The individual outlying points of computational times are displayed as unfilled circles in the first two boxes. The measurements of computational time show that FSPP is solved in less than a second even in the

case of submitted applications with large amounts of services.

4.6 Design and Implementation of a Genetic Algorithm

Choice of the Algorithm

In general, the service placement problem has been shown to be NP-complete [81]. 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 [11, 120, 155, 168], and their lightweight nature, which helps to run GAs on resource-constrained devices at the edge of the network. The main advantage of GAs is that they investigate a large search space and provide a viable qualitative solution in polynomial time [163, 164].

Algorithm Description

As the name implies, GAs intend to mimic evolutionary processes [149]. 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 corresponding gene in the chromosome. Each gene in a

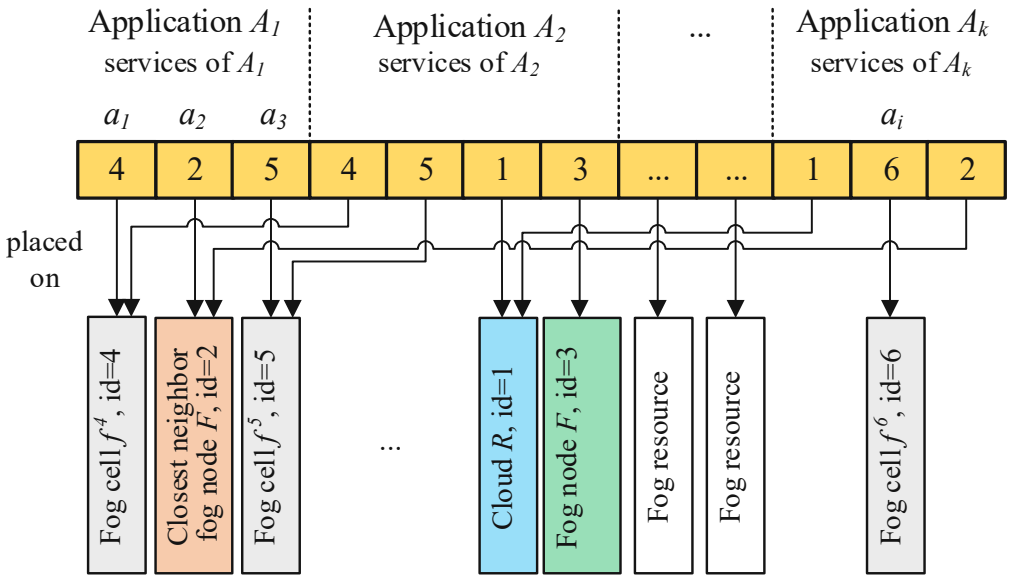

Figure 4.6: Chromosome representation.

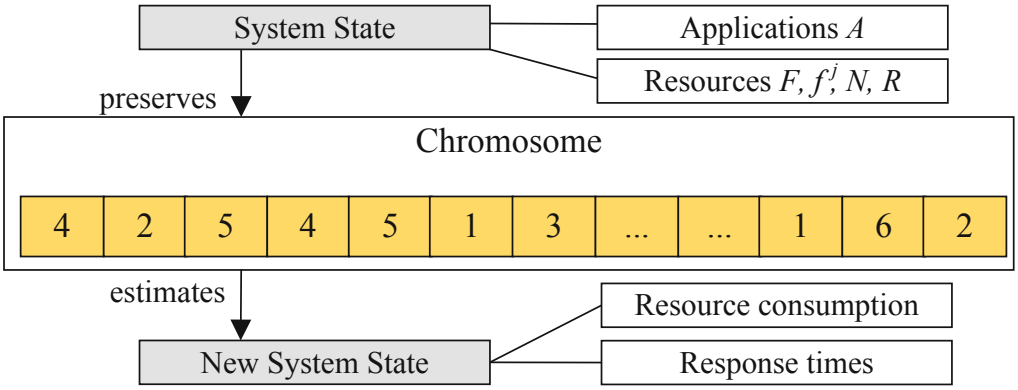

Figure 4.7: Chromosome contents.

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 node, the closest neighbor colony, or the cloud). The position of a gene in a chromosome along with its integer value represents the service placement on the resource (see Figure 4.6). Such chromosome encoding ensures the placement of all services.

System State Apart from encoding the placement, the estimated system state can be derived from the information in the chromosome. This information includes the used CPU, RAM, and storage capacities of fog resources (both fog cells and fog nodes), and corresponding 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 [159]. We consider three types of constraints:

- A set of constraints Ψ on capacities of CPU, RAM and storage resources,
- 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
- 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.

Constraints on Resources

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 (4.22):

$$\delta_{\beta_p(c)} = \begin{cases} 0, & \text{if } \beta_p(c) \leq 0 \\ 1, & \text{if } \beta_p(c) > 0 \end{cases} \tag{4.22}$$

Constraints on Assignments

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 (4.23), where β_υ denotes a constraint, and $\delta_{\beta_\upsilon(c)}$ indicates whether a constraint is violated in the current chromosome c, i.e., $\delta_{\beta_\upsilon(c)} = 1$.

$$D(c) = \sum_{\beta_\upsilon \in \Upsilon} \delta_{\beta_\upsilon(c)} \tag{4.23}$$

Penalties

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 (4.24):

$$F(c) = \sum_{\beta_p \in \Psi} \omega_{\beta_p}(1 - 2\delta_{\beta_p(c)}) + \sum_{\beta_\gamma \in \Gamma} \omega_{\beta_\gamma}(1 - 2\delta_{\beta_\gamma(c)}) - \omega_p D(c) \tag{4.24}$$

Fitness Function Calculation

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 corresponding 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 corresponding 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 in the 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 are 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 of 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.

4.7 Experimenting with Different FSPP Solutions

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 again the fog simulation toolkit *iFogSim* [66] to simulate the fog landscape and cloud resources.

4.7.1 Evaluation Environment

The iFogSim Extensions

The simulation environment has already been presented in Section 4.5.1, and is extended by the entities to enable the GA. To adhere to the FSPP constraints, the functionality to calculate a moving average of deployment times of applications in

Table 4.4: Application resource demands.

Service	U_{a_i} (mips)	M_{a_i} (MB)	S_{a_i} (MB)	m_a (sec)
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

the closest neighbor colony was implemented (as described in Section 4.3). The *ModulePlacementGenetic* class implements the solution. The *FogGeneticAlgorithm* class sets up the algorithm and its parameters, performs its execution, i.e., generates initial population of chromosomes, activates the algorithm, and specifies stopping conditions. The *FogChromosome* class is a solution of the algorithm. It contains the logic to calculate the fitness function. It also keeps the information about utilization of the fog resources in the fog colony and about the types and demands of submitted applications. It is done within the class *SystemState*. Finally, *FogUniformCrossover* and *FogRandomGeneMutation* classes perform accordingly swapping of chromosomes and mutations to enable advancing the population.

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 Section 4.4) implemented by the means of the IBM CPLEX library[3] (called the 'optimization' scenario), and the GA (called the 'genetic' scenario) introduced in Section 4.6. If the fog landscape is not available at all, the execution is performed solely in the cloud (called the 'cloud' scenario).

First-Fit Scenario

In the first-fit scenario, a service placement plan is produced by searching a first-fit fog resource [29]. 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 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 delegated to the closest neighbor colony.

Optimization Scenario

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[4] is applied.

Cloud Scenario

[3]Online; Accessed: Apr. 2023 https://www.ibm.com/products/ilog-cplex-optimization-studio

[4]Online; Accessed: Apr. 2023 https://sourceforge.net/projects/javailp/

Table 4.5: Application deadlines and deployment times.

Application	D_{A_k} (sec)	w_{A_k} (sec)
App1	120	60
App2	300	0
App3	300	60
App4	360	60
App5	240	0

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

4.7.2 Experimental Setup

Application Settings

As setup for the evaluation, we consider five different applications, i.e., motion, video, sound, temperature, and humidity processing applications, and their corresponding sensors and actuators. The IoT applications are simulated by the means of iFogSim. 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 to different input parameters. Service makespan durations are set based on received average data from pre-experiments run in iFogSim for specified services. A summary of the experimental setup is shown in Table 4.4 and Table 4.5.

Fog Landscape Setup

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

Resources

In the closest neighbor colony, the expected deployment time of applications is $\overline{T}^{t}_{w_N} = 3$ minutes, and the sampled deployment time in the previous round of FSPP period is $T^{t-\tau}_{w_N} = 2$ minutes. 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 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 delegated to the closest neighbor colony. The processing cost in the cloud is \$0.30 per BTU, i.e., one hour, as in Chapter 3.

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 GA scenario,

we set the weight factor ω =1 for all constraints [159], and the penalty weight ω_p is 1000, which were explained in Section 4.6.

4.7.3 Metrics

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

Delay The 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.

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

Insights 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.

Cost 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.

4.7.4 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, GA, 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 4.6. A summary of the results of the GA scenario is shown in Table 4.8. 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.

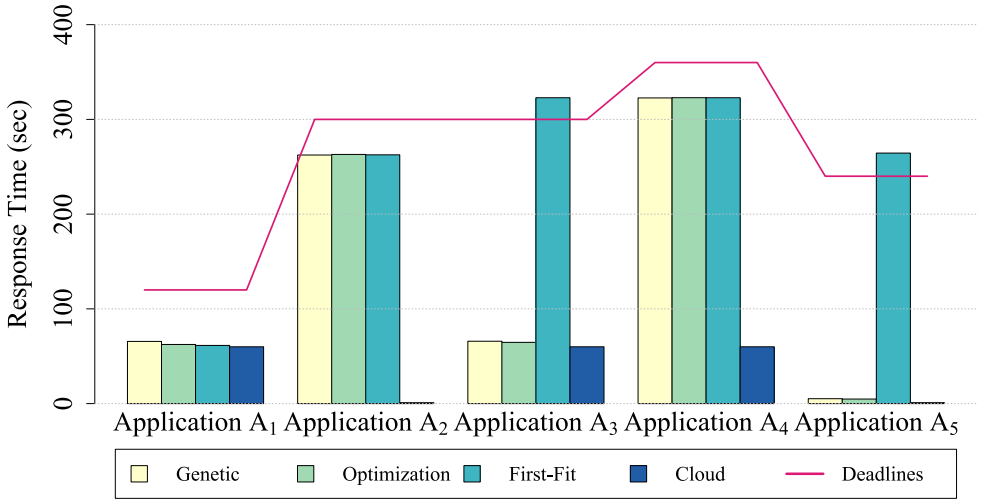

Figure 4.8: Response times of applications.

Deadline Violations

The first-fit scenario results in deadline violations for applications A_3 and A_5 by 22.87 and 24.43 seconds, respectively. In the GA, 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 4.6). In the first-fit scenario, the services are delegated 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 delegated to the closest neighbor colony. Because the resources in the fog landscape are less powerful than cloud resources, the spikes in Figure 4.9 appear. Even though the deadlines in the GA 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.

Service Delays

The response times of applications and delays for single service executions in the four scenarios are depicted in Figure 4.8 and Figure 4.9.

Cloud Execution

In Figure 4.8, the cloud scenario does not violate deadlines, 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.

Impact of Application Load

To show the impact of application parameters on response times, we conduct another experiment which varies the deadline parameter of application A_4 (Figure 4.10) and observes response times of all applications. A_4 was chosen for this experiment as it has the biggest response time (see Table 4.6 and Table 4.7).

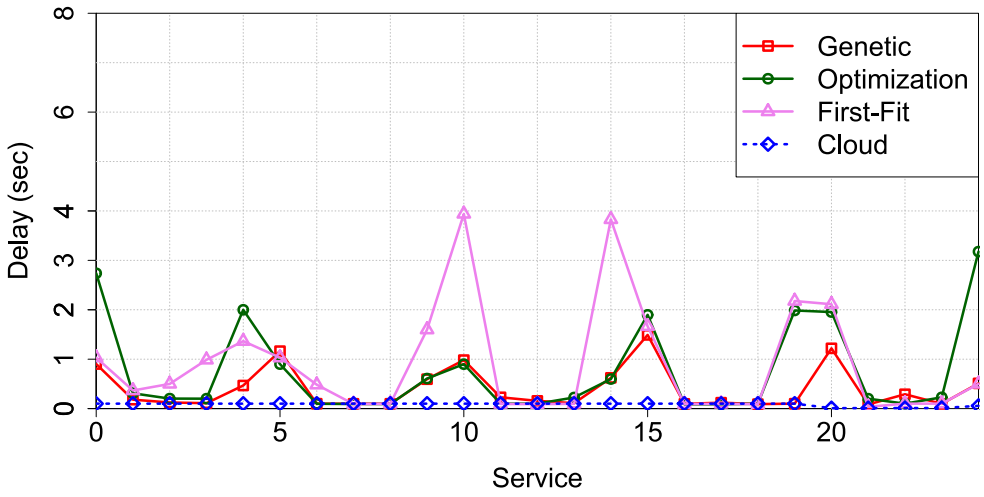

Figure 4.9: Service execution delays.

Table 4.6: Response times and deadlines of applications in different scenarios.

Scenario	Applications and Metrics				
	A_1	A_2	A_3	A_4	A_5
	Response Time r_{A_i} (sec)				
Optimization	62.43	262.6	64.66	322.9	4.89
First-Fit	61.42	262.6	322.8	322.8	264.4
Cloud	60.00	0.001	60.00	60.00	0.001
	$D_{A_k} - r_{A_k}$ (sec)				
Optimization	57.56	37.34	235.3	37.1	235.1
First-Fit	58.57	37.34	-22.87	37.11	-24.43
Cloud	59.99	299.9	239.9	299.9	239.9

Given the parameters of the applications as shown in Table 4.5, when the deadline is below 120 sec, the processing services of A_4 are assigned only to the fog node as this is the closest deadline among all applications. Above 120 sec and below 300 sec, 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 node, and the remaining two processing services of A_4 are executed in the cloud. Above 300 sec, A_4 has enough time to wait for the deployment, and therefore, some services are delegated to the closest neighbor colony, and both τ and $\overline{T}^t_{w_N}$ affect the response time of A_4.

Utilization in First-Fit

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 fog node, and the remaining 11 processing services are delegated to the closest

Table 4.7: Scenario performance comparison with regard to placement and cost.

Scenario	Metrics				
	Placement (%)				Cost
	f	F	N	R	($)
Optimization	40	24	24	12	0.09
First-Fit	40	16	44	0	0.00
Cloud	0	0	0	100	4.81

Table 4.8: GA performance in terms of response time.

Metrics	A_1	A_2	A_3	A_4	A_5
r_{A_i} (sec)	65.65 ($\sigma = 1.48$)	262.47 ($\sigma = 0.27$)	65.83 ($\sigma = 1.38$)	322.64 ($\sigma = 0.25$)	5.19 ($\sigma = 0.29$)
$D_{A_i} - r_{A_i}$ (sec)	54.34 ($\sigma = 1.48$)	37.52 ($\sigma = 0.27$)	234.16 ($\sigma = 1.38$)	37.35 ($\sigma = 0.25$)	234.80 ($\sigma = 0.29$)

Table 4.9: Average placement and cost of resources.

Average Placement		Cost
Resource	(% of all services)	($)
f	28.8 ($\sigma = 3.92$)	
F	11.2 ($\sigma = 1.60$)	0.22
N	24.0 ($\sigma = 2.52$)	($\sigma = 0.02$)
R	36.0 ($\sigma = 4.38$)	

neighbor colony.

GA Scenario Utilization

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 node has enough free resources to host more services. Many cloud placements occur because in the fitness function of each of the 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 node or fog cells.

Utilization during Optimization

In the optimization scenario, all ten sensing and actuating services are placed on fog cells, however the fog node hosts more services, i.e., six services. Another six services are delegated to the closest neighbor colony, and three are executed in the cloud. Such a 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 Figure 4.11.

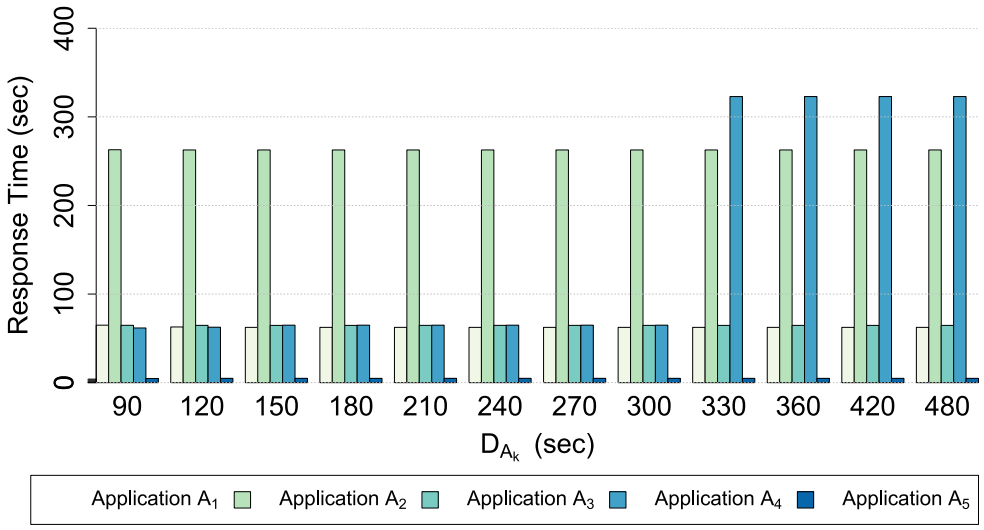

Figure 4.10: Impact of application deadlines on response times.

Previous Deployment Times

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 Figure 4.12, while $\overline{T}^t_{w_N}$ is small, the closest neighbor fog colony performs the processing fast, and therefore the head fog node assigns 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 delegated to the cloud. Also, the significant increase of $\overline{T}^t_{w_N}$ leads to the placement when all the services are executed in the cloud, i.e., the closest neighbor colony is not sufficient any more to host any service. The same considerations are valid for the τ parameter.

Varying Capacities of Resources

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 node increases (see Figure 4.13). 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.

Varying QoS Metrics

The impact of variations of deadlines of applications on the goal function is shown in Figure 4.14. 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 seconds to six minutes, 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 node as they cannot be delegated to the closest neighbor colony, because that would mean

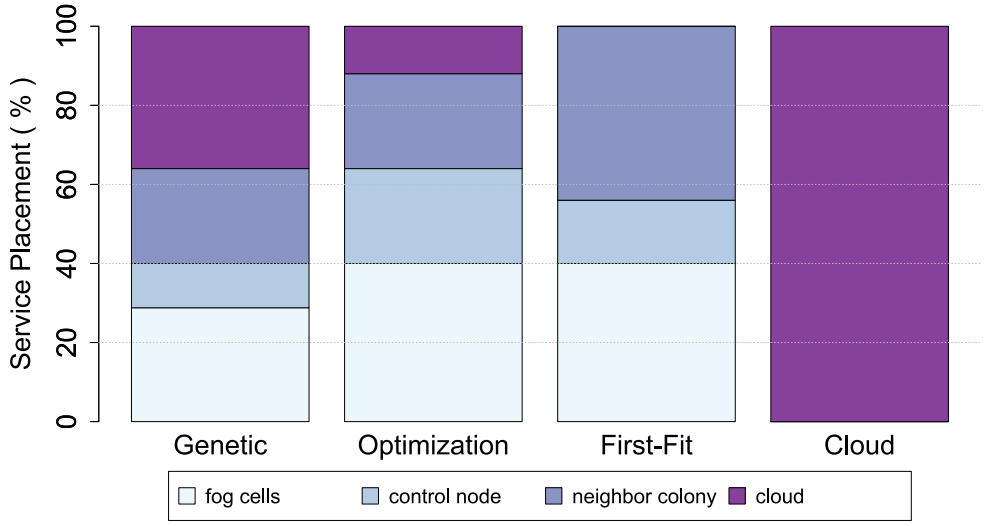

Figure 4.11: Utilization of resources.

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 three minutes, which prevents the services from the considered application to be placed on the fog node, and therefore they are delegated to the cloud. The goal function value in this case is reduced according to the coefficient $\frac{1}{D_{A_i}-w_{A_i}}$, or taking into account also makespan durations as $\frac{1}{D_{A_i}-r_{A_i}}$, and due to the fact that less services are placed in the fog. When the deadline becomes more relaxed compared to the parameters τ and $\overline{T}^t_{w_N}$, i.e., starting from five minutes, services are delegated to the closest neighbor colony, and the goal function slightly increases, as more services are hosted by the fog landscape. After that, the only reduction is observed due to the change of $\frac{1}{D_{A_i}-r_{A_i}}$.

Cost Evaluation The cost of service execution according to the GA 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 GA scenario, charging \$0.09 since only three services are executed in the cloud. The execution cost received in the optimization and GA scenarios are 2% and 4%, respectively, compared to the execution cost in the cloud scenario. The results are shown in Table 4.6 and Table 4.8.

Evaluation Outcome To sum up the most important observations of the evaluation, the execution of the service placement plans produced in the GA 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

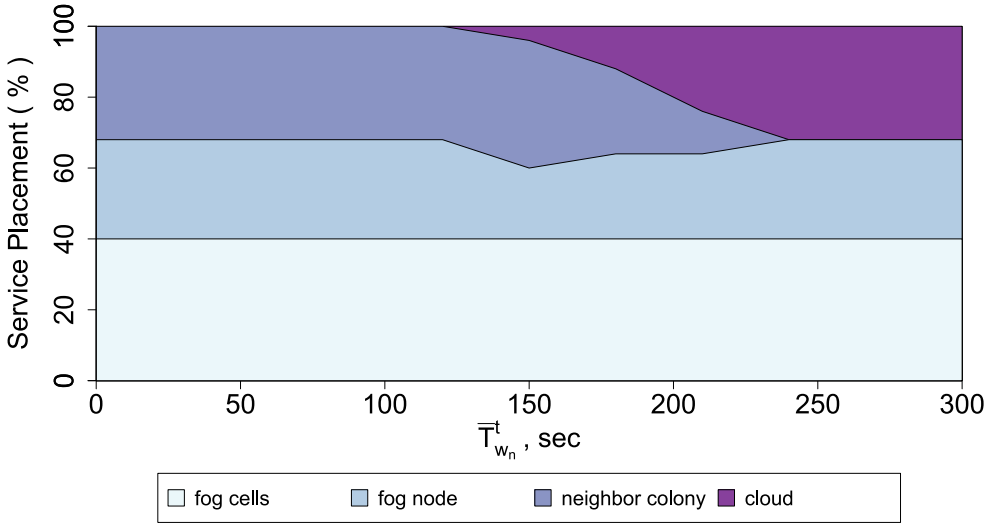

Figure 4.12: Impact of previous deployment time $\overline{T}^t_{w_N}$ on service placement.

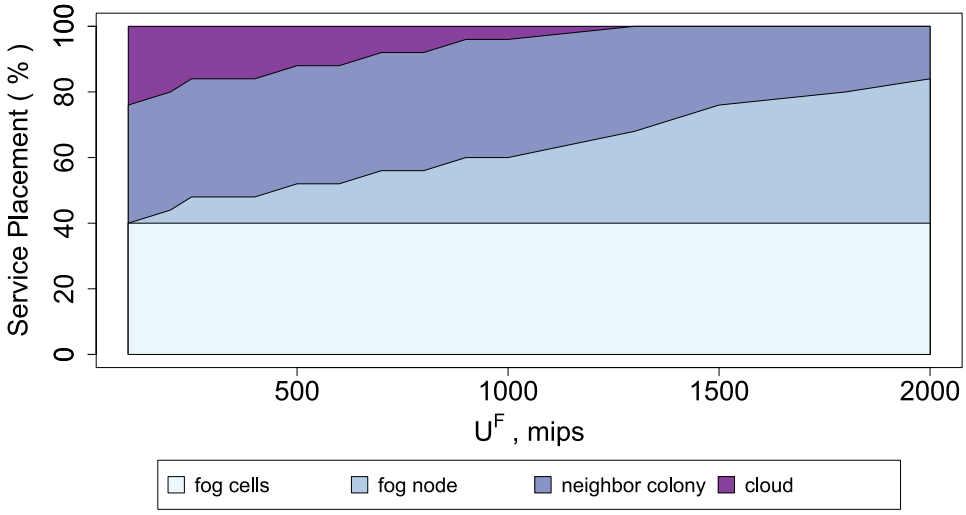

Figure 4.13: Impact of resources of the fog node F on service placement.

method better utilizes the fog landscape resources.

4.8 Summary

As we have defined in Section 1.1, one of research challenges is to achieve optimal resource provisioning and service placement in the fog. Therefore, in this chapter, we provide a model for an IoT application and a resource model for a fog landscape.

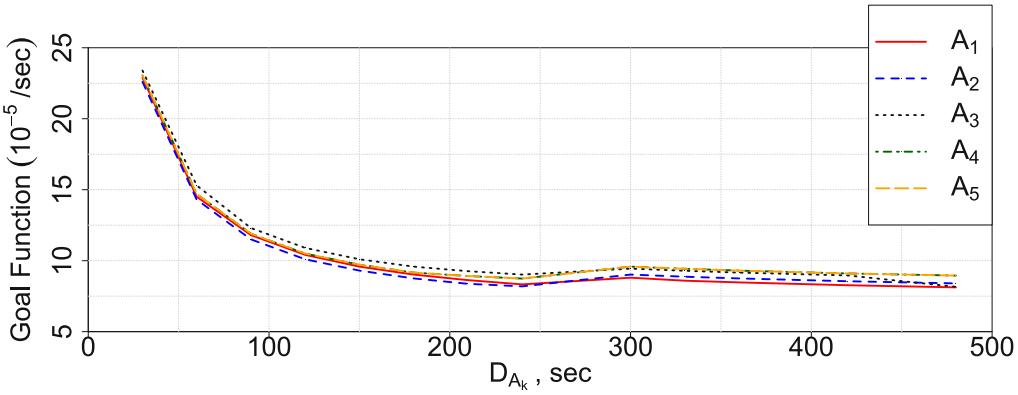

Figure 4.14: Impact of application deadlines on the goal function.

Afterwards, the FSPP is described, and an optimization model is formalized. We study the placement of IoT services on fog resources, taking into account their QoS requirements. In this chapter, we have simulated a fog landscape with multiple colonies and solved the FSPP using several approaches, i.e., a first-fit algorithm, exact optimization, and execution in the cloud. We show that our optimization model prevents QoS violations and leads to 35% less cost of execution if compared to a purely cloud-based approach. The experimental evaluation has shown that the FSPP utilizes the fog landscape for 70% of all services. The solution of the FSPP does not violate deadlines of applications unlike the solution of the baseline approach.

Next, we implement another resource provisioning approach using a GA. Also, we compare all the results with the execution of the same experimental setup with the exact optimization and with the execution in the cloud. Unlike the service placement plans produced by the first-fit heuristics, the service placement plans of the GA 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. This leads to lower execution cost when compared to the average service placement plan produced by the GA. The cost in the optimization method constitutes only 40% of the cost of service placement plans produced by the GA. The GA produces solutions which on average experience a lower deployment delay by exploiting more cloud resources, i.e., on average 36% of the services have been run in the cloud.

This chapter constitutes Contribution II as defined in Section 1.2 and covers requirements 6–10 from Section 2.6:

6. A fog landscape should provide resource provisioning in order to efficiently distribute workload among resources in the fog.
7. A resource model in a fog landscape should represent all available resources at the edge of the network and in the cloud and their corresponding

capabilities and relationships.

8. A distributed IoT application to be executed in the fog should also be modeled.
9. The resource model and the IoT application model should be used for the definition of resource provisioning in the fog, i.e., of a service placement problem.
10. The resource provisioning should adhere to the desired goals and account for fog landscape constrains.

CHAPTER 5

IoT Application Execution with FogFrame

In previous chapters, we have established theoretical principles of fog computing and considered both RAs and conceptual architectures [25, 45, 47, 141]. In this chapter, we focus on the concrete implementation of a system capable to manage a fog landscape and to efficiently execute IoT applications. We describe a system architecture for fog computing, which is built upon our presented conceptual architecture in Chapter 3. With this architecture design, we implement a representative fog computing framework called FogFrame that operates the fog. The resource provisioning and service placement approaches, which have been presented in Chapter 4, are implemented in this framework. To evaluate the framework, we conduct a series of experiments that show how service placement, deployment, and execution is performed by FogFrame.

The framework is built to provide coordinated control over the physical and virtual infrastructure of a fog landscape with computational resources and data sources potentially entering or leaving the fog at any time [124, 143]. Enabling the coordinated cooperation among computational, storage, and networking resources in the fog can be challenging due to such volatility of resources in a fog landscape [6, 33]. For this reason, within the FogFrame framework, we design necessary communication mechanisms for instantiating and maintaining service execution in a fog landscape during runtime.

In this chapter, we explicitly show how a FogFrame-based fog landscape is created and maintained, how it evolves at runtime reacting to different events, and how IoT applications are submitted, processed, and executed. Specifically, we define and implement fault-tolerant resource provisioning and service placement to enable a fog landscape to evolve at runtime. A real-world testbed is implemented

by the means of FogFrame according to the specified architecture and workflows. To evaluate the framework in the volatile fog, we conduct a series of experiments that show how the framework adapts to changes in the available resources, how it balances the workload, and how it recovers from resource failures and overloads.

5.1 Overview

To adopt fog computing, first of all cloud computing properties have to be mirrored to the edge of the network, as has been discussed in Chapter 2. Such properties include on-demand up-scaling and down-scaling, and accordingly leasing of computational resources based on needed workload. The essential technology enabling these properties in cloud computing is the virtualization of cloud resources implemented by the means of VMs and containers as has been discussed in Section 2.1. In a fog landscape, VMs are not sufficient as they are resource-intensive and need several minutes to start and deploy services, which is unacceptable for delay-sensitive IoT applications [45]. A promising solution for this issue is the utilization of lightweight containers as a virtualization mechanism for resources at the edge of the network [25, 103, 144]. Therefore, it is necessary to introduce mechanisms and principles of how to use containers both to manage a fog landscape and to execute distributed IoT applications in the fog.

In FogFrame, this issue is addressed by virtualizing and implementing fog devices in a fog landscape, i.e., fog cells and fog nodes. These fog devices control the IoT devices where they are deployed and are able to instantiate and execute IoT applications on the IoT devices. While implementing such an architecture in practice, we need to consider the volatile nature of a fog landscape, i.e., different IoT devices at the edge of the network are dynamic with regard to the geography of data sources and volume of data to be processed, as well as regarding resource capacities, intended functionalities, and capabilities of IoT devices. Taking into account this volatile nature of the fog landscape and the research questions stated in Section 1.1, we define the following main goals to be achieved by FogFrame: (i) to create and maintain a fog landscape made up from computational resources at the edge of the network and in the cloud, (ii) to establish communication and interaction in the fog landscape, (iii) to efficiently deploy and execute IoT applications. For the latter, in order to efficiently execute IoT applications in a fog landscape, we study the problem of how to distribute services of IoT applications between resources in a fog landscape.

Our contributions can be summarized as follows:

- We design and implement the FogFrame framework, which provides communication and interaction of virtualized resources within a fog landscape.
- We implement functionalities for decentralized service placement, deployment and execution in a fog landscape. Service placement is performed by two heuristic algorithms – a greedy algorithm and a GA. We distinguish ser-

vice deployment at the edge of the network and in the cloud and implement deployment mechanisms.
- We develop mechanisms to react to runtime operational events in the fog landscape, namely, when devices appear and disappear in the fog landscape, and when devices experience failures and overloads. The framework identifies those events and migrates necessary services to balance workload between different resources.
- We evaluate the capabilities of FogFrame with regard to service placement, adherence to QoS parameters, and utilization of fog resources.

In this chapter, the work is based on findings from the previous chapters, where we researched fog computing environments and different service placement approaches, and evaluated them using the simulators CloudSim [34] and iFogSim [66]. However, instead of continuing to use simulators to create the fog infrastructure for testing different resource provisioning approaches, we have implemented FogFrame. Therefore, as an additional outcome, our framework can be freely used in the research community to develop and evaluate different resource provisioning methods for the fog. This is enabled by providing a working software publicly available within our Github repository[1] for reimplementing exchangeable loosely-coupled components, building and connecting a fog computing environment, and executing IoT applications.

The remainder of this chapter is organized as follows: We describe how to configure a fog landscape and enable communication between entities in FogFrame in Section 5.2. Next, we discuss how the fog landscape evolves at runtime. Section 5.3 is devoted to events that can take place in a fog landscape during the execution of IoT applications at runtime. Section 5.4 provides the design details of FogFrame. In Section 5.5, we provide application management mechanisms. Afterwards, Section 5.6 describes the adaptations of the system model for service placement and corresponding placement algorithms in FogFrame. The framework implementation and experiments with resource provisioning are discussed in Section 5.7 and Section 5.8. In Section 5.9, we show experiments and evaluation of how FogFrame reacts to volatile changes in the fog.

5.2 Workflows in a Fog Landscape

We have already introduced a conceptual fog landscape in Section 3.2. In this section, we continue to expand the architecture by taking into account the specification of how a fog landscape can be created and instantiated, and defining necessary workflows and interactions.

Initialization

Setting a fog landscape starts with instantiating the fog controller, which is an initial communication point for fog colonies. After instantiating the fog controller,

[1] Online; Accessed: Apr. 2023 `https://github.com/softls/FogFrame-2.0`

fog devices can enter the fog landscape and start forming fog colonies. When creating the fog landscape, we follow the assumptions that

- All fog devices are able to provide their location data,
- All fog devices are configured with the fog controller communication data to request joining the fog landscape (or are able to get the fog controller communication data, e.g., IP address, through some bootstrapping mechanism), and
- All fog nodes can operate within predefined coverage areas, can form fog colonies, and operate fog devices.

IoT devices are assumed to be using a container-compatible operating system that can be used for the execution of containerized services [103].

Entering a Fog Landscape

To enter the fog landscape, a fog device sends an asynchronous pairing request containing the own location data to the fog controller. The fog controller has a dedicated location service. Based on the location coordinates and coverage areas of all fog nodes in the fog landscape, this location service returns data about the fog node, which becomes a parent to this fog device. This is possible because each fog device contains data about its own device name, IP address, and location coordinates. Other data differs according to the fog device type, for example, fog nodes additionally contain a coverage area parameter. The coverage area defines a geographical area for which a fog node is responsible. The criteria of finding a parent fog node could be different, for example, the calculation of the physical distance between fog devices, but also according to the historically recorded efficiency, or the ratio of successful service execution, or latency. For the purposes of FogFrame, we have implemented searching for the closest parent according to the location of the fog device, which enters the fog landscape.

Pairing Request

If the request is satisfied, the fog device sends a pairing request to this fog node. Upon successful pairing, the device is instantiated as a fog cell or as a fog node in the fog colony and is added to the set of children of the head fog node. If the request is not satisfied, we consider two possible outcomes:

- If the fog device is a fog cell, an error message is returned, and
- If the fog device is a fog node, this fog node becomes the head of a new fog colony as it has a unique range of location coordinates, namely, its coverage area.

This workflow is shown in Figure 5.1.

Fog Colony

A new colony is established if a new fog node joins the fog landscape, provided that it is capable of executing sevices by itself. Otherwise, a fog node waits for other fog cells to connect. A pairing request including the fog node's location coordinates and range is sent from the new fog node to the fog controller. The controller maintains global knowledge of addresses as well as corresponding coordinates and ranges of all the fog nodes. If the new fog node is within the

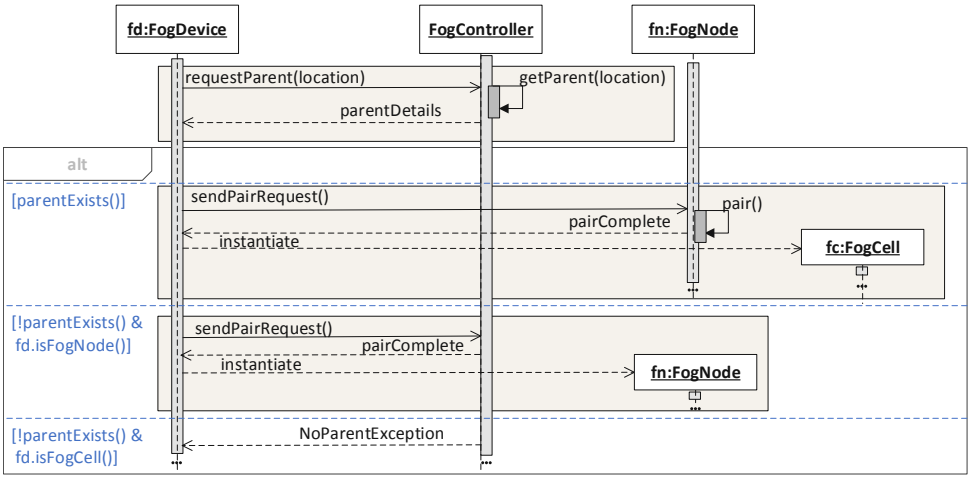

Figure 5.1: Instantiating fog cells and fog nodes in a fog landscape.

coverage area of an existing fog node, the new fog node joins the landscape as a fog node at the lower tier in the hierarchy as depicted in the general overview of the fog in Chapter 3 in Figure 3.1. Otherwise, the joining fog node becomes the head of a new colony. When a fog cell requests to join, the controller responds with the address of the fog node which has a range that includes the coordinates of this cell. If the cell's coordinates are within the coverage area of multiple fog nodes, the controller selects the one closest to the cell based on, e.g., Euclidean distance. Cells are only able to join a landscape if their coordinates are within the coverage of a fog node.

Neighbor Fog Colony

To be able to delegate applications to a neighbor fog colony, each head fog node has to be connected to the head fog node of a neighbor fog colony. This connection is also established when a fog node is added in the fog landscape. It requests the neighbor head fog node from the fog controller. The location service of the fog controller finds the closest neighbor head fog node according to the provided location coordinates. If the request is satisfied, the head fog node sends a pairing request to the closest neighbor head fog node. If the request is not satisfied, the head fog node either connects with a fallback neighbor fog colony, or continues to act autonomously. The head fog node of each colony maintains the address of the head fog node of one neighbor colony. The neighbor fog colony could also be selected based on different criteria, e.g., QoS statistics implemented in each fog colony and stored in the fog controller. Such statistics may include deployment delays, service execution times, resource capacities etc. In FogFrame, we choose the proximity criterion to choose the closest neighbor fog colony.

Fallback Mechanism

Upon joining a fog colony, it is the goal that a pairing request from a fog cell to the direct parent, which is a fog node, can be satisfied. If a pairing request cannot be satisfied, a fallback mechanism is applied. For that, we enable searching

for a fallback parent or grandparent fog node if the closest fog node or even the fog controller are not available. Fallback details can be either implemented as an internal property of a device, or can be sent to the device upon pairing. In FogFrame, fallback parent and grandparent IP addresses are provided as properties of each fog device.

5.3 Maintaining the Fog

A fog landscape is dynamic in its nature, as devices may appear and disappear arbitrarily [64]. Since failures and overloads are inevitable in such systems, the fog landscape needs to refine the network based on periodic and event-based mechanisms at runtime by redeploying running applications. As has already been mentioned in the previous sections, running an application is only possible when all its services are deployed. Hence, if a fog device fails, services which have been running on that fog device need to be redeployed on other fog devices to ensure the application execution. If a new fog cell appears, it is beneficial to use its capacities to release other devices that are not yet overloaded but already close to full capacity, and migrate suitable services from the cloud to reduce additional unnecessary cost of fog landscape operation.

Events To address this, FogFrame can react to certain events:

- Device discovery, if a fog device appears in the fog landscape,
- Device failure, if a fog device is not able to provide services any longer, and
- Device overload, if a fog device is expected to overload, meaning when the CPU, RAM or storage utilization is above a predefined threshold.

In the following, we describe redeployment and placement replanning mechanisms for the runtime events as implemented in FogFrame. Proactivity is ensured by continuously monitoring the fog landscape, while reactivity is implemented with the resource provisioning replanning mechanisms to tackle the runtime events.

Overloaded Devices Resource utilization in all devices in a colony is polled periodically by the head fog node. If a device is identified as overloaded, i.e., it surpasses preconfigured utilization thresholds (related to CPU, RAM and storage capacities), the head fog node redeploys one random service from the overloaded device to another one inside the colony using resource provisioning bounded to the current fog colony. This process continues until resource utilization drops below the threshold. Essentially, this process achieves reactive horizontal autoscaling of applications. Compared to cloud computing, horizontal autoscaling in the fog controls the autoscaling within the head fog node's resource provisioning.

Disconnected Devices All devices in a colony are polled periodically by the head fog node. Devices that do not respond are considered disconnected. When a head fog node discovers a disconnected device, it redeploys the services that were running on the disconnected device (based on the service placement plan)

to other devices in the fog colony using the service placement bounded to the current fog colony (see Section 5.5). Replacing already connected head fog nodes has not been considered. The mechanism could be implemented in the future work to enable a fog node lower in the hierarchy to become the new head fog node in the case when regular polling by the fog controller would show a head fog node in a fog colony as disconnected. If there is no other fog node in the fog colony, the colony fails. If a neighbor fog colony is unresponsive, the head fog node requests the address of a different neighbor fog colony from the fog controller. If the fog controller is unresponsive, fog nodes are also able to accept join-requests (as long as the new device knows the fog node's address) which leads to placing the new device in the lower tier of the hierarchical structure of the colony (see Section 3.2).

New Devices Each time a new fog node or cell joins a colony, it triggers the replanning mechanism executed by the head fog node. More specifically, the head fog node of the colony examines the available resources to identify resource utilization and the number of deployed services in all the devices of the colony. Then, the head fog node opportunistically redeploys services that could be executed in the colony, but are currently running in the cloud, to the new fog cell. Afterwards, services that run on devices operating at maximum capacity are scheduled for redeployment so that some of them will be deployed on the new device. Therefore, through the replanning mechanism, fog colonies react to the increase in the available resources and re-balance the workload to all the devices.

5.4 FogFrame Framework Architecture

In this section, we discuss the high-level architecture of the main components of a fog landscape (see Figure 3.1), namely the fog controller, fog nodes, and fog cells adjusting our previous specification from Section 3.2 to be able to operate in a real-world fog. These components consist of dedicated services and interfaces which provide communication between the components, and within the components as well. The design of the framework is based on lightweight technologies and loosely-coupled components with the goal to create a stable and fault-tolerant distributed system. The extensible modules within each component enable interoperability and a convenient substitution by implementing the specified interface methods.

5.4.1 Fog Cells

Fog cells are software components running on physical IoT devices at the edge of the network, as has been presented in Section 3.2. These cells allow executing IoT services close to the data sources or sinks, instead of involving the cloud.

Components

In the framework, a fog cell consists of a *storage service* which operates a *local storage*, storing connection data, identification data, and application execution

data. The *communication service* is responsible for establishing and maintaining communication with the fog controller and the parent fog node. The *compute unit* executes services and is responsible for data transfer between services. The *fog action control* follows the orders from the fog node of the fog colony, and deploys necessary services by the means of the *container service*. Finally, the *host monitor* monitors the utilization of the fog cell. Figure 5.2 depicts the architecture of fog cells.

Fog Cell APIs Fog cells expose APIs for data transfer and control its actions. The APIs are the same as has been presented in Section 3.2.

5.4.2 Fog Nodes

Components A fog node consists of all the components of a fog cell as well as some additional components. The *shared storage service* operates the *shared storage* which stores a *shared service registry* of all service images to be used for deployment, i.e., it hosts service implementations and enables the fog action control service to search for services and deploy them on the fog cell's compute unit. The local storage on the fog node is similar to the one in the fog cell. However, it stores additional data about fog cells in the fog colony, service placement plans, and execution details of applications. The shared storage service in a fog node is intentionally separated from the local storage container to ensure flexibility and replaceability.

The watchdog observes monitoring data from the host monitor, stores it with the storage service, and compares it to the expected QoS level, i.e., measures the consumption of the cell's computational resources as well as QoS parameters, e.g., the execution time. This information influences the decision-making in the *reasoning service*. The reasoning service is triggered when an application request is submitted for execution to the fog node. The reasoning service calls the *resource provisioning service*, which implements a certain service placement algorithm. The resource provisioning service component produces a service placement plan for the colony's resources to execute received service requests. The service placement plan determines which services should be used to fulfill service requests, and where these services should be deployed, i.e., what resource to use. If there are no sufficient resources in the considered fog cell, or further processing is needed, the whole application request is delegated to the other fog colonies via the head fog node with the communication service.

Events As has already been mentioned above in Section 5.3, FogFrame reacts to runtime events. To handle each of the event types, FogFrame implements corresponding services. To achieve this, *host monitor service* of each fog device records CPU, RAM and storage utilization. Correspondingly, each fog node's watchdog periodically checks if the connected fog devices respond.

Overloads When the CPU power is used up to the maximum level, unexpected performance can take place and compromise the execution of all deployed services. In this

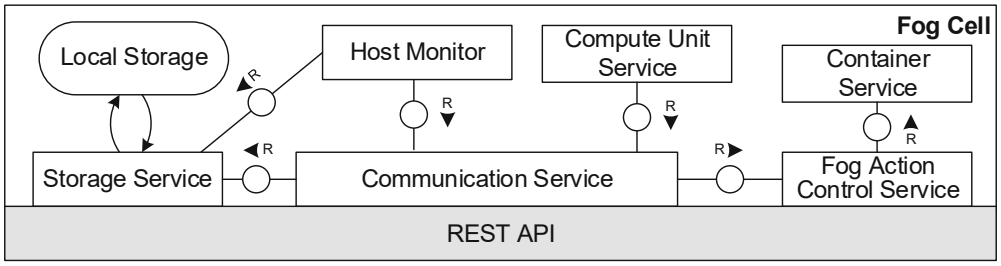

Figure 5.2: Fog cell architecture.

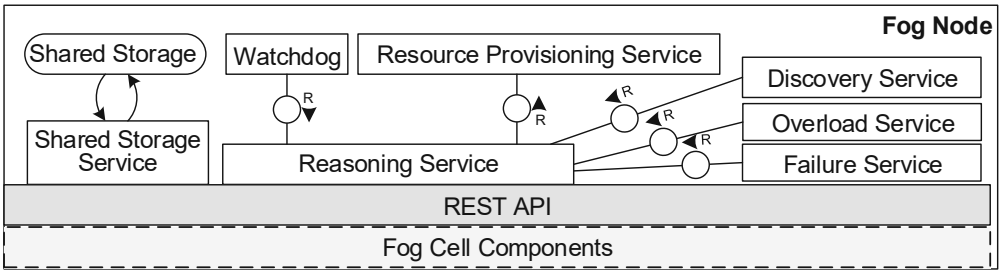

Figure 5.3: Fog node architecture.

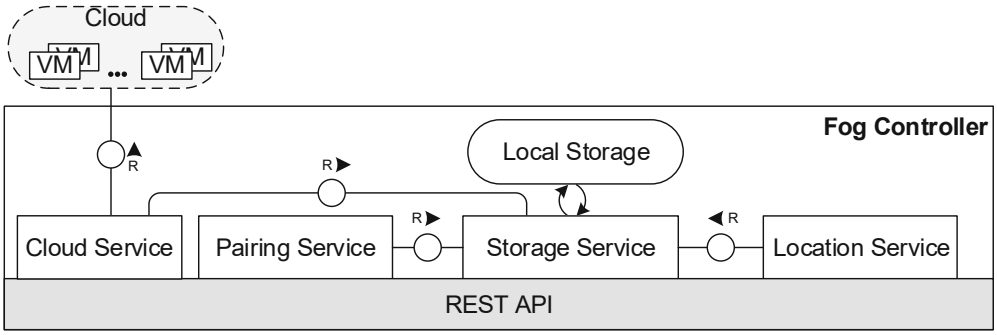

Figure 5.4: Fog controller architecture.

work, we aim for 80% CPU load to get a balance between utilization and room for spikes, ad-hoc processes, and I/O bottlenecks. This threshold was set up during pre-experiments to consider uninterrupted execution and availability of the devices. This threshold is also recommended by AWS[2] and Oracle[3]. This parameter can be easily changed in the FogFrame monitoring component. It is a matter of future research to take into account different thresholds for different

[2]Online; Accessed: Apr. 2023 https://docs.aws.amazon.com/autoscaling/ec2/userguide/as-scaling-simple-step.html

[3]Online; Accessed: Apr. 2023 https://docs.oracle.com/en/cloud/get-started/subscriptions-cloud/mmocs/setting-alert-performance-metric.html

devices according to their capabilities, namely, adaptive thresholds [100].

When a device is identified as overloaded by the head fog node, this event is therefore triggered and the device *overload service* triggers service placement (as presented in Section 5.5) bounded to the current fog colony, and migrates one random container from the affected device. The device overload service migrates services one by one until the affected device is not overloaded anymore. This migration method is similar in cloud computing [132]. The overload policy can be easily reimplemented and substituted with other methods. The framework architecture is loosely coupled and new implementations of used methods can be easily integrated in the future. For instance, a promising approach to predict overload of fog devices presented by [106] could be implemented in future work.

New Devices

If a new fog device joins a fog colony, all its resources are analyzed to identify its current workload. First, if applicable, the discovery service migrates services that were deployed by the fog controller to the cloud to this new device to release cloud resources and save cost. Second, all devices that operate at the maximum capacity and are overloaded, are considered for migration. The *discovery service* triggers the placement of suitable services from overloaded devices and migrates them to the new device either while needed, or until it is filled to a maximum defined number of containers. The number of containers can be adjusted according to the computational capacity of each fog device. This mechanism enables horizontal scalability of the resources in the fog landscape.

It has to be noted that the vertical scalability of resources is bounded by the computational capacities of devices in a fog landscape: If it is possible to extend CPU or RAM on a device, then FogFrame will accordingly use those capacities. This is enabled by the implemented host monitor service.

Disconnected Devices

When a fog cell disappears, i.e., disconnects, from a fog colony due to a failure, the head fog node checks the service assignments to identify whether the failed fog device had any services running. The *device failure service* adds all services that were running on that fog device to a migration list, and triggers service placement bounded to the current fog colony. Failures of fog cells are identified by a parent fog node when the fog cell is disconnected. Failures of fog cells trigger the *migration service*. Because fog nodes take care of communication between their connected fog cells and other fog nodes higher in the fog colony's hierarchy, a failure of such an intermediary fog node not only triggers the migration service to recover all running services, but also requires fog cells to ask for a new parent fog node from the fog controller. In the case when the head fog node of the fog colony fails, then either there is a mechanism in place to connect all the fog colonies resources to a fallback head fog node, or to trigger a complete reorganization of the fog colonies. The latter is a topic for future research, that can be formulated as a meta control mechanism for reactive and proactive reorganization and optimization of fog colonies to ensure system durability and fault tolerance.

It should be noted that an alternative approach would be a decentralized orchestration of fog cells in a fog colony, i.e., without the head fog node. This possible alternative is presented later in the opportunities for future work in Section 7.2. While this leads to higher fault tolerance, it also involves extensive coordination and voting between the involved fog cells. Therefore, we opt for having a head fog node per fog colony and a hierarchy of fog devices. However, we still foresee that another fog node becomes the head fog node in a fog colony, if necessary, e.g., in case the original head fog node fails.

5.4.3 Fog Controller

Fog Controller Components

In the fog controller (see Figure 5.4), the *cloud service* establishes the communication with the cloud and implements necessary functionalities to manage VMs and containers. The *location service* provides connection data for fog devices entering the fog landscape. The *pairing service* is responsible for pairing of fog devices as described in Section 5.2. A *local storage* stores data about the structure of the fog landscape and the usage of cloud resources. The local storage is operated by the *storage service*.

5.5 Application Management

Definitions

To achieve the cooperative execution of IoT applications, the framework enables decentralization of application execution, making it possible that different parts of an IoT application are deployed and executed close to the relevant data sources and data sinks. In Section 3.1, we have provided an overview of deployment models. Because of the benefits of containerized applications mentioned in Section 2.4, in FogFrame the applications are built following the microservice architectural approach and are made of stateless services deployed and executed on fog cells. Examples of such applications include MapReduce applications and stream processing applications [36, 71, 72]. The assumption is that an application can only be executed if all its services are deployed. As mentioned in Section 2.5, an application can be visualized as a DDF [64], which is a directed acyclic graph where vertices denote tasks to be executed in the flow corresponding to services, and edges between vertices are data shipments or control flow connections between those services [91] (see Figure 2.4).

A service is a deployed and running computational software instance which processes a service request in a fog landscape, as has been already mentioned in the architecture discussion in Section 5.4. A service request is a single computational job to be computed on fog devices. A service image is a not yet deployed service binary. Services can be of certain service types. Service types are bound to the capabilities of the devices in the fog landscape. For example, a service intended to receive temperature measurements can be deployed only on a fog cell with a

temperature sensor attached, some services can be executed either in the cloud or in the fog, and other services can be executed only in the cloud.

Application Execution

The application execution starts with an application request which defines a set of services to be placed and deployed in the fog landscape together with QoS information for execution, for example, deadlines on application execution and processing times. It has to be noted that an application can only be executed if all its services are deployed. The deployment of services depends on the service placement mechanism which is applied within each head fog node of each fog colony in the fog landscape. It is possible to integrate arbitrary service placement algorithms into FogFrame. Within this thesis, particular approaches are introduced aiming at utilizing available resources of fog colonies in the most efficient way, as presented in Section 5.6.

In the following, we describe how the application is processed on different resources in the fog landscape: inside a fog colony, in the cloud, or after being delegated to a neighbor fog colony (see Figure 5.5).

Application Deployment

Application deployment and execution can be done based on different settings. The first (and simplest) setting is when each fog colony has enough own resources to execute an application request. In this case, all the necessary services are deployed in the current fog colony. The latency and deployment time are minimal and depend on the computational power of the resources of the colony. As it can be seen in Figure 5.5 (a), a user submits an asynchronous application request to a fog node. The service placement is performed according to the chosen service placement algorithm by the reasoning service of the fog node. It has to be noted that an application request can also be submitted to a fog cell (not depicted in Figure 5.5). If this is the case, the fog cell forwards the request to its parent fog node until the request reaches the head fog node of the corresponding fog colony, which performs service placement.

Service Placement

Service placement is performed in a decentralized manner and is independent in each fog colony. To perform computations, the reasoning service uses information about the availability and utilization of all the fog cells in the fog colony. It applies this information within the implemented resource provisioning and service placements algorithms. The result of the calculations in the algorithms is a service placement plan. After the service placement plan is calculated, the head fog node deploys the necessary services on identified fog cells, and the fog cells immediately start service execution.

Cloud Resources

The second setting of application execution is when apart from executing services in a fog colony, it is necessary to support fog colonies with additional resources from the cloud (Figure 5.5 (a)-(b)). This applies for services which can be executed only in the cloud, for example, big data processing, and those services which cannot be placed on fog devices because of QoS constraints or a lack of resource capabilities, or services which can be executed either on the edge devices

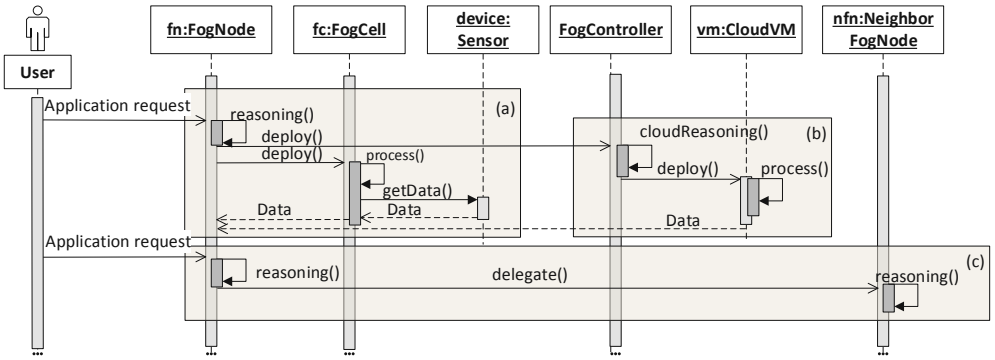

Figure 5.5: Application request processing on different fog resources.

or in the cloud as they do not require specific sensor equipment. If specific services in the application request are assigned to the cloud, the fog node sends the request to execute this service in the cloud. For this, the fog controller authenticates itself with the cloud provider, either leases and instantiates a new VM in the cloud or connects to an existing VM, and deploys the corresponding service container.

Delegation of Applications

The third setting of application execution regards if an application cannot be executed by a fog colony. However, this application requires sensor equipment, therefore it cannot be executed in the cloud. In this case, the request is delegated to a neighbor fog colony. This is the case when the service placement plan determines that there are not enough resource capacities in the current fog colony to execute the application, however there is enough time indicated by the application deadline to postpone the execution. Therefore, the fog node delegates the whole application request to the neighbor fog colony (Figure 5.5 (c)). In Section 4.3, we have already introduced the process of delegation of workload to the neighbor fog colonies. However, in Section 4.5 and Section 4.7 we use simulation environments, while in the framework we aim to execute IoT applications in the real-world fog. First, the delegation of the whole application with all its services to the neighbor colony instead of taking single services allows avoiding high intra-application latency, as fog colonies can be geo-distributed and have different capabilities or domains. Second, it allows avoiding additional coordination between fog colonies on the level of application requests. Detaching single services and sending them for execution in other colonies introduce the new problem of tracking of application execution spanning in multiple fog colonies in a fog landscape.

5.6 Resource Provisioning in FogFrame

Knapsack and Assignment Problem

FogFrame enables decentralized service placement, which is ensured by the reasoning and placement capabilities of each head fog node in fog colonies, as

has already been applied in Chapter 3 and Chapter 4. However, compared to the previous chapters, the resource provisioning in FogFrame accounts for the real-world fog. Therefore the new resource provisioning model has introduced additional constraints, as well as it enables delegation of the whole IoT application requests in the case when a fog colony, to which the request has been initially submitted, is busy and cannot provide necessary capacities of resources for deployment. Fog colonies autonomously perform service placement, even in the case when the fog controller or cloud-based computational resources are not available. The underlying service placement model, which is discussed in detail in this chapter, determines an optimal mapping between services of IoT applications and computational resources in the fog, i.e., at the edge of the network and in the cloud (if available). The resource provisioning and service placement problem has been shown to be NP-hard [7]. For this problem, as described in Section 2.5, an analogy towards the multiple knapsack problem and the assignment problem can be performed [10]: different fog resources are knapsacks, single services of IoT applications are items to be inserted into knapsacks, the weight of knapsacks corresponds to available resources of fog devices, such as CPU, RAM and storage, and the costs of the knapsack are the defined QoS parameters. Services need to be assigned only once. The complexity of a multiple knapsack problem is proven to be $O(n^2 + nm)$ [49], where n is the number of services to be placed and m is the number of fog resources available in a fog colony. For service placement, the objective function is to maximize the utilization of devices at the edge of the network while satisfying the QoS requirements of applications, namely, satisfying deadlines on application deployment and execution time.

Service Placement Requirements

For this, every head fog node considers resources available in its fog colony, cloud resources, and the closest neighbor fog colony. A mapping between applications and computational resources determines the following subsets of service placement:

- Services to be executed on fog devices in the fog colony,
- Services to be executed locally on the fog node, and
- Services to be executed in the cloud.

If at least one service of the application cannot be placed in those subsets, the whole application request is sent to the closest neighbor fog colony. Splitting single services from applications and delegating them to the neighbor fog colony is not considered in order to eliminate tracking of single services in the fog landscape and high intra-application latency and because of the necessary coordination between fog colonies. The service placement approach is reactive, namely, whenever application requests are submitted for execution to a fog node, the service placement algorithm is triggered. Additionally, service placement is triggered each time operational runtime events happen in the fog landscape: appearing and disappearing of resources at the edge, failures and overloads. In the following, we formalize the corresponding system model. Table 5.1 gives an overview of the notation of fog resources and applications.

Table 5.1: Notation of fog resources and applications.

	Parameter	Description
Time	t	Current placement time
	τ	Time difference of previous placement (in sec)
Applications	A	Set of applications to be executed
	A_k	Application
	D_{A_k}	Deadline of A_k
	w_{A_k}	Deployment time of A_k
	$w^t_{A_k}$	Already passed deployment time of A_k at t
	$\overline{T}^t_{w_N}$	Deployment time in the neighbor colony
	m_{A_k}	Makespan duration of A_k
	r_{A_k}	Response time of A_k
	$\|A_k\|$	Number of services in A_k
	a_i	Service in an application A_k
	U_{a_i}	CPU demand of service a_i
	M_{a_i}	RAM demand of service a_i
	S_{a_i}	Storage demand of service a_i
	m_{a_i}	Makespan duration of service a_i
	$Res_{a_i}(F)$	Fog cells able to host service a_i
Fog Landscape	R	Cloud
	F	Head fog node
	N	Closest neighbor to F
	$Res(F)$	Fog cells connected to F
	$C = \{U, M, S\}$	Resource capacities of fog devices
	U_F	CPU capacity of F
	M_F	RAM capacity of F
	S_F	Storage capacity of F
	K_F	Container capacity of F
	f_j	Fog cell
	U_{f_j}	CPU capacity of f_j
	M_{f_j}	RAM capacity of f_j
	S_{f_j}	Storage capacity of f_j
	K_{f_j}	Container capacity of f_j
	d_{f_j}	Latency between F and f_j
	d_R	Latency between F and R
	d_N	Latency between F and N

5.6.1 Adapted Definition of the FSPP

Domain Definition The decision variables $x^{f_j}_{a_i}$, $x^F_{a_i}$, $x^R_{a_i}$ indicate the placement of a service a_i on a specific resource in a fog landscape, namely, on a fog cell f_j, fog node F, or in the cloud R. The decision variable y_{A_k} indicates that the request

for the execution of the application A_k has to be delegated to the closest neighbor fog colony, with the head fog node N. Together, the decision variables form a service placement plan. In Section 4.3, we used other variables for delegation, i.e., of single services. As discussed in Section 5.5, in this adapted definition of the FSPP, which is aiming at IoT application execution in a real-world fog, we delegate the whole application request indicated by the decision variable y_{A_k} in the case when it is not enough resources to execute it in the current fog colony with the head fog node F.

Variables Let a_i denote a service of the application A_k. In order to ensure that a service is compatible with an allocated resource, $Res_{a_i}(F)$ is introduced to denote all the fog cells capable to run service a_i, with $Res_{a_i}(F) \subseteq Res(F)$. This formalism is necessary to account for service types, with which resources can be compatible with, for example, sensing or processing service types. The decision variables of the service placement problem are provided in equations (5.1)–(5.4):

$$x_{a_i}^{f_j} \in \{0,1\},\ \forall a_i \in A_k,\ \forall f_j \in Res_{a_i}(F) \tag{5.1}$$

$$x_{a_i}^{F} \in \{0,1\},\ \forall a_i \in A_k \tag{5.2}$$

$$x_{a_i}^{R} \in \{0,1\},\ \forall a_i \in A_k \tag{5.3}$$

$$y_{A_k} \in \{0,1\} \tag{5.4}$$

Objective Function The objective function of the service placement is to maximize the number of service placements in the available fog colonies, while satisfying the QoS requirements of applications, as defined in (5.5).

$$\max \sum_{A_k}^{A} P(A_k) N(A_k) \tag{5.5}$$

Unlike execution of all services within one fog colony, delegation of the application to the closest neighbor fog colony or execution in the cloud suggests additional delays, which can become a serious constraint when the response time of the application is close to its declared deadline. Hence, we use the prioritization coefficient $P(A_k)$ for each application. The coefficient $P(A_k)$ represents the weight of an application A_k determined by the difference between the deadline D_{A_k} of the application and its already recorded deployment time w_{A_k}, as defined in (5.6).

$$P(A_k) = \frac{1}{D_{A_k} - w_{A_k}} \tag{5.6}$$

w_{A_k} appears in the cases when the application was propagated from another fog colony and had to wait until it is correctly placed on necessary resources.

The priority for deployment is given to the applications with high w_{A_k}, and accordingly little difference $D_{A_k} - w_{A_k}$. $N(A_k)$ denotes the number of services in the application request to be placed in fog colonies, as defined in (5.7).

$$N(A_k) = \sum_{a_i}^{A_k} \left(\sum_{f_j}^{Res_{a_i}(F)} x_{a_i}^{f_j} + x_{a_i}^{F} \right) + |A_k| y_{A_k} \tag{5.7}$$

Constraint on Resources

Constraints The first set of constraints defines the usage of available CPU, RAM, and storage of fog resources (U_{a_i}, M_{a_i}, and S_{a_i}, respectively). The sum load of placed services should be within the tolerance limits of resource capacities of corresponding fog devices, as shown in (5.8)–(5.10). The tolerance limit of a resource is $\gamma \in [0, 1]$, for example, $\gamma = 0.8$ indicates that 80% of all device resources can be used to execute services and the rest 20% should be kept free in order to account for operational stability of the device. As described in Section 5.3, it would also be possible to utilize an approach based on adaptive thresholds. $C = \{U, M, S\}$ denotes the corresponding capacities of CPU, RAM, and storage of fog cells and the fog node, and the corresponding CPU, RAM, and storage demands of services (see Table 5.1).

$$\sum_{A_k}^{A} \sum_{a_i}^{A_k} C_{a_i} x_{a_i}^{f_j} \leq \gamma C_{f_j} \ , \forall f_j \in Res_{a_i}(F) \tag{5.8}$$

$$\sum_{A_k}^{A} \sum_{a_i}^{A_k} C_{a_i} x_{a_i}^{F} \leq \gamma C_F \tag{5.9}$$

$$C = \{U, M, S\} \tag{5.10}$$

Response Time Calculation

To execute applications according to the necessary QoS, the response time r_{A_k} of each application A_k has to be less than the declared deadline D_{A_k} of the application, as defined in (5.11). The response time r_{A_k} is defined as the sum of the total makespan duration m_{A_k} and its deployment time w_{A_k}, as defined in (5.12).

$$r_{A_k} \leq D_{A_k}, \forall A_k \in A \tag{5.11}$$

$$r_{A_k} = m_{A_k} + w_{A_k} \tag{5.12}$$

Makespan Calculation

The total makespan duration m_{A_k} consists of execution times of all single services of the application A_k accounting for the communication delays between the head fog node and chosen for placement resources, as defined in (5.13)–(5.14).

$$m_{A_k} = \sum_{a_i}^{A_k} L_{a_i} \tag{5.13}$$

$$L_{a_i} = \sum_{f_j}^{Res_{a_i}(F)} d(a_i, f_j) x_{a_i}^{f_j} + d(a_i, F) x_{a_i}^{F} + d(a_i, R) x_{a_i}^{R} \tag{5.14}$$

$d(a_i, f_j)$, $d(a_i, F)$, and $d(a_i, R)$ denote the makespan duration m_{a_i} of a service a_i in the case of its placement and execution on fog cell f_j, fog node F, or in cloud R respectively (5.15)-(5.17).

$$d(a_i, f_j) = d_{f_j} + m_{a_i} \tag{5.15}$$
$$d(a_i, F) = m_{a_i} \tag{5.16}$$
$$d(a_i, R) = 2d_R + m_{a_i} \tag{5.17}$$

Example

Let the application $A_1 = \{a_1, a_2, a_3, a_4\}$ be distributed between a fog colony and the cloud (see Figure 5.6). To find the response time of an application r_1, the makespan of each service m_{a_i} is added to the delay in (5.18). In this example, we assume that the application had no previous deployment time, so that $w_{A_1} = 0$.

$$r_1 = d(f_{11}) + m_{a1} + m_{a2} + 2d(R) + m_{a3} + d(f_{12}) + m_{a4} \tag{5.18}$$

Application Deployment Time

The application deployment time w_{A_k} from equation (5.12) accounts for the time spent by the application before all its services are placed on the chosen resources. In the case when one of the services $a_i \in A_k$ cannot be placed in the current fog colony, the whole application needs to be delegated to the neighbor fog colony. Therefore, an additional expected deployment time $\overline{T}^t_{w_N}$ can appear. To define whether this additional deployment time affects w_{A_k} or not, we introduce the auxiliary variable y_{A_k}. Let $y_{A_k} = 0$ if all services can be successfully placed in the current fog colony, and $y_{A_k} = 1$ if at least one service $a_i \in A_k$ cannot be placed in the current fog colony, and the whole application needs to be delegated. Therefore, the application deployment time w_{A_k} is defined as follows:

$$w_{A_k} = w^t_{A_k} + \overline{T}^t_{w_N} y_{A_k} \tag{5.19}$$

Deployment Time in the Neighbor Colony

$\overline{T}^t_{w_N}$ affects the application deployment time only if $y_{A_k} = 1$. The closest neighbor fog node applies its own service placement which can either place all services $a_i \in A_k$ in its fog colony or further postpone the execution of the application

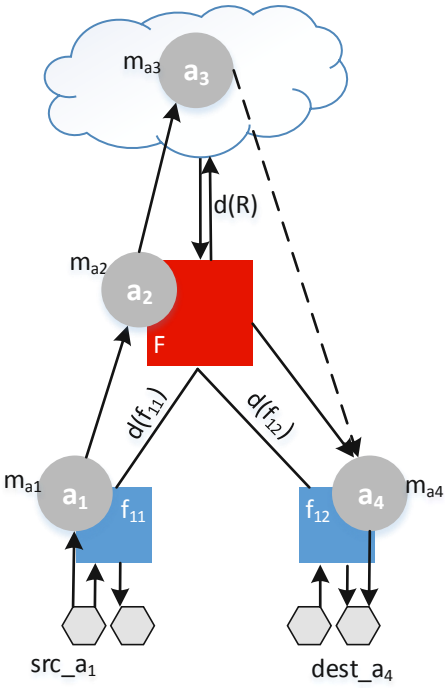

Figure 5.6: Example of response time calculation.

A_k by delegating it further in the fog landscape. To avoid a ping-pong with an application between the two closest fog colonies, additional criteria can be considered, for example, the rate of successful service execution, free capacities, or a percentage of load on a fog colony. In the current work, if the application cannot be deployed in a fog colony or in a neighbor fog colony before a stated deadline, it is either deployed in the cloud if the service types permits that, or is not deployed at all. To calculate the expected deployment time in the closest neighbor colony, $\overline{T}^{t}_{w_N}$ requires to view forward in time. Therefore, we estimate $\overline{T}^{t}_{w_N}$ relying on historical data. $\overline{T}^{t}_{w_N}$ is obtained as the moving average on the latest sampled deployment time $T^{t-\tau}_{w_N}$ per service delegated to the closest neighbor fog colony as defined in (5.20), where $1-\alpha$, $\alpha \in [0,1]$ denotes the discount factor of the moving average, $T^{t-\tau}_{w_N}$ is the already passed deployment time of the service delegated to the neighbor fog node N during the time $t-\tau$, and $\overline{T}^{t-\tau}_{w_N}$ stands for the average deployment time in N as estimated in $t-\tau$.

$$\overline{T}^{t}_{w_N} = \alpha T^{t-\tau}_{w_N} + (1-\alpha)\overline{T}^{t-\tau}_{w_N} \tag{5.20}$$

Container Capacity

Next, the container capacity meaning the number of deployed containers, should not be more than K_{f_j} containers for each of the fog cells and K_F for fog nodes as defined in (5.21) and (5.22), because that may cause overload.

$$\sum_{A_k}^{A}\sum_{a_i}^{A_k} x_{a_i}^{f_j} \leq \gamma K_{f_j} \ , \forall f_j \in Res_{a_i}(F) \tag{5.21}$$

$$\sum_{A_k}^{A}\sum_{a_i}^{A_k} x_{a_i}^{F} \leq \gamma K_{F} \tag{5.22}$$

Number of Assigned Services

We have to provide the condition that in the case when any of the services in an application cannot be placed in the current fog colony, the application request has to be sent to the neighbor colony. For that, we first calculate the number of services placed in the current fog colony and in the cloud:

$$n_{A_k} = \sum_{a_i}^{A_k} \left(\sum_{f_j}^{Res_{a_i}(F)} x_{a_i}^{f_j} + x_{a_i}^{F} + x_{a_i}^{R} \right), \forall A_k \in A \tag{5.23}$$

Delegation to the Neighbor Colony

Next, if the total number of services placed in the current colony and in the cloud is less than the total number of services in the application, that means if $n_{A_k} < |A_k|$, then the application request has to be sent to the closest neighbor colony, namely, $y_{A_k} = 1$, else $y_{A_k} = 0$. This conditional constraint is formulated using big-M coefficients [78], and is represented in (5.24) and (5.25), respectively:

$$n_{A_k} - |A_k| \leq M(1 - y_{A_k}) - 1, \forall A_k \in A \tag{5.24}$$

$$|A_k| - n_{A_k} \leq M y_{A_k} - 1, \forall A_k \in A \tag{5.25}$$

Single Assignment Constraint

Finally, we define that each service a_i can be placed on exactly one computational resource f_j, F, N, or in the cloud R, or the whole application request has to be sent to the closest neighbor fog colony:

$$n_{A_k} + y_{A_k} = 1, \forall A_k \in A \tag{5.26}$$

Estimation of Variables and Constraints

Example To provide an estimation of the number of variables and constraints in the service placement problem, we consider the following example: An application A_1 with ten services $a_1 ... a_{10}$ is submitted for execution to a fog colony with a head fog node F and two fog cells f_1 and f_2. In this example, the assumption is that both fog cells f_1 and f_2 and the head fog node F are able to execute all service types to which the services belong. For this setting, a service placement problem consists of making a decision for in total 41 decision variables, which are:

- Ten decision variables $x^{f_1}_{a_1...a_{10}}$ corresponding to the placement decision of each of the services $a_1...a_{10}$ on fog cell f_1 and accordingly another ten decision variables $x^{f_2}_{a_1...a_{10}}$ of each service $a_1...a_{10}$ on fog cell f_2.
- Ten decision variables $x^{F}_{a_1...a_{10}}$ corresponding to the placement of each service $a_1...a_{10}$ placed on head fog node F.
- Ten decision variables $x^{R}_{a_1...a_{10}}$ corresponding to a propagation decision of each service $a_1...a_{10}$ to the cloud resources R.
- One decision variable y_{A_1} denoting whether the whole application needs to be delegated to the neighbor fog colony.

With regard to the constraints, the first set of constraints deal with CPU, RAM and storage capacities of each device according to (5.8)-(5.10), meaning three constraints for each computation device f_1, f_2, and F. Second, the response time constraint is only one according to the calculations (5.11)-(5.20) corresponding to one application A_1. Next, there are three container capacity constraints according to (5.21)-(5.22) per each f_1, f_2, and F. Afterwards, to calculate whether the application A_1 can be placed in the current fog colony or needs to be delegated to the neighbor fog colony, two additional constraints per application, that is in our case only one application, need to be calculated according to (5.23)-(5.25). And the last set of constraints ensures that each service can be placed only on one resource, that means one constraint according to (5.26). To summarize, the number of variables and constraints depends on the number of resources in the fog landscape, the number of applications requested for execution, and the number of services to be deployed in each application.

5.6.2 Heuristic Algorithms Implementation

Greedy Algorithm

Overview

As mentioned above, we provide two service placement algorithms as examples in FogFrame. The greedy algorithm implemented in the framework (see Algorithm 5.1) is based on finding a first-fit device for each service request [155]. The idea is to walk over sorted fog devices according to their service types, available resource capacities, and incoming service requests, and check whether a fog device is able to host and deploy a service according to the device's utilization. If any service in the application cannot be deployed, the whole application is delegated to the neighbor colony. The main benefit of greedy algorithms is that they produce fast and feasible solutions [155].

Initialization

The algorithm takes the set of fog devices in the fog colony and the incoming service requests as inputs. Line 1 of Algorithm 5.1 initializes an empty assignments list and a new counter parameter. This counter enables several tries to place a service in the algorithm. This counter is necessary to account for released resources in the case when the execution of already deployed services has been finished at the time of service placement.

Sorting of Resources and Services

Lines 2 and 3 sort corresponding sets by service type in order to assign sensor-related services to fog cells with the highest priority because sensor equipment is available only there. For example, a service sensing temperature can be placed only on the device with a temperature sensor attached. If any service can be executed both on a fog device and in the cloud, the service is assigned to the fog device if available.

Attempt to Place Services

Lines 4 to 6 start the loops over the sorted fog devices and service requests. These loops ensure the placement of equipment-specific services on fog cells first, and if they can be placed, an attempt to place all other services is performed. Otherwise, the whole application has to be delegated to the neighbor fog colony.

Check Constraints

Line 7 checks if the service type of the service request corresponds to the service types of the fog device. This is necessary because the constraint on matching service types of a service and device is the main constraint in the service placement, otherwise the assignment is not possible. In Lines 8 and 9, the utilization and the number of already deployed containers is requested from the fog device in order to check if there are enough resources available. This is done in order to check the utilization constraints in Line 10, where the utilization parameters and the number of deployed services are compared to predefined monitoring rules, for example, CPU utilization $< 80\%$.

Deployment Request

If those parameters are satisfied, the fog device is able to host a service, and a deployment request is sent to the fog device in Line 11. In the event of successful service deployment, the fog device sends the detailed information about the deployed container to the fog node. In Line 12, an assignment consisting of the fog device, service request, and the identifier of the deployed container is created. With this assignment, it becomes possible to keep track of all the deployed services and corresponding containers in a fog colony. A successfully executed service request is removed from the input set in Line 13 making sure the service is deployed exactly once.

Stopping Condition

Line 18 checks if the outer-most fog device cycle is finished, the provisioning round counter is smaller than the maximal defined number of tries, and if there are still opened service requests. If this is the case, the round counter is increased and the fog device iterator is re-initialized to restart the provisioning with the remaining service requests (Lines 19 and 20). After all the requests are handled or the maximum number of provisioning rounds is exceeded, the created assignments and the open requests are returned (Lines 23 and 24). If the open requests list is not empty, then the corresponding application requests are sent to the neighbor fog colony and already deployed services from those applications are stopped.

Genetic Algorithm

We have presented a GA in Section 4.6. In this section we adapt this algorithm and introduce additional constraints in the FSPP formulation in FogFrame.

Algorithm 5.1: Greedy algorithm.

```
Input  : Set<Fogdevice> fogDevices, Set<TaskRequest> requests
Output : List<TaskAssignment> assignments, List<TaskRequest>
         openRequests
assignments ← []; round = 0;
sortedRequests ← sortByServiceType(requests);
sortedFogDevices ← sortByServiceType(fogDevices);
for fogDevice ∈ sortedFogDevices do
    for serviceType ∈ fogDevice.serviceTypes do
        for request ∈ sortedRequests do
            if serviceType == request.serviceType then
                utilization ← getUtilization(fogDevice);
                containers ← getContainerCount(fogDevice);
                if checkRules(utilization) & containers <
                MAX_CONTAINERS then
                    container ←
                    sendDeploymentRequest(fogDevice, request);
                    assignments.add(fogDevice, request, container);
                    sortedRequests.remove(request);
                end
            end
        end
    end
    if !sortedFogDevices.hasNext() & round < ROUNDS &
    sortedRequests.size() > 0 then
        round = round + 1;
        sortedFogDevices.reStart();
    end
end
openRequests ←sortedRequests;
return assignments, openRequests
```

Solution Space

As has been discussed in Section 4.6, GAs browse a large search space and provide a viable qualitative solution in polynomial time [159, 163, 164]. One iteration of a GA applies the genetic operators of selection, crossover, and mutation on a generation of solutions of an optimization problem [149]. In our case, the optimization problem is to make a decision about service placement on fog resources, i.e., to produce a service placement plan.

Operators

A generation consists of individuals represented by their chromosomes. Each chromosome denotes one solution to the considered problem, in our case a chromosome is one service placement plan. The GA starts with the process

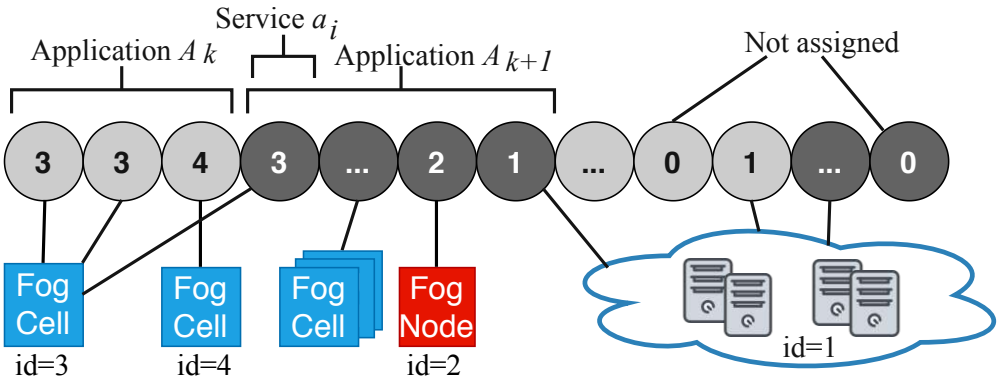

Figure 5.7: Chromosome representation.

of selection of individuals for reproduction. For that, the fitness function of each chromosome is calculated based on the goal function and constraints of the service placement problem, and the best chromosomes are selected. To create an offspring from the selected individuals, the crossover operator swaps genes of each two chromosomes. In order not to lose the best-performing candidate solutions, some individuals with the highest fitness values of their chromosomes are forwarded to the next generation unaltered. They do not participate in crossover, becoming the elite individuals. After that in order to ensure diversity of the population, the mutation operator changes a random number of genes in some of the chromosomes. As a result, a new generation is evolved consisting from the elite and offspring individuals. The algorithm repeats this process until activation of defined stopping conditions. Different approaches can be applied as stopping conditions, for example, the total number of evolved generations, a tolerance value of the fitness function, or elapsed time. In the following, we describe the concrete implementation of the GA in FogFrame.

Chromosome Representation

The chromosome representation is a vector corresponding to a service placement plan (see Figure 5.7). The length of this vector equals to the total number of services in the requested application. Each gene in a chromosome is an integer number corresponding to the identifier of a fog device or the cloud. When a service cannot be placed at any of the devices in the fog colony or in the cloud, it remains unassigned. In this case, the whole corresponding application is delegated to the closest neighbor colony. This chromosome representation ensures the placement of all services, meaning that there are no invalid chromosomes. Additionally, the chromosome representation stores necessary data to estimate utilization of devices that includes CPU, RAM, and storage resources of fog devices, and estimated response time of the application.

Fitness Function

In the fitness function, we encourage the chromosome if it fulfills the constraints of the system model presented in Section 5.6 and apply penalties if the constraints

are violated [109]. The constraints of the optimization problem have been divided into three sets which affect the fitness function to different degrees:

- A set Ψ of constraints on capacities of CPU, RAM and storage resources of fog devices,
- A set Γ of implicit binary constraints derived from the goal function: conformance to service types, indications if cloud or fog colony resources have to be used, and prioritization of own fog colony resources, and
- A set Υ causing the 'death' penalty of the chromosome if the service types, container capacities in devices, or deadlines are violated.

Constraints $\forall\beta_p \in \Psi$

Let c denote a chromosome. For constraints $\forall\beta_p \in \Psi$, if $\beta_p(c) \leq 0$, the constraints are satisfied. If $\beta_p(c) > 0$, then the constraints are not satisfied. These conditions are formalized in (5.27).

$$\delta_{\beta_p(c)} = \begin{cases} 0, & \text{if } \beta_p(c) \leq 0 \\ 1, & \text{if } \beta_p(c) > 0 \end{cases} \tag{5.27}$$

Constraints $\forall\beta_\gamma \in \Gamma$

Similarly, for the Γ set of constraints, if $\beta_\gamma(c) = 0$, then the constraints are satisfied. If $\beta_\gamma(c) = 1$, the constraints are not satisfied.

Constraints $\forall\beta_\upsilon \in \Upsilon$

For the Υ constraints, the penalty distance from the satisfaction of Υ constraints for c is defined in (5.28), where β_υ denotes a constraint, and $\delta_{\beta_\upsilon(c)}$ indicates whether a constraint has been violated in the current chromosome c: $\delta_{\beta_\upsilon(c)} = 1$.

$$D(c) = \sum_{\beta_\upsilon \in \Upsilon} \delta_{\beta_\upsilon(c)} \tag{5.28}$$

Fitness Function

The fitness function is calculated according to (5.29), where $\omega_{\beta_p(c)}$ is the weight factor of $\beta_p \in \Psi$, $\omega_{\beta_\gamma(c)}$ is the weight factor of $\beta_\gamma \in \Gamma$, and ω_p is the penalty weight factor for constraints in Υ. If constraints β_p or β_γ are satisfied in c, then $\delta_{\beta_p(c)}$ and $\delta_{\beta_\gamma(c)}$ become 0, and the corresponding values within the first and the second terms of (5.29) 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 corresponding values resulting from the first and second terms are subtracted from the fitness function. The third term in the fitness function ensures death penalty $\omega_p D(c)$ for having $D(c)$ other than 0, where the penalty factor ω_p has to be big enough to forbid participation of the worst chromosomes to perform crossover and to create the next generation of individuals. The weight values allow changing the impact of constraints on the fitness function. In this work, weights equal to 1, and the death penalty weight equals to 100000. When the GA is running, the fitness value of chromosomes increases. This happens because less penalties are applied to the chromosomes [109].

$$F(c) = \sum_{\beta_p \in \Psi} \omega_{\beta_p}(1 - 2\delta_{\beta_p(c)}) + \sum_{\beta_\gamma \in \Gamma} \omega_{\beta_\gamma}(1 - 2\delta_{\beta_\gamma(c)}) - \omega_p D(c) \tag{5.29}$$

Values of Genetic Operators

The genetic operators were determined based on pre-experiments presented in Section 4.6: We use a 80%-uniform crossover because the genes are integer values, a crossover mixing ratio of 0.5, tournament selection with the arity 2, random gene mutation with a 2% mutation rate, a 20% elitism rate, and a population size of 1000 individuals.

Stopping Condition

Regarding the stopping condition, different options exist. Obviously, the fitness value of the fittest individual in the generation has to be a positive number, since a positive fitness value means that there are no death penalties applied to the individual. Time-based stopping conditions of a number of iterations or execution time of the algorithm are not clear to define [23]. A stopping condition based on improving the variance of the fitness function over generations is identified as assuring the algorithm's convergence.

Fitness Tolerance

We use a tolerance value of the fitness function as the stopping condition of the algorithm. It is calculated by dividing the incremental variance of the fitness function values by the maximum fitness value over generations [23]. The tolerance value of the fitness function is set to $\epsilon = 0.01$, which is enough to obtain the solution and not to converge in local maxima.

With either of the two service placement algorithms presented in Section 5.6.2 in place, we are now able to compute a service placement plan in fog nodes in FogFrame.

5.7 Testbed Implementation

Raspberry Pi Computers

In our evaluation, Raspberry Pi computers are used as fog devices. Raspberry Pis are based on an ARM processor architecture, which is also used in mobile phones, smart phones, and digital television, to name just some examples. In general, nearly 60% of all mobile devices use ARM chips [160]. Therefore, Raspberry Pis can be considered as representative when building a fog landscape [83]. Fog nodes and fog cells are deployed on Raspberry Pi 3b+ units (Quadcore 64-bit ARM, 1GB of RAM), which run the Hypriot operating system (see Figure 5.8 for an example of testbed organization). The detailed setup configuration of FogFrame is described in [14]. The fog controller is deployed with a Docker container on an Ubuntu 18.04LTS VM with a 2-Core CPU, 4GB RAM, which is running on a notebook with Intel Core i7-5600U CPU 2.6 GHz, and 8 GB RAM. The design of the fog controller allows deploying it in the same manner as fog nodes and fog cells on Raspberry Pis. For public cloud resources, we use Amazon AWS EC2 services, specifically t2.micro VMs with the CoreOS operating system which has a Docker environment setup by default. Temperature and humidity sensors

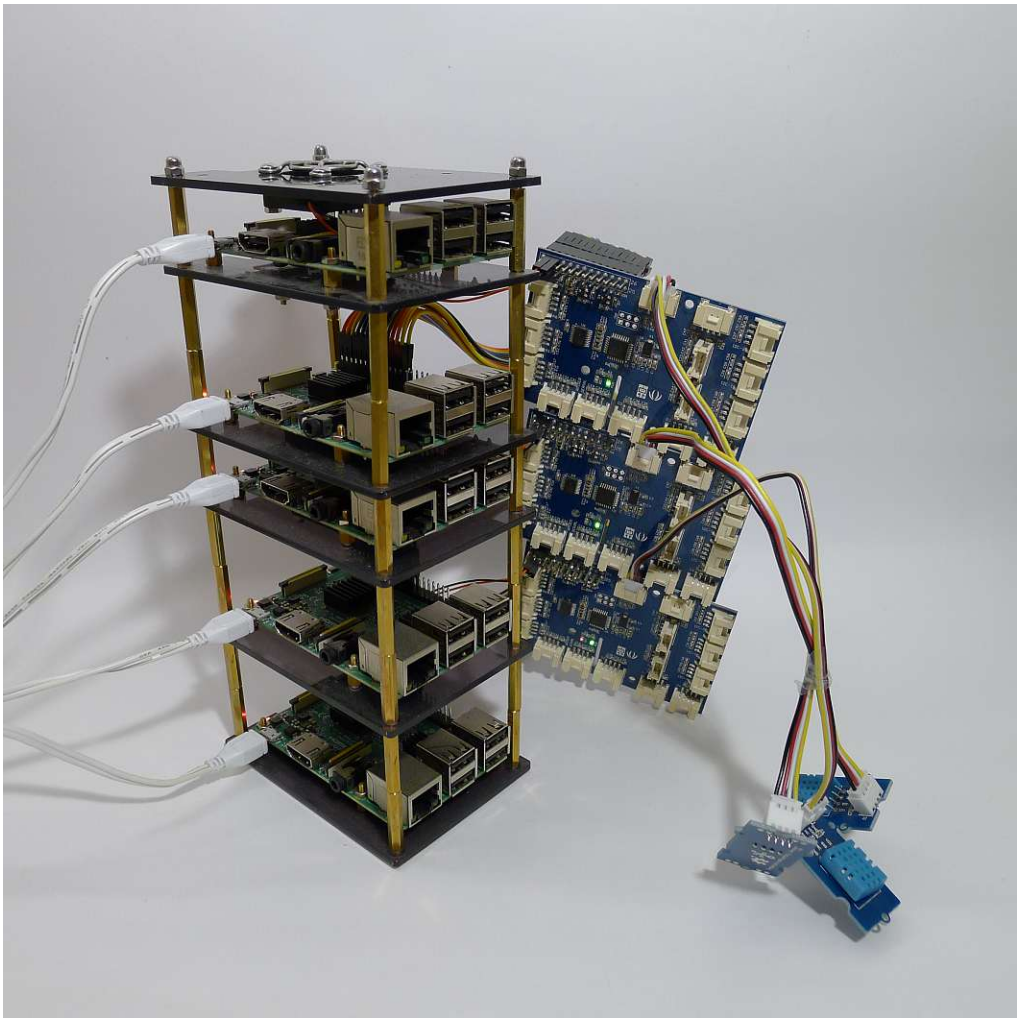

Figure 5.8: Raspberry Pi computers organized in a fog landscape.

are installed on the Raspberry Pis for fog cells by the means of GrovePi sensor boards[4].

Implementation Details

The FogFrame framework is implemented by means of Java 8 in combination with the Spring Boot framework which provides a convenient persistence handling with Spring Data and Java Persistence API. The framework is available as open source software at Github[5].

Service Deployment and Execution

Since the two computational environments in a fog landscape (i.e., cloud and edge) are different, the deployment mechanisms for services also need to differ. In the cloud, services are deployed on Docker containers on VMs, which need to be

[4]Online; Accessed: Apr. 2023 https://www.dexterindustries.com/grovepi/
[5]Online; Accessed: Apr. 2023 https://github.com/softls/FogFrame-2.0

deployed and managed. When a service request is sent to the fog controller from a fog colony and there is no deployed VM in the cloud, the fog controller leases and starts a new VM. If a VM is already running, the corresponding Docker container of the necessary service image is deployed on that VM. The containers are deployed on a cloud VM until a certain limit of containers is reached to ensure the stability of the computational environment. If there are no free resources for another container to be deployed on a VM, a new VM is leased and a container is deployed there. When the execution is finished, containers are stopped. If the VM is running with no load, the VM is stopped and the cloud resources are released again.

Service Deployment in the Fog

For fog colonies, the deployment mechanisms differ. Fog cells and fog nodes are by themselves services running inside their own Docker containers in a Docker runtime environment which is provided by the host operating system of Raspberry Pi units or cloud VMs. Therefore, during service deployment, a problem appears when trying to instantiate other Docker containers in the Docker runtime from inside the Docker containers of the running fog cells and fog nodes. To make it possible for fog cells and fog nodes to start and to stop further Docker containers on the host device, we make use of a *Docker hook* (see Figure 5.11) [14]. The Docker hook provides a communication mechanism from inside the fog cell's and fog node's Docker containers into the Docker environment of the operating system of the Raspberry Pi. It resolves the problem of instantiating other Docker containers on the Docker runtime of the host device from inside the Docker containers of the running fog cells and fog nodes.

Fog Landscape Connectivity

Services of FogFrame intercommunicate via REST APIs. The communication within the testbed is done via a WLAN private network provided by a Linksys Smart WiFi 2.4GHz access point. This access point also acts as a gateway to connect every Raspberry Pi to the Internet. Every component needs to be connected to the Internet since the fog services require the ability to download Docker image data in order to create and deploy services. The private network in which our fog landscape operates is deemed to be secure, and all components of the fog landscape communicate via dedicated API endpoints on certain ports and IP addresses specified in the framework. It is a matter of future work to research other appropriate security mechanisms for fog computing [136].

Virtualization

Regarding the virtualization technology, cloud resources are virtualized by the means of VMs. As discussed in Section 2.4, VMs are not a good choice for fog devices, so for them, we use Docker containers instead. The implemented service deployment and execution mechanisms for the cloud resources and fog colonies are different, since the hardware used in these environments differs. In order to use Docker containers in fog colonies, the base images of containers have to be compatible with the ARM processor architecture of Raspberry Pis, and accordingly in order to use Docker containers in cloud resources, the base images of those containers have to be compatible with the processor architecture of the

cloud-based VMs.

Sharing Service Images

A fog landscape consists of both cloud and edge resources which operate on different computing architectures [103]. Therefore, the service images of each application have to be compiled for the processor architecture they are intended to be executed on. To manage this heterogeneity, we apply two different solutions for sharing service images. Service images of services to be executed in the cloud are stored in the online repository Docker Hub[6], which is accessible by cloud VMs. In this case, service images are downloaded via a link address included in the service request.

Service Repository

For service images of services to be executed in fog colonies, we implement a shared storage that contains the shared service registry (see Section 5.4). Service images intended to be executed at the edge are pushed to a fog node along with the initial application request. Since applications are executed in a distributed manner inside fog colonies, every device in a colony requires access to the service images. For this reason, the devices of each fog colony share the service images in a *service repository* using a *shared storage*. FogFrame uses a distributed key-value store as underlying data management system within each colony to enable a flexible schema for sharing the service image data among fog nodes and cells.

Such distribution is necessary because in order to be executed on specific fog devices or in the cloud, each service image has to be compiled according to the processor architecture of that computational resource. In the case when services need to be executed in the cloud, they are downloaded via a link provided with a service request. This is necessary because we do not consider having a pre-configured pool of idle cloud resources with already stored service images. In the case when services need to be executed on fog devices, service images are sent to a fog node together with the initial application request. In fog colonies, applications (or more precisely: their services) are distributed between different fog devices, therefore every device needs access to the service images. This is ensured by the shared storage component (see Figure 5.11). It has to be noted that there is no limitation on where to host a shared storage because direct IP communication is established in the framework. The communication between services is performed with REST calls. The registration object in service registry contains a service key identifier, its Docker image, exposed ports and privilege rights to let the service almost same rights as the host. The ports and requests are predefined in the configuration of the devices. Fog cell and fog node configurations receive IPs of their parent devices during parent requests when connecting to a fog colony. Deployed services via dedicated endpoints propagate service data to fog cells where they are deployed, and then fog cells propagate data further to the fog nodes.

[6]Online; Accessed: Apr. 2023 `https://hub.docker.com/`

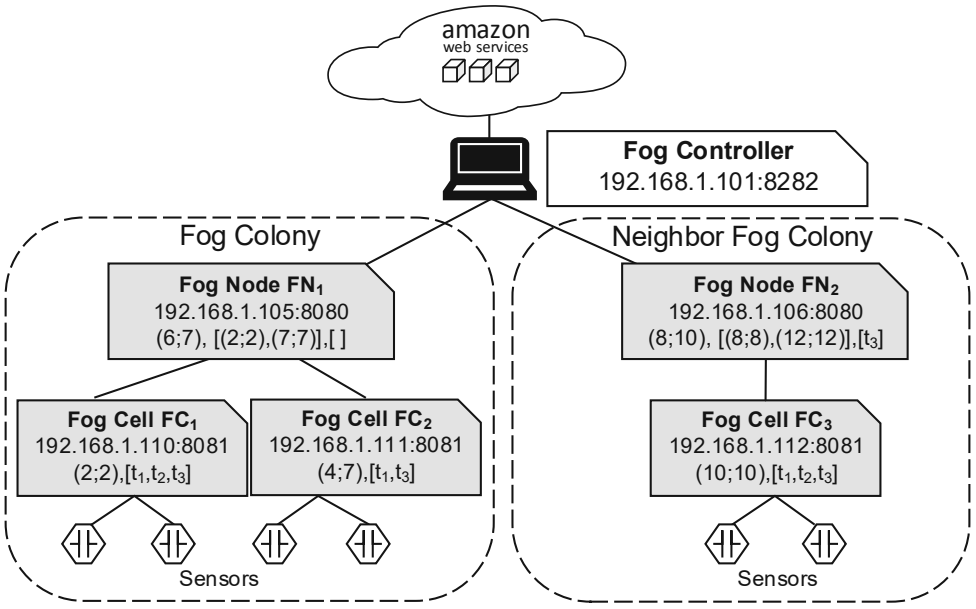

Figure 5.9: Experimental setup.

5.8 Experimenting with Resource Provisioning

In this section, we perform a testbed-based evaluation of FogFrame aiming to show:

- How deployment times of services differ at the edge and in the cloud,
- How the services are distributed in the fog landscape by the service placement algorithms according to different service request arrival patterns, and
- How much time is spent on producing a service placement plan.

5.8.1 Experimental Setup

Application Settings

In our experimental setup, each application consists of a number of services of certain service types, and is characterized by its makespan duration and a deadline on the deployment and execution time, as has been formally described in Section 5.6. For that, we have defined and implemented three possible service types: Services of type t_1 get data from temperature and humidity sensors and are executable only on fog cells because services of this type need sensor equipment; services of type t_2 and t_3 simulate processor load and are executable either on fog devices or in the cloud. We have also developed a dedicated service to be deployed and executed in the cloud which receives sensor readings and writes them to a cloud database.

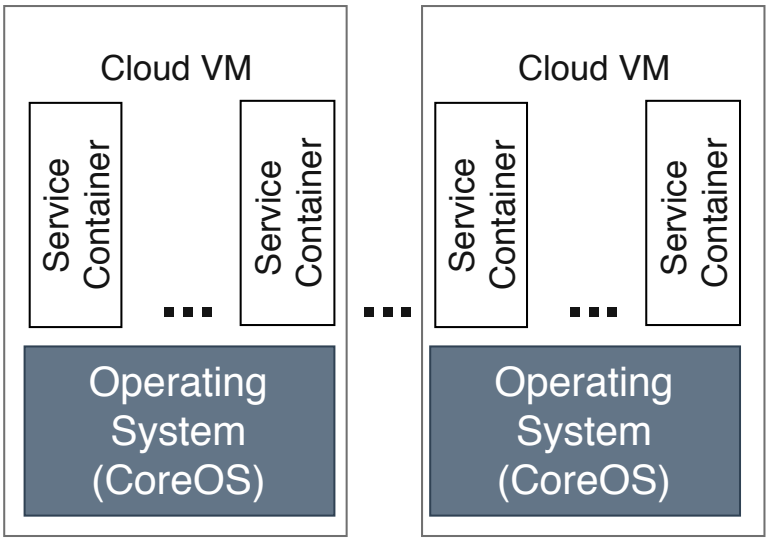

Figure 5.10: Deployment in the cloud.

In our evaluation, we implement two fog colonies with the head fog nodes FN_1 and FN_2, which are connected to the fog controller (see Figure 5.9). The fog colony controlled and orchestrated by FN_1 consists of two fog cells FC_1 and FC_2, which are within the coverage area of FN_1. The fog colony controlled and orchestrated by FN_2 has one connected fog cell FC_3.

5.8.2 Metrics

Service Deployment Time To assess time spent on deployment in the cloud and at the edge of the network, we calculate the service deployment time. This metric is separately evaluated for the cloud and for fog colonies. The service deployment time in the cloud depends on whether there have already been free VMs running, or if a new VM has to be started. Furthermore, the deployment time depends on the availability of the required service image. In case no free VM is available, the service deployment time in the cloud equals the sum of the VM booting time, the time to pull the service image, and the startup time of the Docker container in that VM. If a free VM is available but the service image has to be pulled, the service deployment time in the cloud equals the sum of the time to pull the service image and the startup time of the Docker container, meaning a cold start of the container. When both a free VM and the required service image are available, the service deployment time in the cloud equals the startup time of the Docker container. The service deployment times in fog colonies differ from the ones in the cloud because in the fog colonies no VMs have to be started before deploying the service containers. If a service image is not available locally

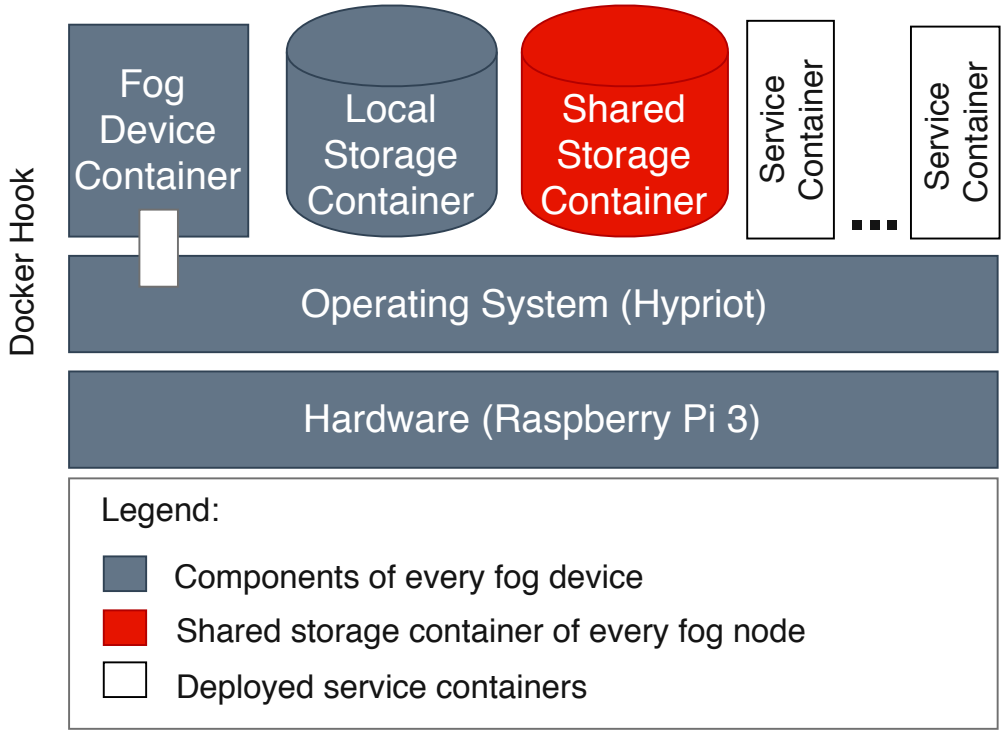

Figure 5.11: Components of fog devices.

on the fog device, the deployment time equals the sum of the time to pull the service image and the startup time of the Docker container. If a service image is available, the service deployment time equals the startup time of the Docker container.

Number of Deployed Services In order to show how services are distributed in the fog landscape by different service placement algorithms, we record the number of deployed services (containers) on each device in fog colonies and in the cloud. We also record the total deployment time of each scenario. Furthermore, we record the computational time of producing a service placement plan depending on the number of service requests.

In order to show how services are recovered in the case of a failure or migrated in the case of device overload, we record average metrics of recovery time per service and time to migrate a service due to a device overload. We also record how fast the framework reacts to a new device appearing in the fog landscape and deploys services on it.

Statistical Confidence

In the area of cloud computing and accordingly in fog computing, experiments are prone to variations due to a multitude of factors, for example, hardware differences and network quality. Some of the factors cannot be mitigated, instead, a sufficient number of repetitions of experiments ensures that their results are

not received due to a chance, but have a sound statistical confidence [110]. In order to record the mentioned metrics and show the distribution of results, we execute each experiment ten times. Through ten repetitions per experiment it was noted that the results did not show large variations, and it is a reasonable figure for the number of repetitions.

5.8.3 Experiments

Assessment of Deployment Time

In the first experiment, the aim is to assess deployment times. We show how deployment times of services differ in the cloud and fog colonies. The application used for this scenario is a cloud-edge data processing application with an equal number of 15 service requests to be deployed in the cloud and on the fog devices. The makespan duration of the application is 1 minute.

Arrival Patterns

Second, we show how services are placed on different fog devices in time. For this experiment, we use applications with different numbers of service requests according to different arrival patterns: constant, pyramid and random walk (see Figure 5.12). The arrival of application requests in all patterns happens every 60 seconds. The difference between arrival patterns is in the number of service requests in every application request that is submitted to the head fog node. In the constant arrival, an application request with 10 service requests is submitted to the fog node. Pyramid arrival means application requests with 5, 10, 15, 10, and 5 service requests accordingly are submitted with the same time span of 60 seconds between application requests, and then the pattern repeats itself. In the random walk arrival pattern, an application consisting of a random number of service requests from 2 to 15 is submitted for execution every 60 seconds. The total number of application requests is 20 per one experiment regardless the chosen pattern. The numbers of service requests in each arrival pattern aims to test the load on the fog to the maximum and over the maximum capacities in the fog resources in order to observe delegations of requests between fog colonies. The start of each experiment is marked with the first submitted application request. The last request in each experiment is submitted at the 19th minute into the experiment. Each application has a makespan duration of 1 minute and a deadline on deployment and execution times of 3 minutes. The arrival patterns are shown along with the representative results of experiments. We evaluate the two service placement algorithms presented in Section 5.6.2 – the greedy algorithm and the GA. One VM and one Docker container with the service to write sensor data into the cloud database are started before the experiment to receive sensor data. Therefore, in total in this evaluation we observe 60 different experiments which take more than 1600 minutes (approx. 27 hours) of compute time in the fog based on the average time of one experiment (see Table 5.2).

Computational Time

In the third experiment, we submit applications with different numbers of service requests to fog node FN_1, and observe the time needed for the GA to produce a service placement plan in each case. The experiment is repeated ten times

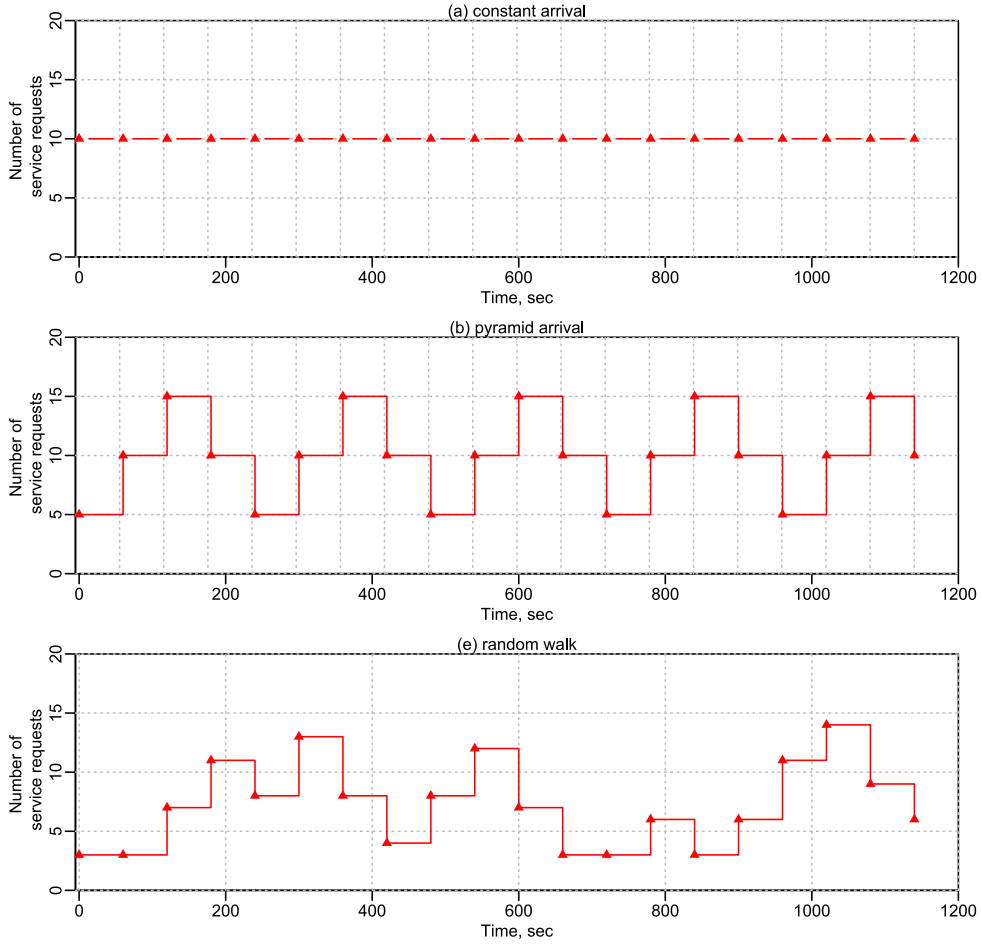

Figure 5.12: Different arrival patterns of service requests.

for 5, 10, 15, 25, 50, 100, 200, and 400 services, in order to show how the GA's computational time increases if the number of services grows. Computational times are recorded only for the GA because, as has been described in Section 5.6.2, in the greedy algorithm the deployment happens immediately when appropriate edge devices are checked for placement, while in the GA a service placement plan is generated, and only afterwards the services are deployed.

5.8.4 Results and Discussion

Assessment of Deployment Time

In the application executed in the first experiment regarding the assessment of the deployment time, there are 30 service requests. Out of these, 15 need to be deployed in the cloud, and 15 need to be deployed in the fog colonies. When services have to be deployed in the cloud, in addition to the high start-up times of VMs, VMs do not have previously stored or cached data, for example, previously

Table 5.2: Assessment of deployment time.

(in seconds)	max	min	μ	σ
Total (edge)	41.78	27.22	29.76	4.08
Per service (edge)	2.78	1.81	1.98	0.27
Total (cloud)	251.03	180.75	209.42	20.41
Per service (cloud)	16.74	12.05	13.96	1.36
VM startup	72	40	48	12
Image pull (start)	33	32	32	0.01
Total	278.94	210.42	239.18	18.77

used service images. Therefore, for the cloud VMs, Docker images need to be pulled every time. In contrast, fog devices download service images only once and then reuse them whenever needed as the images have been cached. As can be seen in Table 5.2, there is a significant difference between the measured service deployment times at the edge and in the cloud. The average total deployment time at the edge is at about 29.76 seconds ($\sigma = 4.08$), whereas the average total deployment time in the cloud is 209.42 seconds ($\sigma = 20.41$). The Docker image pull times of the VMs have been also recorded in this experiment. It takes on average 32 seconds ($\sigma = 0.01$) to pull and start the Docker container in the cloud (see Table 5.2).

Summary of Experiment 1

To summarize the outcome of this experiment, we compare the deployment times in the cloud and in fog colonies. The deployment time in the cloud is higher than the deployment times in the fog colonies because of additional latency, VM start-up time, and service image download time before instantiating corresponding containers. In fog colonies, the shared service registry ensures caching of all available service images at the time fog devices enter the fog colony. In the already running VMs in the cloud, all necessary service images are already cached and can be reused. However, each additional new VM in the cloud requires instantiation time and service image download time and caching time. Having a pre-configured pool of idle cloud resources with cached service images in the same manner as the shared service registry in fog colonies would negate the whole concept of on-demand resources of the cloud. Therefore, cloud resources can be an on-demand addition to the fog landscape, but are not suited to be the only computational resource for latency-sensitive IoT applications.

Service Placement with Different Arrival Patterns

If applying the greedy algorithm presented in Section 5.6.2 and different arrival patterns (see Figure 5.13), services are placed on fog cells to the maximum capacity according to the available utilization parameters. If both fog cells in the first fog colony (see Figure 5.9) are loaded to the maximum capacity, and a new application request arrives with some services which need sensor equipment, the deployment in the own fog colony becomes impossible, and therefore such a request is delegated to the closest neighbor fog colony.

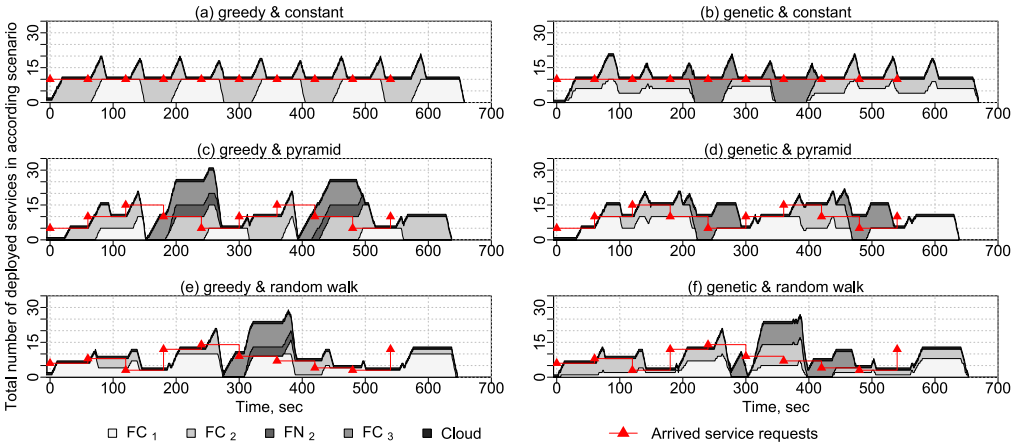

Figure 5.13: Results of experiments with different placement algorithms and arrival patterns.

Performance of Algorithms

In the GA, as discussed in Section 5.6.2, the requested applications are distributed in a more balanced way between fog colonies and the cloud. Fog devices are loaded less than to the maximum capacity. By delegating applications between the colonies and distributing single services on different fog devices, the GA placement spreads the load on the resources in fog colonies more efficiently, which may be crucial if additional application requests are submitted and their services need specific equipment, for example, temperature and humidity sensors.

In the pyramid and random walk arrival patterns, both algorithms perform almost alike due to the fact that even if the greedy algorithm loads one fog cell for all services in the application request, in most of the application requests the workload is less than the maximum capacity of the available fog devices. However, as can be seen in Figure 5.13, the load on fog devices is nevertheless more distributed if the GA is used to compute a service placement plan.

Summary of Experiment 2

To summarize the results of experiment of varying different arrival patterns of service requests and placement algorithms, the GA performs better with regard to distributing service requests within fog colonies. This makes it possible for newly requested applications to be placed on the necessary resources. While the greedy placement does not involve cloud resources, the GA spreads the load between the fog colonies and the cloud. In the closest neighbor fog colony, the deployment time per service is longer as there is only one fog cell connected, the fog node's resources also execute services, and services are deployed sequentially. One particular positive aspect in the GA's placement plan is that the resources in the fog landscape are not close to overload, which gives more opportunities for newly requested services to be deployed. The results of the experiment are summarized in Table 5.3.

Table 5.3: Overview of the experiments with different arrival patterns.

Metrics	Algorithm	Constant	Pyramid	Random
Deployment time per scenario (sec)	Greedy	694.10 (σ =70.92)	643.50 (σ =9.54)	663.67 (σ =19.99)
	Genetic	684.00 (σ =16.34)	644.88 (σ =9.00)	654.90 (σ =7.06)
Service deployment time (sec)	Greedy	3.67 (σ =1.85)	2.18 (σ =0.25)	2.90 (σ =1.43)
	Genetic	2.08 (σ =0.32)	2.12 (σ =0.27)	2.00 (σ =0.21)
Service deployment time in the neighbor fog colony (sec)	Greedy	2.52 (σ =0.60)	2.50 (σ =0.24)	2.98 (σ =0.30)
	Genetic	2.12 (σ =0.13)	2.23 (σ =0.27)	2.38 (σ =0.56)
Service deployment time, cloud (sec)	Genetic	3.25 (σ =0.18)	3.97 (σ =0.13)	3.19 (σ =0.92)
Number of services delegated	Greedy	18 (σ =4)	24 (σ =4)	16 (σ =6)
	Genetic	24 (σ =9)	22 (σ =4)	13 (σ =4)

Computational Time

The measurements in this experiment show that the computational time of producing a service placement plan using the GA is less than a second on average in all the cases below 50 service requests (see Figure 5.14). After that, an increase is observed, however, that increase is still within reasonable boundaries: a service placement plan for 200 and 400 service requests is produced on average in 2.26 seconds ($\sigma = 0.16$) and 3.64 seconds ($\sigma = 0.19$), respectively.

Summary of Experiment 3

The relatively small spread of whiskers and box sizes in Figure 5.14 show that the computational time is rather stable for the specified number of services. The results mean that increasing considerably the number of services to be deployed affects the computational time, as the GA calculates fitness values for each chromosome in each population as well as estimations to response times of applications and fog landscape resource utilization.

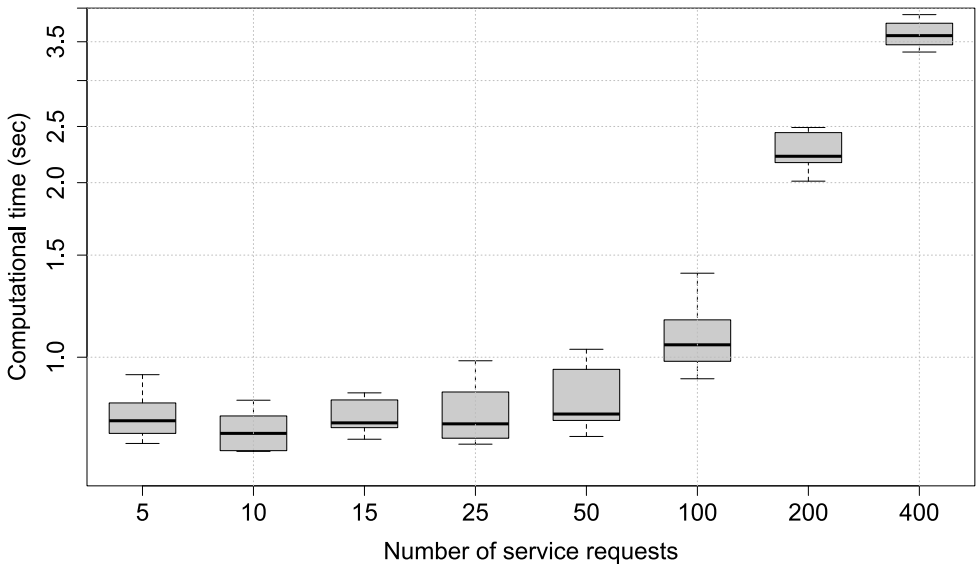

Figure 5.14: Computational time of producing a service placement plan by the GA.

5.9 Experimenting with Runtime Events

To evaluate FogFrame, we examine how services are distributed in the fog landscape under different arrival patterns of service requests, and how FogFrame reacts to different runtime events in the fog landscape.

5.9.1 Experimental Setup

Fog Landscape Setup

As can be seen in Figure 5.15, fog nodes FN_1 and FN_3 are connected to the fog controller. The fog colony controlled and orchestrated by FN_1 consists of fog node FN_2, and two fog cells FC_1 and FC_2. The fog colony controlled and orchestrated by FN_3 has one connected fog cell FC_3. The implementation details of the framework have been already presented in Section 5.7. The difference of this fog landscape setup compared to the setup in the previous section is that we showcase multiple fog nodes in a fog colony, i.e., we add the additional FN_2, which is a fog node that acts as a fog cell, i.e., it can execute certain services, however, it also provides all the communication between fog cells FC_1 and FC_2 and the head fog node FN_1.

Application Settings

In the experimental setup, each application consists of a number of service requests of certain service types. For the purposes of the evaluation, we have defined and implemented three possible service types: Services of type t_1 receive data from temperature and humidity sensors and are executable only on fog cells because

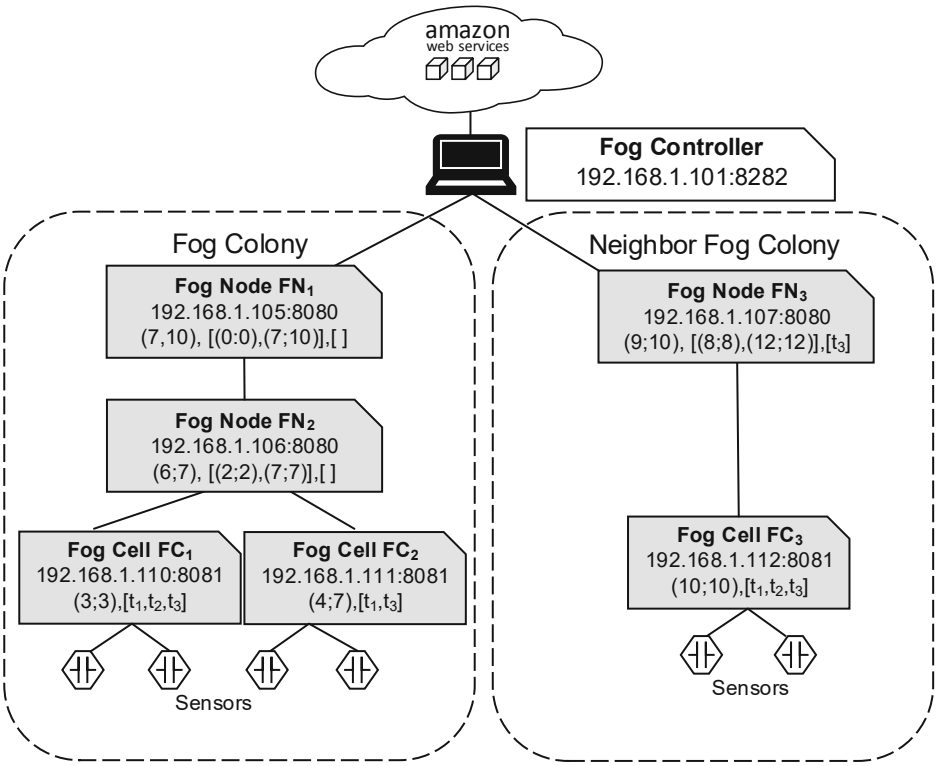

Figure 5.15: Evaluation setup.

services of this type need sensor equipment; services of type t_2 and t_3 simulate processor load by continuously writing into a string, and are executable either in fog colonies or in the cloud. We have also developed a dedicated *cloud service* which receives sensor readings and writes them to a cloud database.

5.9.2 Experiments

Baseline

To evaluate the behavior of FogFrame at runtime tackling the runtime events, we apply different arrival patterns of application requests, i.e., constant, pyramid, and random walk, as has been already presented in Section 5.8.3, and observe service placement. The arrival patterns are shown along with the representative results of experiments in Figure 5.16. The baseline for evaluation is the uninterrupted operation of the fog landscape observed during the first ten minutes of each run of the experiment. After ten minutes, failures are automatically introduced.

Failures

Specifically, failures are generated with the probability $P(Failure) = 0.05\%$ per second, i.e., $P(Failure) = 30\%$ in 10 minutes, and recorded to be introduced in each run. The device failure is simulated by stopping the fog cell application

container at the chosen device. In reality, device failures can be hardware, software, or communication problems. Whenever a fog cell has a failure, we use its Raspberry Pi to deploy a new fog cell. In this evaluation after the failure of FC_1 we deploy a new cell FC_4, and then after the failure of FC_4, we deploy a new cell FC_5.

Service Migrations

We observe how the system redeploys services running on the disconnected fog cell and measure time-to-recover per service. The overloads may occur in the course of execution, e.g., if the CPU load is 100%. In the case of an overload, the device is still operating, however, some services have to be redeployed to release resources, e.g., using a newly joined fog cell, already existing resources, or cloud resources. We run the experiment ten times for each arrival pattern in combination with each placement algorithm and provide statistical distributions of results.

Metrics

In order to show the results of the experiments, we measure the following metrics:

- Deployment time per service at the edge and in the cloud,
- Time-to-recover per service due to failures,
- Number of redeployed services due to failures,
- Percentages of successful recovery, and
- Time-to-redeploy a service due to overloads.

In order to examine the behavior of FogFrame, we also record the workload of each device in terms of the number of deployed services over time. We examine this load in a stacked form, i.e., the load in each device and the total load.

5.9.3 Results and Discussion

Different Arrival Patterns and Algorithms

The evaluation results for the six combinations of the two algorithms (first-fit, GA) and three arrival patterns, i.e., constant, pyramid, random walk, during the first ten minutes of each run are shown in Figure 5.16. In the first-fit placement, the services are placed on the fog cells to the maximum capacity. If both fog cells FC_1 and FC_2 are loaded, and a new application request arrives with services which need sensor equipment, the deployment in the current fog colony becomes impossible, and the fog node decides to delegate the application to the closest neighbor colony. In the GA, more time is spent for the deployment of single services compared to the first-fit placement because the algorithm requests device utilization information before starting calculations. However, more application requests are delegated to the neighbor fog colony, and the load on the resources is better balanced, which allows to avoid overloads and is crucial if new services need sensor equipment.

Experimenting with Runtime

After ten minutes, we observe how FogFrame reacts on different runtime events. The first case to consider is when a connected and successfully paired fog cell loses its connection with the fog landscape due to a device failure. In this experiment,

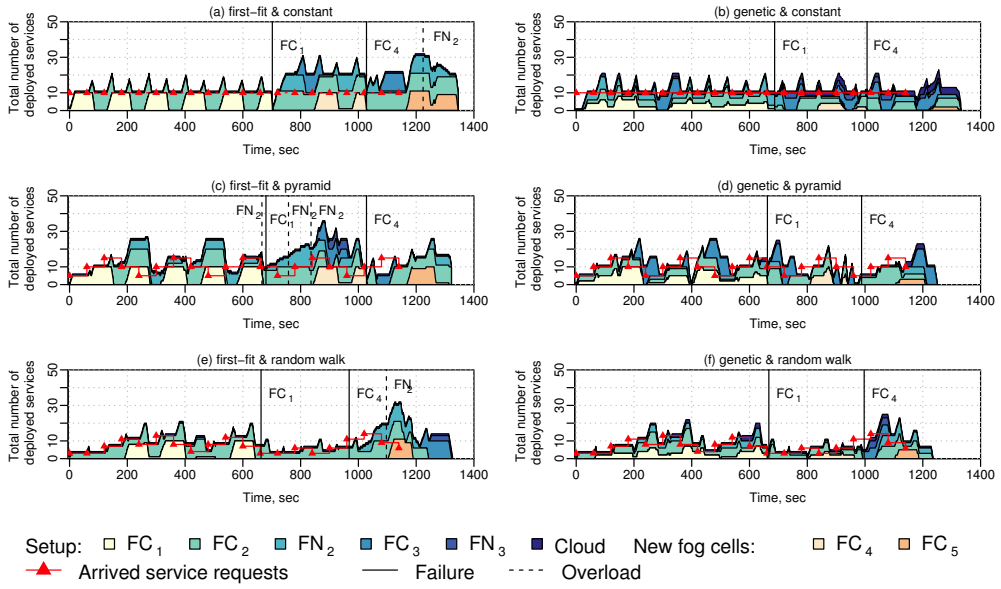

Figure 5.16: Results of experiments with different placement algorithms and arrival patterns.

the device to fail is the Raspberry Pi of FC_1. We deploy new fog cells: FC_4 after the first failure of FC_1, and FC_5 after the second failure of FC_1.

In the first-fit placement, the percentage of successful recovery is lower than in the GA placement. This happens because in the first-fit placement both fog cells are fully loaded, and some services which need sensor equipment cannot be redeployed in the fog colony. In the GA, the fog colony's resources are not loaded to the maximum capacities, and therefore less services need to be redeployed, and consequently the fog colony has enough resources to perform such redeployment. In the course of the experiments, overloads occur mostly in FN_2 because this fog node apart from executing services also handles the communication between the connected fog cells and the cloud. In the GA placement, the underlying fog cells and FN_2 are not fully loaded, and FN_2 has enough utilization capacities to deal with the communication.

Experiment Summary

To summarize this experiment (see Table 5.4), the GA performs better with regard to distributing service requests within the fog colony and between the fog colonies. The positive aspect in the GA placement is that the resources in the fog landscape are not close to overload, as can be also seen in Figure 5.16, giving more opportunities for newly requested services to be deployed, i.e., because of better availability of sensor equipment.

Table 5.4: Overview of the experiments with runtime events.

Metrics	Algorithm	Constant	Pyramid	Random
Deployment time per service, edge (sec)	First-fit	2.41 σ =1.10	3.07 σ =0,93	3.06 σ =1.08
	Genetic	2.92 σ =1.16	3.02 σ =0.92	3.05 σ =1.17
Deployment time per service, cloud (sec)	Genetic	2.78 σ =0.18	2.69 σ =0.13	3.18 σ =0.63
Number of services delegated	First-fit	26 σ =21	24 σ =21	24 σ =10
	Genetic	59 σ = 9	46 σ = 20	26 σ = 13
Number of services to recover	First-fit	21 σ =3	11 σ =6	10 σ =5
	Genetic	6 σ = 4	7 σ = 4	6 σ = 2
Time-to-recover per service (sec)	First-fit	3.88 σ =3.12	2.49 σ =1.18	1.98 σ =0.12
	Genetic	2.09 σ =0.24	2.02 σ =0.18	2.07 σ =0.10
Successful recovery (%)	First-fit	90.00 σ =12.84	89.36 σ =14.62	99.11 σ =2.36
	Genetic	100.00 σ =0.00	100.00 σ =0.00	100.00 σ =0.00
Time-to-redeploy per service (sec)	First-fit	1.80 σ =0.06	9.41 σ =2.95	3.16 σ =1.13
	Genetic	2.46 σ =0.05	2.49 σ =0.05	2.37 σ =0.17

5.10 Summary

In this work, we have designed and developed a framework for creating and maintaining a fog landscape, and enabling IoT application execution according to specified requirements. The foundation for the framework are lightweight container technologies and loosely-coupled components to provide a stable and fault-tolerant distributed system. By using this framework and based on single-board computers, we have built a real-world fog computing testbed able to receive requests for execution, allocate resources, deploy services, and execute IoT applications. Having addressed the question of how to create a fog landscape, we have focused on the development of the communication and application management mechanisms. The framework enables the easy integration of new components due to well-defined interfaces.

FogFrame provides the necessary mechanisms for configuring a fog landscape as well as for supporting application management and lifecycle. During the implementation, we have identified and resolved technical issues of how to create a real-world fog landscape based on Raspberry Pi computers, which are considered as representative devices for fog computing as discussed in Section 5.7. We have investigated how to instantiate containers in different computing environments and how to store images of services and share those images within the available infrastructure. A crucial part of this work has been devoted to the problem of how to effectively distribute services in a fog landscape. Therefore, we formalized a system model, and implemented a GA as well as a greedy algorithm for service placement taking into account our findings about the real-world testbed.

Experiments are conducted to assess deployment times of applications, service placement algorithms with different arrival patterns of service requests, and computational time depending on the workload to be processed. The framework is evaluated with regard to QoS parameters of IoT applications and the utilization of fog resources using a real-world operational testbed. The evaluation shows that the service placement is adapted according to the demand and available resources in the fog landscape. The greedy placement leads to the maximum utilization of edge devices keeping at the edge as many services as possible, while the placement based on GA keeps devices from overloading by balancing between the cloud and the edge. When comparing edge and cloud deployments, the service deployment time at the edge takes 14% of the deployment time in the cloud. If fog resources are utilized at maximum capacity, and a new application request arrives with the need of certain sensor equipment, service deployment becomes impossible, and the application needs to be delegated to other fog resources. The GA better accommodates new applications and keeps the utilization of edge devices at about 50% CPU. The GA placement distributes services in a balanced way, while the greedy algorithm loads each of the fog devices with the maximum available resource capacity. If there are no sufficient resources in the current fog colony, applications are delegated to the closest neighbor fog colony. The computational time of the GA remains stable for the specified number of services, however with a considerably increasing number of services to be deployed, the GA performs accordingly more computational operations to evaluate each possible solution.

In this chapter, we focus also on monitoring and management of computational resources in the cloud and at the edge of the network, as well as on the orchestration of those resources in order to deploy arbitrary services. To enable these functionalities, FogFrame provides means for communication between different resources in a fog landscape, means for application management, and means for tackling runtime operational events that may occur in such a volatile system, i.e., failures or losses of connection between devices. We evaluate the framework and the proposed mechanisms to tackle the volatility in the fog using the testbed.

During our experiments, we recorded the workload on different computational resources at the edge of the network and in the cloud, deployment times of services, time to recover and recovery rates in the case of failures and overloads. The framework successfully reacts to runtime events:

- Services are recovered when devices disappear from the fog landscape;
- Cloud resources and highly utilized devices are released by migrating services to new devices;
- In case of overloads, services are migrated in order to release resources.

Services are successfully recovered and migrated when fog cells fail or experience overload. Device discovery ensures efficient balancing and horizontal scalability in the fog landscape. This releases cloud resources and other fog cells, which may be intensively used in a fog colony, by migrating necessary services onto discovered devices. The framework records information for measuring efficiency and for evaluating performance based on application execution and resource utilization monitoring.

This chapter constitutes Contribution III as defined in Section 1.2 and covers requirements 11-18 from the identified requirements in Section 2.6:

11. A fog computing framework should be able to create and maintain a fog landscape made up from computational resources at the edge of the network and in the cloud.
12. The framework should establish communication and interactions in the fog.
13. It should efficiently perform service delivery, deployment and execution of IoT applications.
14. For a resource- and QoS-efficient execution of the applications, resource provisioning should be implemented.
15. IoT application execution may be requested at any time. The framework should accordingly react to the workload and trigger resource provisioning. The available resource pool should be shared and optimized among submitted IoT applications.
16. The IoT application execution as well as the resource pool should be monitored at runtime. The monitoring should provide necessary data for the evaluation of resource provisioning constraints.
17. The framework should dynamically react to the volatility of the fog. Corresponding countermeasures should be performed to tackle the volatility, i.e., when devices lose connection or get overloaded, as well as when new resources appear in the fog.
18. The framework should provide means to overview execution results and calculate different performance metrics both of the framework runtime and resource provisioning, e.g., deployment and execution time, number of successfully executed applications, failure rate, QoS of single applications, or utilization of resources.

CHAPTER 6

State of the Art

The idea and initial results in this dissertation have been inspired by the seminal works of Bonomi et al. [25, 26], Vaquero et al. [141], and Dastjerdi et al. [45]. The authors discussed that applications may be placed in the fog, spanning potentially heterogeneous cloud and edge resources, and that fog computing has to enable different communication links, i.e., within the fog on different levels: between different IoT devices, at the edge and with the cloud. In their discussion of basic fog features, different concepts to realize fog architectures are considered, including both centralized and decentralized, i.e., P2P, approaches. In particular, the authors introduce the notion of edge clouds, which are private fogs made up from IoT devices, resembling the notion of fog colonies in this dissertation.

This inspiration resulted in the first core publications of this dissertation (see Core Publications) in early 2016 and 2017 abiding the mainstream in fog computing research and introducing first concepts and resource provisioning models. After that, in publications in 2018 and 2021, the work focused on the real-world implementation and evaluation of various resource provisioning approaches with the fog computing framework FogFrame.

Naturally, there is conceptual as well as fundamental work in related areas, which needs to be regarded, despite it being published concurrently or later than our publications. Therefore, this chapter considers fog computing conceptual mechanisms and simulations, resource provisioning and service placement methods, real-world fog computing environments and implemented testbeds, and investigates the state of the art on how to create and maintain a fog landscape.

6.1 Resource Provisioning and Service Placement

Surveys

In the following, we consider related work in the area of resource provisioning and service placement in the fog. Notably, hundreds of studies on service placement in the fog have been presented in recent years [19, 76, 123, 127]. An interesting systematic review has been presented by Kashani et al. [88]. It provides comprehensive descriptions, advantages and disadvantages of approximate, exact, fundamental, and hybrid methods of load balancing in fog computing. Most of the methods mentioned in the survey are simulated, and it is promising to implement the mentioned methods in a real-world fog computing environment. Furthermore, some cloud computing resource provisioning methods can be adapted to fog computing. For example, in a survey by Afzal and Kavitha [7], advantages and limitations of existing load balancing methods in cloud computing are considered. These methods can be implemented in FogFrame within the fog controller's reasoning mechanisms to manage additional cloud resources for fog colonies.

In the following, we focus on the closest work to our research on resource provisioning and service placement in fog computing.

Hong et al. [77] and Saurez et al. [126]

Hong et al. [77] 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. In the work of Saurez et al. [126] the approach of Hong et al. is extended by implementing the envisioned APIs and adding algorithms regarding the discovery, migration, and deployment of fog cells and services. Furthermore, an experimental real-world fog landscape is set up using containers deployed on several servers.

Aazam and Huh [1]

Aazam and Huh [1] present a sophisticated resource provisioning mechanism based on the prediction of resource demands by means of historical usage data, records of past customer activities, and pricing models. This approach is aimed at cost optimization, while resource allocation depends on the probability fluctuations of user workload demands, types of services they need, and their loyalty index. In another paper, the authors propose an improvement of the theoretical resource management model in terms of specifications of utilization and QoS in the context of multimedia IoT devices [3]. These are interesting works focusing specifically on pricing models and economy aspects, which are not implemented in FogFrame. FogFrame's scope is on the fog landscape itself enabling efficient execution of IoT applications, i.e., how to create a fog colony from the first device, how to make devices interact, and how to execute applications in the fog. In the papers of Aazam and Huh, the authors focus on establishing customer service from the very first customer, i.e., how resources for a customer need to be estimated and allocated based on customer history. In order to implement the mentioned pricing models in the real-world fog in FogFrame, another level of management, i.e., customer management, is needed within fog colonies.

Vögler et al. [145]

In the work of Vögler et al. [145], a rule-based approach to optimize deployment topologies on edge devices is presented. This approach offers an elastic application deployment by the means of defining a 'hot pool' of resources per service of requested applications to enable additional scaling. A framework for dynamic generation of deployment topologies for IoT applications called DIANE is proposed. This framework monitors the deployment infrastructure, groups it according to available resources, and stores the results for later analysis. It dynamically deploys topologies for IoT applications by the means of a rule-base algorithm, and provides their monitoring. In DIANE, only a rule-based algorithm is used for resource provisioning, while in our work, we formulate a concrete optimization problem. We also consider and implement a hierarchical structure of a fog landscape focusing specifically on establishing communication between different devices in the fog landscape, and providing application management. We also consider runtime service placement optimization to account for dynamic infrastructural changes in a fog landscape. We concentrate on a concrete formalization of the optimization problem to be able to use this formalization in order to apply different optimization and heuristic algorithms to solve this problem.

Urgaonkar et al. [138]

Urgaonkar et al. [138] present an approach on how to distribute services between edge clouds, which resemble our notion of fog colonies. The authors assume that there is a central cloud node, which controls the edge clouds. The placement of services onto resources is formulated as a Markov Decision Problem and solved by means of a control algorithm based on Lyapunov optimization, minimizing the cost of execution and accounting for delays and location in constraints. In contrast, in our work, fog resource and application models are explicitly formulated. The fog landscape is considered as an extension of the cloud, and the optimization model takes into account QoS metrics of applications.

Deng et al. [48]

Deng et al. [48] formulate a workload allocation problem for a fog landscape. For this, the power consumption, computation and communication delays of three subsystems are considered, i.e., of fog computing, WAN communication, and cloud computing. According to these three subsystems, the authors formulate three separate optimization problems to minimize power consumption and delays in different environments. After solving these problems, the approximation problem of how to unite these three solutions is proposed, but the evaluation of this approximation is left for future work. In our work, we use different criteria for service placement, i.e., we take into account QoS parameters of applications, however, we do not account for power consumption of fog resources. This aspect may enhance our optimization problem in the future.

Brogi and Forti [30]

Brogi and Forti [30] introduce the *FogTorch* tool which aims to perform resource provisioning in a fog landscape. FogTorch accepts a fog landscape infrastructure and application specifications as inputs, and calculates a deployment model. The basis for the deployment is QoS-aware service placement. The placement approach at first preprocesses all input requests to search a map of available resources

for each service in each request and backtracks the results of preprocessing to guarantee the deployment of all services. Then it applies a heuristic fail-first algorithm to ensure the deployment of those services for which there are fewer compatible nodes and which have bigger demands for resources. In contrast, in our work, we implement FogFrame along with our own resource provisioning mechanisms, and apply it in a real-world Raspberry Pi testbed. The integration of FogFrame with FogTorch may become a good opportunity in the future to deal with the reoptimization of network topologies of a fog landscape.

Xiao and Krunz [153] Xiao and Krunz [153] consider an offloading problem in fog computing. Optimization is performed based on power consumption and Quality of Experience (QoE) parameters. Their approach is called *offload forwarding*. In contrast to our work, the authors use different criteria for optimization, namely, QoE and power consumption. This is an interesting approach, and our models may be extended to take into account power efficiency of the fog landscape and QoE inputs from users.

Ni et al. [108] Ni et al. [108] propose a resource allocation technique for fog computing which is based on Priced Timed Petri nets. In the application model used in their work, a fog application is orchestrated from single services. The time and price for execution of these services differ for each device. The resources for allocation are chosen by the users depending on the information received from the Petri net and their own demands. In our approach, users are not involved in the orchestration component, and the reasoning service reacts to application requests automatically.

Saurez et al. [126] Saurez et al. [126] propose to allocate resources and migrate services in the fog based on two possible triggers, i.e., meeting latency constraints and resolving resource pressure. In contrast, in our work, apart from latency and resource constraints, we take into account QoS parameters, namely, deadlines on deployment and execution of applications.

Nardelli et al. [107] Nardelli et al. [107] introduce several heuristic approaches to efficiently identify service placement considering the volatility of computing resources. They simulate different network topologies and sizes of fog infrastructure. In their work, several meta-heuristic algorithms are implemented: a greedy first-fit, tabu search and local search algorithms. According to the findings of Nardelli et al., the greedy first-fit algorithm is the fastest, however with the worst quality, whereas the local search heuristics shows the best performance trade-off. In our work, we apply different heuristic algorithms for service placement in a real-world fog landscape. We also introduce different placement mechanisms to account for device discovery, failures and overloads.

Mseddi et al. [105] Mseddi et al. [105] introduce service placement implemented by the means of particle-swarm optimization, a greedy algorithm, and an exact optimization. The goal of their placement is to maximize the number of executed applications

adhering to their time constraints. According to their results, Mseddi et al. state that the particle-swarm optimization yields high resolution times and is not viable in fog computing environments. Their greedy algorithm aims to minimize the distance and delay between used fog resources taking into account their utilization. In contrast, our greedy algorithm aims to maximally utilize a fog colony adhering to QoS requirements and capacities of available resources as well as to the types of services. In general, the service placement approach in our work differs from the work of Mseddi et al. since it considers multiple fog colonies and offloading of applications as well as contains separate policies to tackle operational events in a fog landscape.

Spillner et al. [133]

A novel rule matching approach to execute composite fog applications is proposed by Spillner et al. [133]. Their goal is to automate deployment patterns of applications on different compute resources by solving the assignment problem with a match-making approach. Their holistic definition of rules enables specification and control of constraints in the system. The placement of single services of IoT applications is achieved by ranking of all possible combinations of deployments on all resources. The demonstration of performance is done via OsmoticToolkit, which is an emulator of different computing environments. Their work considers different types of rules and constraints, which may be beneficial to enhance our FogFrame model in future. Compared to the work of Spillner et al., we address some of their mentioned limitations, for example, we account for the volatility in the fog and perform migrations of services based on fog runtime events. Furthermore, our watchdog services monitor each of the fog devices, and provide those characteristics to the reasoner component in each fog node.

Abedi and Pourkiani [5]

In the work of Abedi and Pourkiani [5], the authors introduce a service placement algorithm based on an artificial neural network aimed to minimize response times of applications while distributing them in the fog landscape. Their approach is simulated with MATLAB, and therefore it is not clear how long it takes to produce such a neural network in the real world. It has to be noted, that this approach, as well as any other Machine Learning (ML) model, requires a considerable volume of training data. This means that before any neural network can be created, other placement algorithms or service placement policies have to be used to historically record those placement decisions to receive a viable training dataset.

Mostafa et al. [104] and Abdulkareem et al. [4]

Mostafa et al. [104] also implement an artificial neural network in a simulated environment to make predictions of service placement based on historical placement data. The algorithms in our work could provide a basis for training data and eventually be substituted by ML models. An interesting recent survey of Abdulkareem et al. [4] discusses areas where ML can be applied in fog computing: ML for specific IoT service implementations and ML for decision making in resource provisioning.

Baresi and Mendonça [15] In work by Baresi and Mendonça [15], building blocks of a Serverless Edge Platform are proposed to enable different application scenarios in densely distributed fog with the Function-as-a-Service (FaaS) application model. The authors envision multiple such serverless edge platforms at the network edge, which are capable to self-organize and advertise their own resource pool capabilities and performance scores. Alongside each of such platforms there is a load balancer, which implements least response time optimization. The authors also propose a mechanism to enable stateful services by storing session tokens. Their interesting approach to deal with stateful services may be introduced in the future into FogFrame's resource provisioning.

Bermbach et al. [20] In a recent work by Bermbach et al., a conceptual auction approach to enable FaaS in the fog is introduced. In this approach, application developers place bids for executing serverless FaaS funtions in the fog. These requests are collected and presented to fog nodes, which can either accept the storage and execution of the each of services or delegate them based on available workload capacities. Their decision is based on performing comparison of bids and finding more 'attractive' ones. The system proposed is called AuctionWhisk which is realized as a proof-of-concept simulated testbed of a FaaS platform. In FogFrame, compared to AuctionWhisk, we solve different challenges. We focus on enabling both, a real-world fog landscape with volatile fog devices potentially entering or leaving the fog thus forming fog colonies, and on resource provisioning to efficiently execute IoT applications.

Summary To conclude this overview, the considered related works in resource provisioning and service placement differ in many aspects, e.g., in the way the fog is simulated or represented within models, how the goals of resource provisioning are formulated and implemented, in introducing different constraints depending on the enabled computational environment. This thesis introduces various approaches while enabling end-2-end workflow, i.e, from creating a fog landscape to executing real-world IoT applications and tackling volatility of the fog at runtime. Each of the discussed works contains ideas and mechanisms that may applied to improve this work.

6.2 Fog Computing Frameworks

To the best of our knowledge, already existing contributions in the area of fog computing are often evaluated by the means of simulators along with artificially generated data, since there is still a lack of research testbeds which could be used to evaluate different mechanisms in fog computing. Therefore, in this section, we focus on the works which provide concrete implementations of fog architectures.

de Brito et al. [47] In the work of de Brito et al. [47], an IoT testbed is proposed based on Docker Swarm and the OpenMTC M2M Framework. Their architecture consists of two

main entities, a fog orchestration agent and a fog orchestrator. An orchestration mechanism for microservices in the fog environment is also discussed in their work. The basis for that mechanism are Docker labels which are used to check resource consumption constraints of applications. In the FogFrame framework, we explicitly discuss in what way a fog landscape is formed and how communication between different devices is established. In the work of de Brito et al., they propose self-announcement mechanism for establishing communication, i.e., when a fog device appears in the fog landscape, it announces itself either as a fog orchestration agent, or as a fog orchestrator. In our framework, we also do self-announcement, however, in our implementation the fog landscape is structured and hierarchical, and self-announced devices, i.e., both fog nodes and fog cells, can become members of certain fog colonies depending on different criteria, i.e., closest distance. With regard to orchestration, our approach is rather different to Docker labels, i.e., we formalize a system model, and introduce different service placement algorithms.

Tsai et al. [114]

Tsai et al. [114] implement a distributed analytical fog computing platform based on Raspberry Pis using TensorFlow and Kubernetes. Their testbed consists of a centralized server and fog devices connected by an Ethernet switch. The programming of the application model is performed by the means of TensorFlow, i.e., applications are split into small operators. Kubernetes controls and monitors the fog landscape, checks the available resources, and deploys containers of operators on-demand. In contrast, in our work, we have developed an own distributed management system for the fog landscape which is not centralized like Kubernetes. Different fog devices in the fog landscape have different functionality. Our fog nodes are lightweight compared to Kubernetes, and they monitor, manage and orchestrate their own fog colonies.

Yigitoglu et al. [162]

Yigitoglu et al. [162] introduce the fog computing framework Foggy. In the Foggy framework there is a three-tier infrastructure, namely edge devices, network infrastructure, and cloud services. Their network infrastructure tier consists of nodes and an orchestration server. The orchestration server creates and maintains a resource catalog which contains data about the available resource pool, i.e., capacities, connections, and utilization. The authors also propose different strategies to promote reliability when performing deployments of services. The testbed is implemented based on Raspberry Pis. Resource provisioning in their work is performed by an orchestration server, which runs on every node in the network, and implements a first-fit provisioning method. Compared to our work, we distinguish between different types of nodes, i.e., fog cells and fog nodes. Our fog nodes perform orchestration in the own colonies. Hence, we do not have only one orchestration server for a fog landscape, but many interconnected fog colonies. We consider multiple fog colonies and the hierarchical structure of a fog landscape; this is not foreseen by Yigitoglu et al. We focus on specific communication and deployment mechanisms. And in our work different resource

optimization methods are implemented.

He et al. [70] He et al. [70] propose a simulated fog computing model introducing a static dedicated and volatile *opportunistic* fog landscape as well as fog masters and fog workers as the main entities in a fog landscape resembling our fog nodes and fog cells. Similarly to our approach, the presented model enables multiple fog masters in one fog environment. In their pairing mechanism, He et al. consider invitations from fog workers in order to enter the fog landscape, while in our work fog cells and fog nodes perform self-announcement. Even though their system is simulated, He et al. provide very interesting insights on interactions within different fog environments. A multi-tier fog computing model is simulated. Dedicated fogs are static, while opportunistic fogs are volatile and consist of various computational, storage and networking resources which may enter or leave the fog. Considering the question of how to form a fog landscape, in opportunistic fogs, fog devices are joined by invitation from fog masters. Each fog has at least one fog master. An interesting aspect is that one fog can have multiple fog masters to improve reliability. However, it is not described in detail, how the coordination and management is performed by those multiple fog masters. Compared to our work, FogFrame is not a simulation framework, but a real-world fog computing framework. Fog masters resemble fog nodes in FogFrame, and fog workers resemble fog cells. We also enable multiple fog nodes in one fog colony. In order to enter the fog landscape, we use self-announcement mechanism, while He et al. propose an invitation mechanism.

Battulga et al. [17] Battulga et al. [17] introduce the *FogGuru* platform for fog computing implemented via a real-world testbed. Their representative fog landscape is built out of five Raspberry Pis united in a cluster and a cloud tier. The cloud is utilized to host a static service to process sensor data. Their system utilizes a publish-subscribe mechanism to push sensor data through a stream processing system and further into the cloud tier. For orchestration purposes, Docker Swarm is used, and one of the five Raspberry Pi units is used as a Swarm Manager. Unlike our work, the work of Battulga et al. shows how to utilize a publish-subscribe mechanism in fog computing. Their testbed is static, unlike ours, where we explicitly tackle runtime operation events in the fog landscape and migrate necessary services when needed.

Buyya et al. [99] In the work of Buyya et al. [99], alongside with a simulated environment via iFogSim, the authors implement a small static testbed of eight smartphones as IoT devices and five standard computers that act together as a fog cluster of resources interconnected with LAN. This fog cluster operates within their framework called *FogBus*. Service placement is based on a time-optimized QoS-based policy and follows application deadlines. For this, a heuristic evolutionary algorithm to create a placement map of applications onto available resources in the fog cluster is applied. To address possible failures of resources within the fog cluster, in their work a replication mechanism is provided. Compared to their work, our proposed

framework ensures cooperation between different fog colonies. The fog landscape automatically detects if devices appear or disappear from the fog landscape, and places, migrates, and optimizes services accordingly.

Xu and Zhang [156]

Another Raspberry Pi-based testbed called *piFogBed* is presented by Xu and Zhang [156]. Their system has a coordinator deployed on a standard computer that contains user management functionality, a device allocator for service placement of user applications, a container manager to save service images to DockerHub, a network simulator and an application execution controller. Fog nodes are deployed on four Raspberry Pi units and execute applications. Service placement is implemented in a set of policies that ensure the utilization of closest fog nodes until their capacities reach a certain threshold and taking into account bandwidth and delay constraints. Their work is a good example of holistic and detailed experiments. Compared to the work of Xu and Zhang, we consider multiple fog colonies that utilize a decentralized service placement for application execution.

Hasenburg et al. [68]

The fog computing testbest *MockFog* is presented by Hasenburg et al. [68]. Their system emulates fog computing entities and infrastructure in the cloud, and implements and demonstrates an interesting approach of executing IoT applications in the fog with a predefined orchestration schema. The fog infrastructure module of MockFog allows modeling the needed fog resources. The application management defines the IoT applications requirements, and allows specifying containers of applications and assign where to deploy them referencing the needed fog resources. Last but not least, the experimentation module allows users to create, configure, and establish different scenarios of experiments. Emulations in MockFog provide network graphs with specified connections and parameters. FogFrame operates in a real-world volatile fog, comprising from the devices at the network edge and in the cloud, and where devices can enter or disappear from the fog landscape. MockFog's application management with network graphs and connections would be interesting to use in FogFrame, as in the current implementation of FogFrame IoT applications with their parameters are specified with API calls, telling to FogFrame where to get each service image and what are the application demands.

Cech et al. [38]

An interesting combination of blockchain and fog computing technologies is proposed in the work of Cech et al. [38]. Their decentralized architecture is based on MultiChain nodes embedded in more powerful fog cells and a P2P network. This P2P network overlay provides a distributed data store to share sensor data between resources in a fog landscape. The authors implement a testbed with three Raspberry Pi units connected to a standard computer. Docker Swarm is used for the orchestration of applications. Blockchain functionality is implemented by a Docker image of blockchain based on the MultiChain framework. In our work, we focus on the fog landscape itself, on how it is formed, how the communication is performed between different fog colonies, and how the volatility is tackled. The mechanism of Cech et al. could be applied for a distributed data store, as well as to enable tracking of application execution around the fog landscape.

Varshney and Simmhan [143]

A conceptual work by Varshney and Simmhan [143] describes coordination models to be applied in fog computing. In their work, three different coordination models in fog computing are considered, i.e., hierarchical, P2P, and hybrid. It is emphasized that the hierarchical model enables only vertical communication. The cloud is defined as the root of this hierarchy and is responsible for all the coordination, orchestration, and execution of applications. In the P2P model, horizontal communication is considered between different devices in the fog landscape. Such communication has to be set up by a central entity, which can be located either in the cloud or at the edge of the network, and which has a global view on the overall pool of resources. The network topology is to be maintained by the means of a distributed hash table. In a hybrid coordination model, the authors consider both horizontal communication between different resources as well as vertical communication to establish an ordered fog landscape infrastructure. We place our work in such a hybrid model. In FogFrame, the central entity to form and maintain a fog landscape is the fog controller. Fog colonies are interconnected and collaboratively can recover from failures and execute applications even if the fog controller is not available. With regard to fog landscape operation, Varchney and Simmhan propose to change the coordination model depending on changes of the fog landscape at runtime.

Zhang et al. [169]

Zhang et al. [169] propose a conceptual regional cooperative fog computing architecture. The authors distinguish between edge and fog layers, and a cloud center. The fog layer is coordinated by a separate *local coordination server*. The cloud center is a *super fog server*, however the coordination is in the exclusive responsibility of the local coordination server. The authors focus on cooperation in the fog landscape in terms of service migration, maintaining uninterrupted service and reliability. With regard to communication, intra-fog and inter-fog communication and management are envisioned. The virtualization mechanism proposed in their work is by the means of VMs. Compared to our work, we use containers instead of VMs, since VMs are heavyweight and therefore not suited for rather lightweight IoT devices [103]. Similarly to Zhang et al., we also do not design any coordinating fog landscape entities in the cloud. A local coordination server resembles a fog node being the head of a fog colony. However, we provide communication and collaboration between different fog colonies by the means of their corresponding fog nodes.

Karagiannis et al. [87]

Karagiannis et al. [87] propose an alternative to the hierarchical structure of organizing the fog. The authors introduce self-organizing fog nodes that enable flat network modeling as well as improved fault tolerance compared to hierarchical fogs. Such fog nodes maintain connectivity with the neighbor fog nodes, and share their resources to execute collaboratively IoT applications. The self-organizing fog nodes form groups of fog devices. One node can belong to multiple groups. Fog nodes are added to the groups by their proximity to other fog nodes. Karagiannis et al. also provide an organization algorithm for fog nodes. In this dissertation,

the fog is hierarchical within a fog colony. However, when considering multiple fog colonies in a fog landscape, the approach of Karagiannis et al. could be beneficial to help choosing a neighbor fog colony where to delegate application requests.

Shneidman et al. [129]

In the technical report of Shneidman et al. [129], the authors provide an infrastructure called Hourglass that is meant to be deployed over an established network and to enable management of data flows in applications. Interestingly, Hourglass operates in terms of circuits, i.e., connected data producers and consumers together with data flow links between them. Circuits correspond to the DDF model discussed in Section 2.5.1. However, Hourglass has an own XML-based description language to define all the parameters, semantics, and endpoints. Services are either data producers or consumers. There is also a mechanism for announcement of services and circuits with a distributed registry. Hourglass is implemented as a testbed over a simulated distributed system. All in all, compared to FogFrame, Hourglass's aim is to make an application's circuit functional in the volatile network, while FogFrame aims to enable both: infrastructure of a functioning fog landscape and the means and algorithms to efficiently distribute IoT applications tackling similar issues of volatility, mobility and geography. Some concepts could be borrowed from this work instead of the DDF model, e.g., FogFrame could benefit from providing more complex definitions of IoT applications, for example, from their concept of topics and predicates to describe relations between services and circuits. The circuit manager of Hourglass somewhat corresponds to FogFrame runtime event policies within the reasoning service. In Hourglass, circuits are initiated by an application, i.e., it should implement a certain conforming compatible wrapper. Then, an application needs to find and decide from where to receive data and establish and maintain corresponding circuits with its circuit manager. In FogFrame, an IoT application consists of multiple services which are deployed over available resources in a fog colony, in this case, an application cannot choose anything, the distribution is done on the fog level in the head fog nodes according to the calculated service placement plan.

Wang et al. [147]

In the work of Wang et al. [147], the containerized resource management framework FogBus2 is built based on the new lightweight technology K3s[1]. FogBus2 implements different scheduling algorithms to execute IoT applications combining resources in the fog and the cloud. K3s is a backbone for the framework consisting of a master server deployed in the cloud and agents in the fog. In each pod only a single component from FogBus2 is embedded. Each pod of K3s is assigned to a different node thus showcasing workload balancing. One stated challenge by the authors is that components need to bind their IP addresses before the system can execute applications. Indeed, the K3s platform could provide FogFrame with a similar backbone to establish networking. Nevertheless, resource provisioning in FogFrame is integrated into the fog nodes. Also, in FogFrame we tackle volatility by triggering different policies corresponding to different events. Service registry

[1]Online; Accessed: Apr. 2023 `https://k3s.io/`

Table 6.1: Overview of implementations of fog architectures.

Work	Implemented	VMs	Containers	Centralized Reasoning	Distributed Reasoning	Intra-fog Communication	Inter-fog Communication
Battulga et al. [17]	✓	✓	✓	✓		✓	
Cech et al. [38]	✓		✓	✓		✓	
de Brito et al.[47]	✓		✓		✓	✓	
He et al. [70]					✓	✓	
Buyya at al. [99]	✓	✓		✓		✓	
Tsai et al. [114]	✓	✓		✓		✓	
Vögler et al.[145]	✓	✓	✓	✓		✓	
Xu and Zhang [156]	✓	✓	✓	✓		✓	
Yigitoglu et al. [162]	✓		✓	✓		✓	✓
Zhang et al.[169]		✓			✓	✓	✓
FogFrame	✓	✓	✓		✓	✓	✓

is distributed, i.e., whenever a service is registered, it is propagated to all fog nodes. This is not available in FogBus2. Similar statement could be made about OpenYurt[2]: on the infrastructure level it could enable networked devices, however all the services of FogFrame for establishing fog colonies with devices entering and leaving the fog and the execution of IoT applications remain relevant.

Summary The findings from the related fog computing architectures are summarized in Table 6.1. This table provides insights for the most relevant considered work of whether it is implemented or simulated, makes use of VMs and containers, is a centralized solution or distributed, accounts for the communication within a dedicated fog computing environment, and considers the communication between multiple fog colonies.

In this dissertation, we model a conceptual fog landscape and an IoT application to be executed by the means of fog resources. Based on these preliminaries, we formulate the FSPP, which aims to maximize the utilization of fog resources and the adherence to QoS parameters. We simulate a fog landscape and solve the FSPP by a first-fit algorithm, a GA, and an exact optimization method. In general, the contributions in resource provisioning and service placement discussed in Section 6.1 differ from our work in terms of a system model for service placement in the fog landscape, parameters which are included in this model, and algorithms to provide a solution for this model. We have implemented these mechanisms in FogFrame and evaluated them in a real-world testbed.

[2]Online; Accessed: Apr. 2023 `https://openyurt.io/`

With regard to the framework implementation, we provide design details and workflows in a fog landscape, eliminate the usage of simulators and implement a representative real-world Raspberry Pi-based testbed with the FogFrame framework. The framework

- Introduces mechanisms to create a fog landscape and account for its volatile nature,
- Provides decentralized application execution in multiple fog colonies,
- Discusses communication mechanisms between different fog colonies,
- Introduces a service placement problem formulation to account for practical issues dealing with delegating and deployment of applications,
- Implements a greedy algorithm and a GA to solve the service placement problem, and
- Reacts to runtime events in the fog landscape and migrates necessary services to balance workload between resources.

We extensively evaluate the framework with regard to deployment times of services and utilization of resources.

CHAPTER 7

Conclusion

In this last chapter, we aim to outline key takeaways from this thesis. In Section 7.1, we provide answers to the research questions as specified in Section 1.1 and summarize the main contributions demonstrating what new artifacts, models, and methods have been developed, and how they advance the already existing research in the area of resource provisioning in fog computing. After that, we shape topics and possible research directions towards future work in Section 7.2.

7.1 Summary of Contributions

This thesis aims to research, design, implement, and evaluate concepts, artifacts, and methods to enable resource provisioning in fog computing. There are two main views considered in this research: IoT applications and novel computational environments. Regarding IoT applications, the modern use of ubiquitous IoT devices generates an enormous high volume of data and is characterized by the high velocity and variety of data. This imposes new challenges to the ways of how to process data, e.g., how to optimize processing, how to adhere to the variety of IoT devices emitting data, as well as how to execute IoT applications more efficiently. This results in new requirements towards processing architectures, and thus new computational environments emerge, i.e., fog computing. Fog computing combines various heterogeneous resources at the edge of the network and in the cloud, and aims to implement new concepts, workflows, and architectures to fulfill IoT demands.

In this dissertation, we begin with the discussion of key enablers of fog computing, i.e., the IoT and cloud computing. We discuss what common features and approaches of these major technology trends are the conditions for fog computing, and which silos need to be addressed. We discuss several RAs that are relevant to further advance the research in fog computing, but which do not provide to the

Chapter 2

needed extent the functionalities for resource provisioning in the fog. We discuss the MEC framework, OpenFog, and FORA RAs to show which main levels of management and pillars of requirements need to be considered when working with the IoT. Next, to enable IoT application execution in the fog, we consider how to represent computational resources in fog computing. For this, we discuss container technologies aimed at both wrapping fog functionalities and enabling the decentralized execution of IoT applications. Last but not least, after the review of infrastructure topics and enablers in fog computing, we devote attention towards providing a holistic background on resource provisioning, giving insights about existing approaches towards enabling resource provisioning in distributed environments.

Chapter 3 In the main part of this dissertation, first, we provide concepts and functionalities of a fog landscape. We discuss which entities have to be implemented to realize the fog as well as means-end relationships between those entities. We introduce the concept of fog colonies, which aim to act as mini data centers executing IoT applications. We show a fog landscape consisting of multiple fog colonies, and fog colonies consisting of fog nodes and multiple fog cells with attached IoT devices. In order to enable the cloud-to-thing continuum in a fog landscape, cloud resources complement fog colonies. This organization of a fog landscape is hierarchical and enables the decentralization of IoT application execution, where each fog colony becomes a geo-specific computational environment. To implement such a hierarchical fog landscape, we introduce a fog computing model, its components and communication details. The essential part of this model is the reasoning capability of fog nodes. Fog nodes perform resource provisioning and service placement. We provide our first system model for resource provisioning that aims to maximize utilization of fog colony's resources while executing IoT services. To evaluate the system model, we create a fog landscape by simulating desired computational resources and their parameters. The introduced fog landscape entities and the fog computing architecture constitute Contribution I of this thesis.

Chapter 4 Second, we continue to advance into the topic of resource provisioning in fog computing. In order to work with different optimization and heuristic methods to implement resource provisioning, we focus on the holistic specification of a resource model of a fog landscape as well as on a model of IoT applications to be executed in the fog. We use these notions to formalize the FSPP. A solution to the FSPP provides an optimized decision on how to place services onto fog resources, either in fog colonies or in the cloud. We set the goal of the FSPP to efficiently execute IoT applications, i.e., according to the desired QoS. Constraints in the FSPP include adherence to capabilities and capacities of fog resources, e.g., utilization of CPU, RAM and storage, and to demands of applications, e.g., constraints on deployment and response times. This model differs from the previous model in the first part of the dissertation by new decision variables and

new constraints to enable delegation of services to other fog colonies in the case when the current fog colony has got not enough resources to execute services. Two solutions are proposed to the FSPP: first-fit heuristics and exact optimization. The evaluation of the proposed resource provisioning approach is performed by the means of simulating the fog.

Next, we advance into introducing one more approach, namely a GA to solve the FSPP. This approach introduces fog chromosomes as a solution to the FSPP, where each gene in a chromosome corresponds to one assignment of a service onto a specific fog resource in the fog colony. Chromosomes keep the information about utilization of resources in a fog landscape as well as QoS demands of submitted IoT applications. We calculate the specified FSPP goal as well as apply constraints within each chromosome's fitness function. We evaluate the GA within the same fog landscape as during the exact optimization. Our findings show that the FSPP optimization prevents violations of QoS. The GA is more efficient in balancing load on fog resources in the fog colony by delegating more services to the neighbor colony, while the exact optimization utilizes to the allowed maximum the resources in the fog colony and does less delegations to the neighbor colony. The GA produces solutions which on average experience lower deployment delays by the cost of exploiting more cloud resources. The FSPP together with the implemented heuristic approaches and the exact optimization constitute Contribution II of this thesis.

Chapter 5

Advancing into the topic of resource provisioning needs insights from using a real-world fog. Working purely theoretically and evaluating results with simulations does not reveal the big picture of problems and challenges that are coming from the fog. Therefore, we design and implement FogFrame, which is a fog computing framework aimed to enable efficient resource provisioning in the fog. By the means of this framework, an operating fog landscape is created and maintained. We describe main workflows in the framework to show how fog colonies are formed, how communication is established between different resources, and how services are executed in fog colonies. Application management is enabled in the framework, it shares workload between different resources in the fog, and deploys and executes IoT applications. The investigated resource provisioning approaches and the formalization of the FSPP from Contribution II are adjusted according to the real-world fog landscape requirements and implemented in FogFrame. During simulations from Contribution II, the challenge to track applications in the fog was not evident, however, in the real-world testbed created in this contribution, this issue evolved into an additional set of constraints in the FSPP, which was accordingly extended and implemented. The evaluation of resource provisioning is performed via the real-world fog computing testbed, created from multiple single-board computers, i.e., Raspberry Pis with attached sensors, and supplemented with cloud resources. We have developed a real-world application to sense and process data on different levels in the fog. The underlying hardware

of the real-world fog landscape is monitored and the monitoring data is analyzed to show how the workload is spread between different devices, as well as how the framework reacts to the runtime volatility of the fog. Having implemented FogFrame, we introduce and develop according application migration mechanisms allowing FogFrame to dynamically react to those volatile changes in the fog, i.e., failures and overloads of devices, appeared new devices or disconnected devices in the fog. All the considered mechanisms are implemented and extensively evaluated. The framework that operates a real-world fog landscape together with the implemented resource provisioning approaches constitutes Contribution III of this dissertation.

At this point, it is beneficial to revisit the research questions as described in Section 1.1 that shaped the research plan over this thesis. In the following, we summarize how the presented contributions answer these questions.

Research Question I

How can heterogeneous devices at the edge and the cloud be utilized together to enable fog computing?

This dissertation provides concepts and artifacts to represent heterogeneous IoT devices at the edge of the network, as well as functionalities and workflows needed to utilize them in the fog. We show how coordinated control is established over the physical and virtual IoT infrastructure at the edge together with additional resources from the cloud in order to efficiently execute IoT applications. We address the issue of virtualization and implement mechanisms and wrappers of functionalities to reflect different levels of resources in the fog with fog colonies. These entities are fog cells and fog nodes. Fog nodes within fog colonies manage and maintain the fog network organization within fog colonies, tackle the volatility and communicate with other fog colonies. Fog cells deploy and execute services from IoT applications. Since the edge and the cloud are different computational environments in a fog landscape, the deployment mechanisms for services differ as well. Together with the conceptual artifacts, we solve practical problems of deploying multiple service instances on a single fog device. The novelty compared to the state of the art is in the introduced functionalities, capabilities, and communication within and between fog colonies.

Research Question II

How can the execution of IoT applications in a fog landscape be optimized for resource efficiency and adherence to QoS and SLAs?

The focus in this thesis is on resource provisioning approaches in fog computing. We formulate the FSPP with its goals and constraints, among which we introduce novel relationships and interactions needed to provide a functioning fog landscape. We explicitly consider how to delegate IoT applications in the fog and the challenges around it. As an additional outcome, the FogFrame framework is publicly available and can be freely used in the research community to develop and evaluate different resource provisioning methods within a real-world fog.

Research Question III

How to achieve a highly available, consistent, and fault-tolerant real-world fog landscape?

We design and build a real-world fog landscape based on lightweight containers that aim at efficient resource provisioning, fault-tolerant and decentralized IoT application execution. While most fog environments in research are simulated or are static, in FogFrame we address specific processes of how to instantiate the fog from a single fog device, and how to populate fog colonies device-by-device, which parameters need to be considered to enable communication, and how to pair devices. The FogFrame framework together with resource provisioning approaches accounts for availability of resources in the fog by efficiently balancing the load. Consistency is provided by various fall-back mechanisms in a fog landscape that ensure that the volatility of the fog does not crash the deployed IoT applications. Fault tolerance is ensured by implementing approaches to keep the record of the current fog landscape state and migrate services in the case when runtime events in the fog cause failures and change network constellations.

To summarize the contributions, their value in the state of the art research has still a significant weight. Today, we would not be in the position to start research with only few state of the art works from which to derive new requirements, as was the case in 2016. This would allow to concentrate the research on other topics, to use already enabled fog environments or frameworks, or to focus more on resource provisioning or other relevant topics as discussed in the next section. Nevertheless, FogFrame remains a decent framework, it is not a simulation but a real-world testbed aimed to investigate resource provisioning and service placement in the fog. Resource provisioning algorithms remain relevant. However, a possibility to run new possibly proprietary solvers within IoT devices should be further investigated. What could be improved - is the FSPP definition, for example, by adding more constraints and goals for optimization from other related works.

7.2 Opportunities for Future Work

Future research can be dedicated to advancing resource provisioning models and approaches for fog computing, as well as to working with FogFrame introducing new artifacts and mechanisms to manage the fog, to enhance monitoring of resources and the IoT application lifecycle, or to introduce new levels of optimization of the fog infrastructure, thus enabling new patterns in inter- and intra-fog communication.

Multiple Fog Landscapes

In this dissertation, we instantiate and execute IoT applications with a fog landscape consisting of several fog colonies. It is a promising research topic to investigate a meta control layer over the fog to enable not only communication between fog colonies, but also between multiple fog landscapes. Such additional coordination could be established either within the fog controller or be enabled by some new component located in the cloud for additional meta control layer in order to orchestrate multiple fog landscapes. Various components of the introduced framework architecture can be substituted or extended, e.g., resource provisioning and service placement methods or conceptual workflows on how to pair devices in order to create the fog. FogFrame has intentionally been designed and implemented in a loosely-coupled manner in order to enable modularity and extendability.

Reorganization

A particular issue to address in the future work is how to scan and reconfigure the fog landscape network organization based on different runtime events. When devices appear and disappear in a fog landscape, the resulting network constellation may become not optimal in terms of latency, bandwidth, network hops, location mapping, and connection preservation. Another promising improvement is adopting the ETSI standard on context information management and NSGI-LD API metadata [58] within the implemented API of the FogFrame framework. This would unambiguously enable geographic location queries, temporal data, and linked data coming from different sources. Currently, in the configuration of all our fog devices in a fog landscape, we define a fall-back parent and grandparent fog nodes. In the case when a head fog node is not available and a fog resource needs to enter a fog colony, then it will connect with an already known fall-back parent. Replacing already connected head fog nodes has not been considered. However, such a mechanism can be implemented: If a fog controller detects that one of head fog nodes is disconnected, this should trigger a reconfiguration event in the corresponding fog colony. In the same way as the head fog node has means to detect disconnected devices and tackle such events with a corresponding policy, the fog controller should be enabled with the means to detect disconnected head fog nodes of the colonies with the possibility to reassign the head and reconfigure the fog colony. We have a mechanism to track 'children' of each fog node, therefore, we would need to enable a fog controller to track children fog nodes of each head fog node, and then to reconnect to one of the existing fog nodes. From the children perspective, each fog device in case the head fog node

disappears will have to receive a new parent. A concept for choosing priorities for choosing a new parent should be considered in future work. Therefore, optimizing fog landscape topologies is a promising research challenge, i.e., to establish a meta-level of infrastructure for additional optimization and reconfiguration [86].

New Delivery Models

Another possible research direction is the investigation and implementation of different application and delivery models that facilitate the development of IoT applications to be executed in the fog, i.e., helping developers to functionally split IoT applications into smaller instances or services and parametrize their capabilities and resource demands [69]. In this work, we followed the DDF application delivery model, however, other techniques should be investigated too [13].

Creating the Fog

Having addressed the question of how to create a fog landscape, it is possible to use these results as the basis for developing other communication and application management mechanisms. Regarding communication, the framework enables easy integration of new components according to well-defined interfaces. For example, we can consider enabling each head fog node in a fog colony to keep not only one closest neighbor nog node, but a list of multiple neighbors. This will require additional mechanism to choose a neighbor for delegating application requests. Regarding the application management, other resource provisioning and service placement algorithms can be introduced to perform optimization according to different goals, e.g., with regard to different cost models of fog resources or energy efficiency [7, 88, 106].

Application Mobility

Another research issue is that in the current implementation of the framework, we do not consider delegating a single service from an application to a neighbor fog colony. By delegating the whole application, we avoid high intra-application latency and unnecessary coordination between fog colonies on the level of service requests. It remains a matter of future work to identify approaches to address mobility of the fog and enable various service execution tracking scenarios in a fog landscape [24, 53]. Regarding application execution, service descriptions with dynamic workload are not supported in the most cloud platforms offering only fixed capacities of their services. One of the solutions to tackle this issue in future may be regular update of the application model, e.g., daily, to adjust the load according to operation measurements of services.

Service Types

In the evaluation of FogFrame, we implemented several types of services to be executable in the framework and corresponding to the processes of sensing and processing data both at the edge and purely in the cloud. The framework demonstrates how different service types can be deployed in the cloud and at the edge. In real-world scenarios, service types can be even more heterogeneous depending on the purpose and IoT device equipment, and therefore there is a need in functionalities to accommodate different IoT resources, protocols and architectures in the fog [95].

New Solvers While the conducted experiments have shown that FogFrame is able to serve its purpose to provide coordinated control of a fog landscape and to execute applications, there are nevertheless some limitations with regard to the framework itself. A considerable limitation is that the software to be executed on fog resources have to be adapted or reimplemented according to the processor architecture of the hardware of IoT devices, e.g., according to the ARM processor architecture in a Raspberry Pi. This deals with both internal services of FogFrame as well as applications submitted by the users. For example, dynamic programming solvers which can be used in service placement, such as IBM CPLEX solver, Java ILP, or Gurobi, have not yet provided library distributions runnable on the devices with the ARM processor architecture so far. This problem is not specific to our work, but is a common problem when using fog infrastructure.

Machine Learning The proliferation of IoT data resulted in high popularity of Big Data topics and ML for various use cases. To combine fog computing and ML, not only ML as part of IoT applications can be investigated, but also to facilitate fog computing itself, e.g., for resource management in the fog, for predictions of failures of devices, for predictions of user behavior and arrivals of application requests, or for predictions of QoS violations and security issues. However, when intending to apply and run ML algorithms on fog devices, similar limitations apply, as have been described above: the ARM processor architecture of most IoT devices as well as limited computational, energy, and storage capacities highly limit the usage options of ML models and their types in production runtime. ML models trained conventionally can be used for the high-level organization of fog colonies in a fog landscape or between fog landscapes, for example, for finding best network configurations [4].

Fog Security Fog computing is still a developing research area in terms of security mechanisms. In this particular work, the private network in which the fog landscape operates is deemed to be secure, and all components of the fog landscape communicate via dedicated API endpoints on certain ports and IP addresses specified in the framework. Additionally, ingress and egress rules can be set up on each device allowing only specific framework-related interactions. Other software and hardware security mechanisms for fog computing need to be further investigated [121, 136].

Economical Impact Even though practical use cases, such as transportation, smart cities, smart buildings etc., of fog computing have been already researched and described in related work, there is still a big gap between those use cases being academic research and becoming a product available for industry. In order to marketize a product, issues of e-government and e-commerce, pricing for users, ownership and cost of investment related to fog computing need to be investigated. There has been very few works related to economical impact and issues of fog computing [1, 2, 3]. Most publications either focus on infrastructures, architectures and simulations, and scratch the surface of this topic. Several interesting articles have been published, for example, an overview of relevant needed portals and information

services is described within an end-2-end smart city platform [167]. The author suggests that with the help of fog computing the cost of digitization of regions for many use cases as well as corresponding investment costs can be drastically reduced. Kim et al. [90] introduce a market model game for fog computing theoretically describing a platform for brokering different dependencies between service providers, subscribers, fog resource owners.

To conclude, the opportunities of the future work are manifold both on infrastructure side of the fog and on functional side of the execution of IoT applications. The open topics include the research and development to enable the fog in its pre-life, early-life or functional-life, as well as resource provisioning approaches with different goals and constraints. FogFrame provides a foundation for other researchers to extend it with new functionalities and validate new approaches on different functional and technical levels.

Glossary

application is built following the microservice architectural approach from stateless services which need to be deployed and executed to achieve a certain result. 103

application request defines a set of services to be placed and deployed in the fog landscape together with QoS information for execution, for example, deadlines on application execution and processing times. 104

distributed application is an application formed from cooperating services, which exploit a virtual computational environment. 30

fog cell is a component of a fog colony that is an IoT device, has computational power and is able to host and execute arbitrary services. 42

fog colony is a micro data center made up from an arbitrary number of fog devices or so-called fog cells. 42

fog computing a horizontal, system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum. 2

fog controller is a mediator between fog colonies and the cloud in a fog landscape. 45, 46

fog device is a fog cell or a fog node, i.e., fog cells and fog nodes are two types of fog devices in a fog colony. 45

fog landscape a combination of IoT devices at the edge of the network and cloud resources that cooperate. 2

fog node is a component of a fog colony that manages and orchestrates fog cells; fog nodes are themselves extended fog cells and are able to host services and perform management activities, such as service placement and deployment. 45

service image is a not yet deployed service binary. 103

service placement plan is a result of calculation of a service placement algorithm; it contains decisions indicating the placement of each needed service on a specific resource in a fog landscape, namely, on a fog cell, fog node, in the cloud, and a decision about delegating the whole application to a neighbor fog colony. 104

service type is bound to the capabilities of the devices in the fog landscape, e.g., a service intended to receive temperature measurements can be deployed only on a fog cell with a temperature sensor attached, some services can be executed either in the cloud or in the fog, and other services can be executed only in the cloud. 103

smart system is a system that connects and processes data from distributed data sources while using already existing computational resources, decreasing processing latency, and offering the means to process data on-site, i.e., in a privacy-aware manner. 3

Acronyms

API Application Programming Interface. 24

CoAP Constrained Application Protocol. 25, 26
CPS Cyber-Physical System. 64
CRI Container Runtime Interface. 29

DDF Distributed Data Flow. 21

FaaS Function-as-a-Service. 142
FBP Flow-based Programming. 22
FORA Fog Computing for Robotics and Industrial Automation. 22
FSPP Fog Service Placement Problem. 6

GA Genetic Algorithm. 7

IIoT Industrial Internet of Things. 15
ILP Integer Linear Programming. 34
IoT Internet of Things. 1

M2M Machine-to-Machine. 25
MCC Mobile Cloud Computing. 21
MEC ETSI Mobile Edge Computing. 22
ML Machine Learning. 141

OCI Opensource Container Initiative. 28
OMA Open Mobile Alliance. 26
OPC UA Open Platform Communications Unified Architecture. 25

P2P Peer-to-Peer. 42

QoE Quality of Experience. 140
QoS Quality of Service. 2

RA Reference Architecture. 13

SLA Service Level Agreement. 2

TSN Time-Sensitive Networking. 25

VM Virtual Machine. 2
VNF Network Function Virtualization. 22
VNF Virtualized Network Function. 22

Bibliography

[1] Mohammad Aazam and Eui-Nam Huh. Dynamic Resource Provisioning Through Fog Micro Datacenter. In *12th IEEE International Workshop on Managing Ubiquitous Communications and Services (PerCom 2015)*, pages 105–110, St. Louis, MO, USA, 2015. IEEE. DOI: 10.1109/PERCOMW.2015.7134002.

[2] Mohammad Aazam and Eui-Nam Huh. Fog Computing Micro Datacenter Based Dynamic Resource Estimation and Pricing Model for IoT. In *29th IEEE International Conference on Advanced Information Networking and Applications (AINA 2015)*, pages 687–694, Gwangju, Korea, 2015. IEEE. DOI: 10.1109/AINA.2015.254.

[3] Mohammad Aazam, Marc St-Hilaire, Chung-Horng Lung, and Ioannis Lambadaris. MeFoRE: QoE based resource estimation at Fog to enhance QoS in IoT. In *23rd IEEE International Conference on Telecommunications (ICT 2016)*, pages 1–5, Thessaloniki, Greece, 2016. IEEE. DOI: 10.1109/ICT.2016.7500362.

[4] Karrar Hameed Abdulkareem, Mazin Abed Mohammed, Saraswathy Shamini Gunasekaran, Mohammed Nasser Al-Mhiqan, Ammar Awad Mutlag, Salama A. Mostafa, Nabeel Salih Ali, and Dheyaa Ahmed Ibrahim. A Review of Fog Computing and Machine Learning: Concepts, Applications, Challenges, and Open Issues. *IEEE Access*, 7:153123–153140, 2019. DOI: 10.1109/ACCESS.2019.2947542.

[5] Masoud Abedi and Mohammadreza Pourkiani. Resource Allocation in Combined Fog-Cloud Scenarios by Using Artificial Intelligence. In *5th IEEE International Conference on Fog and Mobile Edge Computing (FMEC 2020)*, pages 218–222, Paris, France, 2020. IEEE. DOI: 10.1109/FMEC49853.2020.9144693.

[6] Haftay Gebreslasie Abreha, Carlos J. Bernardos, Antonio De La Oliva, Luca Cominardi, and Arturo Azcorra. Monitoring in Fog Computing: State-of-the-Art and Research Challenges. *International Journal of Ad Hoc and Ubiquitous Computing*, 36(2):114–130, 2021. 10.1504/ijahuc.2021.113384.

[7] Shahbaz Afzal and G. Kavitha. Load Balancing in Cloud Computing – A Hierarchical Taxonomical Classification. *Journal of Cloud Computing*, 8(22):1–24, 2019. DOI: 10.1186/s13677-019-0146-7.

[8] Amal Al-Qamash, Iten Soliman, Rawan Abulibdeh, and Moutaz Saleh. Cloud, Fog, and Edge Computing: A Software Engineering Perspective. In *IEEE International Conference on Computer and Applications (ICCA 2018)*, pages 276–284, Beirut, Lebanon, 2018. IEEE. DOI: 10.1109/COMAPP.2018.8460443.

[9] Ziyad R. Alashhab, Mohammed Anbar, Manmeet Mahinderjit Singh, Yu-Beng Leau, Zaher Ali Al-Sai, and Sami Abu Alhayja'a. Impact of Coronavirus Pandemic Crisis on Technologies and Cloud Computing Applications. *Journal of Electronic Science and Technology*, 19(1):1–12, 2021. DOI: 10.1016/j.jnlest.2020.100059.

[10] Silvio Roberto Martins Amarante, Filipe Maciel Roberto, André Ribeiro Cardoso, and Joaquim Celestino. Using the Multiple Knapsack Problem to Model the Problem of Virtual Machine Allocation in Cloud Computing. In *16th IEEE International Conference on Computational Science and Engineering (CSE 2013)*, pages 476–483, Syndey, Australia, 2013. IEEE. DOI: 10.1109/CSE.2013.77.

[11] A.R. Arunarani, Dheeravath Manjula, and Vijayan Sugumaran. Task Scheduling Techniques in Cloud Computing: A Literature Survey. *Future Generation Computer Systems*, 91(4):407–415, 2019. DOI: 10.1016/j.future.2018.09.014.

[12] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer Networks*, 54(15):2787–2805, 2010. DOI: 10.1016/j.comnet.2010.05.010.

[13] Cosmin Avasalcai, Bahram Zarrin, and Schahram Dustdar. EdgeFlow — Developing and Deploying Latency-Sensitive IoT Edge Applications. *IEEE Internet of Things Journal*, 9(5):3877–3888, 2022. DOI: 10.1109/JIOT.2021.3101449.

[14] Kevin Bachmann. Design and Implementation of a Fog Computing Framework. Master's thesis, Vienna University of Technology (TU Wien), Vienna, Austria, 2017. Online; Accessed: Apr. 2023 `http://www.infosys.tuwien.ac.at/staff/sschulte/paper/Bachmann_Master.pdf`.

[15] Luciano Baresi and Danilo Filgueira Mendonça. Towards a Serverless Platform for Edge Computing. In *IEEE International Conference on Fog Computing (ICFC 2019)*, pages 1–10, Prague, Czech Republic, 2019. IEEE. DOI: 10.1109/ICFC.2019.00008.

[16] Richard Baskerville, Abayomi Baiyere, Shirley Gregor, Alan Hevner, and Matti Rossi. Design Science Research Contributions: Finding a Balance between Artifact and Theory. *Journal of the Association for Information Systems*, 19(5):358–376, 2018. DOI: 10.17705/1jais.00495.

[17] Davaadorj Battulga, Daniele Miorandi, and Cédric Tedeschi. FogGuru: a Fog Computing platform based on Apache Flink. In *23rd IEEE Conference on Innovation in Clouds, Internet and Networks and Workshops (ICIN 2020)*, pages 156–158, Paris, France, 2020. IEEE. DOI: 10.1109/ICIN48450.2020.9059374.

[18] Paolo Bellavista, Javier Berrocal, Antonio Corradi, Sajal K.Das, Luca Foschini, and Alessandro Zanni. A survey on fog computing for the internet of things. *Pervasive and Mobile Computing*, 52:71–99, 2019. 10.1016/j.pmcj.2018.12.007.

[19] Julian Bellendorf and Zoltán Ádám Mann. Classification of Optimization Problems in Fog Computing. *Future Generation Computer Systems*, 107:158–176, 2020. DOI: 10.1016/j.future.2020.01.036.

[20] David Bermbach, Jonathan Bader, Jonathan Hasenburg, Tobias Pfandzelter, and Lauritz Thamsen. AuctionWhisk: Using an auction-inspired approach forfunction placement in serverless fog platforms. *Software: Practice and Experience*, 52(5):1143–1169, 2022. DOI: 10.1002/spe.3058.

[21] David Bermbach, Frank Pallas, David Pérez, Pierluigi Plebani, Maya Anderson, Ronen Kat, and Stefan Tai. *A Research Perspective on Fog Computing*, pages 198–210. Springer, 2018. DOI: 10.1007/978-3-319-91764-1_16.

[22] David Bernstein. Containers and Cloud: From LXC to Docker to Kubernetes. *IEEE Cloud Computing*, 1(3):81–84, 2014. DOI: 10.1109/MCC.2014.51.

[23] Dinabandhu Bhandari, C. A. Murthy, and Sankar K. Pal. Variance As a Stopping Criterion for Genetic Algorithms with Elitist Model. *Fundamenta Informaticae*, 120(2):145–164, 2012. DOI: 10.5555/2594813.2594815.

[24] Luiz F. Bittencourt, Javier Diaz-Montes, Rajkumar Buyya, Omer F. Rana, and Manish Parashar. Mobility-Aware Application Scheduling in Fog Computing. *IEEE Cloud Computing Journal*, 4(2):26–35, 2017. DOI: 10.1109/MCC.2017.27.

[25] Flavio Bonomi, Rodolfo Milito, Preethi Natarajan, and Jiang Zhu. Fog Computing: A Platform for Internet of Things and Analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*, volume 546

of *Studies in Computational Intelligence*, pages 169–186. Springer, 2014. DOI: 10.1007/978-3-319-05029-4_7.

[26] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog Computing and Its Role in the Internet of Things. In *MCC Workshop on Mobile Cloud Computing (MCC 2012)*, pages 13–16, Helsinki, Finland, 2012. Elsevier. DOI: /10.1145/2342509.2342513.

[27] Eleonora Borgia. The internet of things vision: Key features, applications and open issues. *Computer Communications*, 54:1–31, 2014. DOI: 10.1016/j.comcom.2014.09.008.

[28] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescape. Integration of Cloud Computing and Internet of Things: a Survey. *Future Generation Computer Systems*, 56.

[29] Richard P. Brent. Efficient Implementation of the First-Fit Strategy for Dynamic Storage Allocation. *ACM Transactions on Programming Languages and Systems*, 11(3):388–403, 1989. DOI: 10.1145/65979.65981.

[30] Antonio Brogi and Stefano Forti. QoS-aware Deployment of IoT Applications Through the Fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, 2017. DOI: 10.1109/JIOT.2017.2701408.

[31] Björn Butzin, Frank Golatowski, and Dirk Timmermann. Microservices Approach for the Internet of Things. In *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016)*, pages 1–6, Berlin, Germany, 2016. IEEE. DOI: 10.1109/ETFA.2016.7733707.

[32] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility. *Future Generation Computer Systems*, 25(6):599–616, 2009. DOI: 10.1016/j.future.2008.12.001.

[33] Charles C. Byers. Architectural Imperatives for Fog Computing: Use Cases, Requirements, and Architectural Techniques for Fog-Enabled IoT Networks. *IEEE Communications Magazine*, 55(8):14–20, 2017. DOI: 10.1109/MCOM.2017.1600885.

[34] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A.F. De Rose, and Raykumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software Practice and Experience*, 41:23–50, 2011. DOI: 10.1002/spe.995.

[35] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. Optimal Operator Placement for Distributed Stream Processing

Applications. In *10th ACM International Conference on Distributed and Event-based Systems (DEBS 2016)*, pages 69–80, Irvine, CA, USA, 2016. ACM. DOI: 10.1145/2933267.2933312.

[36] Valeria Cardellini, Vincenzo Grassi, Francesco Lo Presti, and Matteo Nardelli. On QoS-aware scheduling of data stream applications over fog computing infrastructures. In *IEEE Symposium on Computers and Communication (ISCC 2015)*, pages 271–276, Larnaca, Cyprus, 2015. IEEE. 10.1109/ISCC.2015.7405527.

[37] Luca Catarinucci, Danilo de Donno, Luca Mainetti, Luca Palano, Luigi Patrono, Maria Laura Stefanizzi, and Luciano Tarricone. An IoT-Aware Architecture for Smart Healthcare Systems. *IEEE Internet of Things Journal*, 2(6):515–526, 2015. DOI: 10.1109/JIOT.2015.2417684.

[38] Hendrik L. Cech, Marcel Großmann, and Udo R. Krieger. A Fog Computing Architecture to Share Sensor Data by Means of Blockchain Functionality. In *IEEE International Conference on Fog Computing (ICFC 2019)*, pages 31–40, Prague, Czech Republic, 2019. IEEE. DOI: 10.1109/ICFC.2019.00013.

[39] Antorio Celesti, Davide Mulfari, Maria Fazio, Massimo Villari, and Antonio Puliafito. Exploring Container Virtualization in IoT Clouds. In *2th IEEE International Conference on Smart Computing (SMARTCOMP 2016)*, pages 1–6, St. Louis, MO, USA, 2016. IEEE. DOI: 10.1109/SMARTCOMP.2016.7501691.

[40] Min Chen, Shiwen Mao, and Yunhao Liu. Big Data: A Survey. *Mobile Networks and Applications*, 19(2):171–209, 2014. DOI: 10.1007/s11036-013-0489-0.

[41] Michele Compare, Piero Baraldi, and Enrico Zio. Challenges to IoT-Enabled Predictive Maintenance for Industry 4.0. *IEEE Internet of Things Journal*, 7(5):4585–4597, 2020. DOI: 10.1109/JIOT.2019.2957029.

[42] OpenFog Consortium. OpenFog Reference Architecture for Fog Computing, 2016. Online; Accessed: Apr. 2023 `https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_`.

[43] Vitor Goncalves da Silva, Marite Kirikova, and Gundars Alksnis. Containers for virtualization: An overview. *Applied Computer Systems*, 23(1):21–27, 2018. DOI: 10.2478/acss-2018-0003.

[44] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. Vivaldi: A Decentralized Network Coordinate System. *ACM SIGCOMM Computer Communication Review*, 34(4):15–26, 2004. DOI: 10.1145/1030194.1015471.

[45] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N. Calheiros, Soumya K. Ghosh, and Rajkumar Buyya. Fog Computing: Principles, Architectures, and Applications. In *Internet of Things: Principles and Paradigms*, chapter 4, pages 61–75. Morgan Kaufmann, 2016. DOI: 10.1016/B978-0-12-805395-9.00004-6.

[46] Suparna De, Payam Barnaghi, Martin Bauer, and Stefan Meissner. Service Modelling for the Internet of Things. In *Federated Conference on Computer Science and Information Systems (FedCSIS 2011)*, pages 949–955, Szczecin, Poland, 2011. IEEE. Online; Accessed: Apr. 2023 `https://annals-csis.org/proceedings/2011/pliks/113.pdf`.

[47] Matthias Santos de Brito, Saiful Hoque, Thomas Magedanz, Ronald Steinke, Alexander Willner, Daniel Nehls, Oliver Keils, and Florian Schreiner. A Service Orchestration Architecture for Fog-enabled Infrastructures. In *2nd International Conference on Fog and Mobile Edge Computing (FMEC 2017)*, pages 127–132, Valencia, Spain, 2017. IEEE. DOI: 10.1109/FMEC.2017.7946419.

[48] Ruilong Deng, Rongxing Lu, Chengzhe Lai, and Tom H. Luan. Towards Power Consumption-Delay Tradeoff by Workload Allocation in Cloud-Fog Computing. In *International Conference on Communications, Mobile and Wireless Networking Symposium (ICC 2015)*, pages 3909–3914, London, UK, 2015. IEEE. DOI: 10.1109/ICC.2015.7248934.

[49] Paolo Detti. A Polynomial Algorithm for the Multiple Knapsack Problem with Divisible Item Sizes. *Information Processing Letters*, 109(11):582–584, 2009. DOI: 10.1016/j.ipl.2009.02.003.

[50] Hoang T. Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches. *Wireless Communications and Mobile Computing*, 13:1587–1611, 2013. DOI: 10.1002/wcm.1203.

[51] Zoltán Ádám Mann. Resource Optimization Across the Cloud Stack. *IEEE Transactions on Parallel and Distributed Systems*, 29(1):169–182, 2018. DOI: 10.1109/TPDS.2017.2744627.

[52] Amnon H. Eden. Three Paradigms of Computer Science. *Minds and Machines*, 17(2):135–167, 2007. DOI: 10.1007/s11023-007-9060-8.

[53] Said El Kafhali, Chorouk Chahir, Mohamed Hanini, and Khaled Salah. Architecture to Manage Internet of Things Data Using Blockchain and Fog Computing. In *4th International Conference on Big Data and Internet of Things (BDIOT 2019)*, pages 1–8. ACM, 2019. DOI: 10.1145/3372938.3372970.

[54] ETSI. ETSI GS MEC 002 V.1.1.1 (2016-03) Mobile Edge Computing (MEC); Technical Requirements, 2016. Online; Accessed: Apr. 2023 https://www.etsi.org/deliver/etsi_gs/mec/001_099/002/01.01.01_60/gs_mec002v010101p.pdf.

[55] ETSI. ETSI GS MEC 003 V.1.1.1 (2016-03) Mobile Edge Computing (MEC); Framework and Reference Architecture, 2016. Online; Accessed: Apr. 2023 https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003/01.01.01_60/gs_MEC003v010101p.pdf.

[56] ETSI. ETSI Multi-access Edge Computing, 2016. Online; Accessed: Apr. 2023 https://www.etsi.org/technologies/multi-access-edge-computing.

[57] ETSI. ETSI GS MEC 011 V.1.1.1 (2017-07) Mobile Edge Computing (MEC); Mobile Edge Platform Application Enablement, 2017. Online; Accessed: Apr. 2023 https://www.etsi.org/deliver/etsi_gs/MEC/001_099/011/01.01.01_60/gs_MEC011v010101p.pdf.

[58] ETSI GS CIM 009. ETSI GS CIM 009 V1.1.1 - Context Information Management (CIM); NGSI-LD API, 2019. Online; Accessed: Apr. 2023 https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.01.01_60/gs_cim009v010101p.pdf.

[59] Niroshinie Fernando, Seng W. Loke, and Wenny Rahayu. Mobile Cloud Computing: A Survey. *Future Generation Computer Systems*, 29(1):84–106, 2013. DOI: 10.1016/j.future.2012.05.023.

[60] Iván Froiz-Míguez, Tiago M. Fernández-Caramés, Paula Fraga-Lamas, and Luis Castedo. Design, Implementation and Practical Evaluation of an IoT Home Automation System for Fog Computing Applications Based on MQTT and ZigBee-WiFi Sensor Nodes. *Sensors*, 18(8):1–42, 2018. DOI: 10.3390/s18082660.

[61] Thomas Frühwirth, Lukas Krammer, and Wolfgang Kastner. Dependability demands and state of the art in the internet of things. In *20th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2015)*, pages 1–4, Luxembourg, 2015. DOI: 10.1109/ETFA.2015.7301592.

[62] Dimitros Georgakopoulos, Prem Prackash Jayaraman, Maria Fazia, Massimo Villari, and Rajiv Ranjan. Internet of Things and Edge Cloud Computing Roadmap for Manufacturing. *IEEE Cloud Computing*, 3(4):66–73, 2016. DOI: 10.1109/MCC.2016.91.

[63] Ammar Gharaibeh, Mohammad A. Salahuddin, Sayed Jahed Hussini, Abdallah Khreishah, Issa Khalil, Mohsen Guizani, and Ala Al-Fuqaha. Smart

Cities: A Survey on Data Management, Security, and Enabling Technologies. *IEEE Communications Surveys and Tutorials*, 19(4):2456–2501, 2017. DOI: 10.1109/COMST.2017.2736886.

[64] Nam Ky Giang, Michael Blackstock, Rodger Lea, and Victor C.M. Leung. Developing IoT Applications in the Fog: a Distributed Dataflow Approach. In *5th International Conference on the Internet of Things (IoT 2015)*, pages 155–162, Seoul, Korea, 2015. IEEE. DOI: 10.1109/IOT.2015.7356560.

[65] Tom Goethals, Merlijn Sebrechts, Ankita Atrey, Bruno Volckaert, and Filip De Turck. Unikernels vs Containers: An In-Depth Benchmarking Study in the Context of Microservice Applications. In *8th IEEE International Symposium on Cloud and Service Computing (SC2 2018)*, pages 1–8, Paris, France, 2018. IEEE. DOI: 10.1109/SC2.2018.00008.

[66] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing Environments. *Software Practice and Experience*, 47:1275–1296, 2017. DOI: 10.1002/spe.2509.

[67] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network Function Virtualization: Challenges and Opportunities for Innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015. DOI: 10.1109/MCOM.2015.7045396.

[68] Jonathan Hasenburg, Martin Grambow, and David Bermbach. Mockfog 2.0: Automated execution of fog application experiments in the cloud. *IEEE Transactions on Cloud Computing*, (01):1–15, 2021. DOI: 10.1109/TCC.2021.3074988.

[69] Jonathan Hasenburg, Sebastian Werner, and David Bermbach. Supporting the Evaluation of Fog-Based IoT Applications During the Design Phase. In *5th Workshop on Middleware and Applications for the Internet of Things (M4IoT 2018)*, page 1–6, Rennes, France, 2018. ACM. DOI: 10.1145/3286719.3286720.

[70] Jianhua He, Jian Wei, Kai Chen, Zuoyin Tang, Yi Zhou, and Yan Zhang. Multi-tier Fog Computing with Large-scale IoT Data Analytics for Smart Cities. *IEEE Internet of Things*, 5(2):677–686, 2018. DOI: 10.1109/JIOT.2017.2724845.

[71] Thomas Hiessl, Vasileios Karagiannis, Christoph Hochreiner, Stefan Schulte, and Matteo Nardelli. Optimal Placement of Stream Processing Operators in the Fog. In *3rd IEEE International Conference on Fog and Edge Computing (ICFEC 2019)*, pages 1–10, Larnaca, Cyprus, 2019. IEEE. DOI: 10.1109/CFEC.2019.8733147.

[72] Christoph Hochreiner, Michael Vögler, Stefan Schulte, and Schahram Dustdar. Cost-Efficient Enactment of Stream Processing Topologies. *PeerJ Computer Science*, 3(e141):1–36, 2017. Online; Accessed: Apr. 2023 `https://peerj.com/articles/cs-141/`.

[73] Christoph Hochreiner, Michael Vögler, Johannes M. Schleicher, Christian Inzinger, Stefan Schulte, and Schahram Dustdar. Nomadic Applications Traveling in the Fog. In *Cloud Infrastructures, Services, and IoT Systems for Smart Cities (CN4IoT 2017)*, pages 151–161, Bridnisi, Italy, 2017. Springer. DOI: 10.1007/978-3-319-67636-4_17.

[74] Christoph Hochreiner, Michael Vögler, Philipp Waibel, and Schahram Dustdar. VISP: An Ecosystem for Elastic Data Stream Processing for the Internet of Things. In *20th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2016)*, pages 1–11, Vienna, Austria, 2016. IEEE. DOI: 10.1109/EDOC.2016.7579390.

[75] Philipp Hoenisch, Ingo Weber, Stefan Schulte, Liming Zhu, and Alan Fekete. Four-fold Auto-Scaling on a Contemporary Deployment Platform using Docker Containers. In *13th International Conference on Service Oriented Computing (ICSOC 2015)*, volume 9435 of *LNCS*, pages 316–323. Springer, 2015. DOI: 10.1007/978-3-662-48616-0_20.

[76] Cheol-Ho Hong and Blesson Varghese. Resource Management in Fog/Edge Computing: A Survey on Architectures, Infrastructure, and Algorithms. *ACM Computing Surveys*, 52(5):1–37, 2019. DOI: 10.1145/3326066.

[77] Kirak Hong, David Lillethun, Umakishore Ramachandran, Beate Ottenwälder, and Boris Koldehofe. Mobile Fog: A Programming Model for Large–Scale Applications on the Internet of Things. In *1st ACM SIGCOMM Workshop on Mobile Cloud Computing (MCC 2013)*, pages 15–20, Hong Kong, China, 2013. DOI: 10.1145/2491266.2491270.

[78] John N. Hooker and Mauricio A. Osorio. Mixed logical-linear programming. *Discrete Applied Mathematics*, 96-97:395–442, 1999. DOI: 10.1016/S0166-218X(99)00100-6.

[79] Arunima Hota, Subasish Mohapatra, and Subhadarshini Mohanty. Survey of Different Load Balancing Approach-Based Algorithms in Cloud Computing: A Comprehensive Review. In Himansu Sekhar Behera, Janmenjoy Nayak, Bighnaraj Naik, and Ajith Abraham, editors, *International Conference on Computational Intelligence in Data Mining (CIDM 2017)*, pages 99–110, Singapore, 2019. Springer. DOI: 10.1007/978-981-10-8055-5_10.

[80] Pengfei Hu, Sahraoui Dhelim, Huansheng Ning, and Tie Qiu. Survey on Fog Computing: Architecture, Key Technologies, Applications and Open

Issues. *Journal of Network and Computer Applications*, 98:27–42, 2017. DOI: 10.1016/j.jnca.2017.09.002.

[81] Xin Huang, Sivakumar Ganapathy, and Tilman Wolf. Evaluating Algorithms for Composable Service Placement in Computer Networks. In *IEEE International Conference on Communications (ICC 2009)*, pages 2276–2281, Dresden, Germany, 2009. IEEE. DOI: 10.1109/ICC.2009.5199007.

[82] IEEE. IEEE 1934-2018 - IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing, 2018. Online; Accessed: Apr. 2023 `https://standards.ieee.org/standard/1934-2018.html`.

[83] Haris Isakovic, Denise Ratasich, Christian Hirsch, Michael Platzer, Bernhard Wally, Thomas Rausch, Dejan Nickovic, Willibald Krenn, Gerti Kappel, Schahram Dustdar, and Radu Grosu. CPS/IoT Ecosystem: A Platform for Research and Education. In *Cyber Physical Systems. Model-Based Design (CyPhy WESE 2018)*, pages 206–213, Turin, Italy, 2019. Springer. DOI: 10.1007/978-3-030-23703-5_12.

[84] Anil B. Jambekar and David I. Steinberg. An implicit enumeration algorithm for the all integer programming problem. *Computers and Mathematics with Applications*, 4(1):15–31, 1978. DOI: 10.1016/0898-1221(78)90004-4.

[85] Paul Johannesson and Erik Perjons. *A Method Framework for Design Science Research*, pages 75–89. Springer, 2021. DOI: 10.1007/978-3-319-10632-8_4.

[86] Vasileios Karagiannis and Stefan Schulte. Comparison of Alternative Architectures in Fog Computing. In *4th IEEE International Conference on Fog and Edge Computing (ICFEC 2020)*, pages 19–29, Melbourne, Australia, 2021. IEEE. DOI: 10.1109/ICFEC50348.2020.00010.

[87] Vasileios Karagiannis, Stefan Schulte, João Leitão, and Nuno Preguiça. Enabling Fog Computing using Self-Organizing Compute Nodes. In *3rd IEEE International Conference on Fog and Edge Computing (ICFEC 2019)*, pages 1–10, Larnaca, Cyprus, 2019. IEEE. DOI: 10.1109/CFEC.2019.8733150.

[88] Mostafa Haghi Kashani, Ahmad Ahmadzadeh, and Ebrahim Mahdipour. Load Balancing Mechanisms in Fog Computing: A Systematic Review, 2020. DOI: 10.48550/arXiv.2011.14706.

[89] Shivanjali Khare and Michael Totaro. Big data in iot. In *10th International Conference on Computing, Communication and Networking Technologies (ICCCNT 2019)*, pages 1–7. IEEE, 2019. DOI: 10.1109/ICCCNT45670.2019.8944495.

[90] Daewoo Kim, Hyojung Lee, Hyungseok Song, Nakjung Choi, and Yung Yi. Economics of fog computing: Interplay among infrastructure and service providers, users, and edge resource owners. *IEEE Transactions on Mobile Computing*, 19(11):2609–2622, 2020. DOI: 10.1109/TMC.2019.2925797.

[91] Georgia Kougka and Anastasios Gounaris. Optimization of Data Flow Execution in a Parallel Environment. *Distributed and Parallel Databases*, 37:385–410, 2019. DOI: 10.1007/s10619-018-7243-3.

[92] Simon Kuenzer, Vlad-Andrei Bădoiu, Hugo Lefeuvre, Sharan Santhanam, Alexander Jung, Gaulthier Gain, Cyril Soldani, Costin Lupu, Ştefan Teodorescu, Costi Răducanu, Cristian Banu, Laurent Mathy, Răzvan Deaconescu, Costin Raiciu, and Felipe Huici. Unikraft: Fast, Specialized Unikernels the Easy Way. In *16th European Conference on Computer Systems (EuroSys 2021)*, page 376–394, Online Event, United Kingdom, 2021. ACM. DOI: 10.1145/3447786.3456248.

[93] Mohit Kumar, Subhash C. Sharma, Anubhav Goel, and Simar Preet Singh. A Comprehensive Survey for Scheduling Techniques in Cloud Computing. *Journal of Network and Computer Applications*, 143(C):1–33, 2019. DOI: 10.1016/j.jnca.2019.06.006.

[94] Philipp Leitner, Waldemar Hummer, Benjamin Satzger, Christian Inzinger, and Schahram Dustdar. Cost-Efficient and Application SLA-Aware Client Side Request Scheduling in an Infrastructure-as-a-Service Cloud. In *5th International Conference on Cloud Computing (CLOUD 2012)*, pages 213–220, Honolulu, HI, USA, 2012. IEEE. DOI: 10.1109/CLOUD.2012.21.

[95] Marco Lombardi, Francesco Pascale, and Domenico Santaniello. Internet of Things: A General Overview between Architectures, Protocols and Applications. *Information*, 12(2), 2021. DOI: 10.3390/info12020087.

[96] Francesc Lordan, Daniele Lezzi, Jorge Ejarque, and Rosa M. Badia. An Architecture for Programming Distributed Applications on Fog to Cloud Systems. In *Parallel Processing Workshops (Euro-Par 2017)*, pages 325–337, Santiago de Compostela, Spain, 2018. Springer. DOI: 10.1007/978-3-319-75178-8_27.

[97] Andriy Luntovskyy and Josef Spillner. *Smart Grid, Internet of Things and Fog Computing*, pages 135–210. Springer, 2017. DOI: 10.1007/978-3-658-14842-3_5.

[98] Alexander Maedche, Shirley Gregor, Stefan Morana, and Jasper Feine. Conceptualization of the Problem Space in Design Science Research. In *Extending the Boundaries of Design Science Theory and Practice (DESRIST 2019)*, pages 18–31, Worcester, MA, USA, 2019. Springer. DOI: 10.1007/978-3-030-19504-5_2.

[99] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Edge Affinity-based Management of Applications in Fog Computing Environments. In *12th IEEE/ACM International Conference Utility and Cloud Computing (UCC 2019)*, pages 61–70, Auckland, New Zealand, 2019. ACM. DOI: 10.1145/3344341.3368795.

[100] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Adaptive Resource Configuration for Cloud Infrastructure Management. *Future Generation Computer Systems*, 29(2):472–487, 2013. DOI: 10.1016/j.future.2012.07.004.

[101] Alberto Medina, Anukool Lakhina, Ibrahim Matta, and John Byers. BRITE: An Approach to Universal Topology Generation. In *9th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2001)*, pages 346–354. IEEE, 2001. DOI: 10.1109/MASCOT.2001.948886.

[102] Muniswami Menaka and K.S. Sendhil Kumar. Workflow scheduling in cloud environment – challenges, tools, limitations and methodologies: A review. *Measurement: Sensors*, 24(100436):1–6, 2022. DOI: 10.1016/j.measen.2022.100436.

[103] Roberto Morabito, Vittorio Cozzolino, Aaron Yi Ding, Nicklas Beijar, and Jorg Ott. Consolidate IoT Edge Computing with Lightweight Virtualization. *IEEE Network*, 32(1):102–111, 2018. DOI: 10.1109/MNET.2018.1700175.

[104] Nour Mostafa, Ismael Al Ridhawi, and Moayad Aloqaily. Fog Resource Selection using Historical Executions. In *3rd IEEE International Conference on Fog and Mobile Edge Computing (FMEC 2018)*, pages 272–276, Barcelona, Spain, 2018. IEEE. DOI: 10.1109/FMEC.2018.8364078.

[105] Amina Mseddi, Wael Jaafar, Halima Elbiaze, and Wessam Ajib. Joint Container Placement and Task Provisioning in Dynamic Fog Computing. *IEEE Internet of Things Journal*, 6(6):10028–10040, 2019. DOI: 10.1109/JIOT.2019.2935056.

[106] Biji Nair and Mary Saira Bhanu Somasundaram. Overload Prediction and Avoidance for Maintaining Optimal Working Condition in a Fog Node. *Computers and Electrical Engineering*, 77(C):147–162, 2019. DOI: 10.1016/j.compeleceng.2019.05.011.

[107] Matteo Nardelli, Valeria Cardellini, Vincenzo Grassi, and Francesco Lo Presti. Efficient Operator Placement for Distributed Data Stream Processing Applications. *IEEE Transactions on Parallel and Distributed Systems*, 30(8):1753–1767, 2019. DOI: 10.1109/TPDS.2019.2896115.

[108] Lina Ni, Jinquan Zhang, Changjun Jiang, Chungang Yan, and Kan Yu. Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets. *IEEE Internet of Things Journal*, 4(5).

[109] Özgür Yeniay. Penalty Function Methods for Constrained Optimization with Genetic Algorithms. *Mathematical and Computational Applications*, 10(1):45–56, 2005. DOI: 10.3390/mca10010045.

[110] Alessandro Vittorio Papadopoulos, Laurens Versluis, André Bauer, Nikolas Herbst, Jóakim von Kistowski, Ahmed Ali-Eldin, Cristina L. Abad, José Nelson Amaral, Petr Tůma, and Alexandru Iosup. Methodological Principles for Reproducible Performance Evaluation in Cloud Computing. *IEEE Transactions on Software Engineering*, 47(8):1528–1543, 2021. DOI: 10.1109/TSE.2019.2927908.

[111] Apostolos Papageorgiou, Bin Cheng, and Ernö Kovacs. Real-Time Data Reduction at the Network Edge of Internet-of-Things Systems. In *11th International Conference on Network and Service Management (CNSM 2015)*, pages 284–291, Barcelona, Spain, 2015. IEEE. DOI: 10.1109/CNSM.2015.7367373.

[112] Thales G. Marques Pedro Roger M. Vasconcelos, Gisele Azevedo A. Freitas. KVM, OpenVZ and Linux Containers: Performance Comparison of Virtualization for Web Conferencing Systems. *International Journal Multimedia and Image Processing*, 6(1/2):27–32, 2016. DOI: 10.20533/ijmip.2042.4647.2016.0039.

[113] Ken Peffers, Tuure Tuunanen, and Börn Niehaves. Design Science Research Genres: Introduction to the Special Issue on Exemplars and Criteria for Applicable Design Science Research. *European Journal of Information Systems*, 27(2):129–139, 2018. DOI: 10.1080/0960085X.2018.1458066.

[114] Pei-Hsuan Tsai and Hua-Jun Hong and An-Chieh Cheng and Cheng-Hsin Hsu. Distributed Analytics in Fog Computing Platforms using Tensorflow and Kubernetes. In *19th Asia-Pacific Network Operations and Management Symposium (APNOMS 2017)*, pages 145–150, Seoul, Korea, 2017. IEEE. DOI: 10.1109/APNOMS.2017.8094194.

[115] Paul Pop, Bahram Zarrin, Mohammadreza Barzegaran, Stefan Schulte, Sasikumar Punnekkat, Jan Ruh, and Wilfried Steiner. The fora fog computing platform for industrial iot. *Information Systems*, 98:1–26, 2021. DOI: 10.1016/j.is.2021.101727.

[116] Carlo Puliafito, Enzo Mingozzi, Francesco Longo, Antonio Puliafito, and Omer Rana. Fog Computing for the Internet of Things: A Survey. *ACM Transactions on Internet Technology*, 19(2):1–41, 2019. DOI: 10.1145/3301443.

[117] Wajid Rafique, Lianyong Qi, Ibrar Yaqoob, Muhammad Imran, Raihan Ur Rasool, and Wanchun Dou. Complementing IoT Services Through Software Defined Networking and Edge Computing: A Comprehensive Survey. *IEEE Communications Surveys and Tutorials*, 22(3):1761–1804, 2020. DOI: 10.1109/COMST.2020.2997475.

[118] Mahfuzur Rahman and Peter Graham. Hybrid Resource Provisioning for Clouds. *Journal of Physics: Conference Series*, 385:1–12, 2012. DOI: 10.1088/1742-6596/385/1/012004.

[119] Neetu Raveendran, Huaqing Zhang, Lingyang Song, Li-Chun Wang, Choong Seon Hong, and Zhu Han. Pricing and resource allocation optimization for iot fog computing and nfv: An epec and matching based perspective. *IEEE Transactions on Mobile Computing*, 21(4):1349–1361, 2022. 10.1109/TMC.2020.3025189.

[120] K. Hemant Kumar Reddy, Ashish Kr. Luhach, Buddhadeb Pradhan, Jatindra Kumar Dash, and Diptendu Sinha Roy. A Genetic Algorithm for Energy Efficient Fog Layer Resource Management in Context-Aware Smart Cities. *Sustainable Cities and Society*, 63(1):1–19, 2020. DOI: 10.1016/j.scs.2020.102428.

[121] Ronita Rezapour, Parvaneh Asghari, Hamid Haj Seyyed Javadi, and Shamsollah Ghanbari. Security in Fog Computing: A Systematic Review on Issues, Challenges and Solutions. *Computer Science Review*, 41:1–21, 2021. DOI: 10.1016/j.cosrev.2021.100421.

[122] Hamish Sadler, Alistair Barros, and Wayne Kelly. Fog and Edge Oriented Embedded Enterprise Systems Patterns: Towards Distributed Enterprise Systems That Run on Edge and Fog Nodes. In *55th Hawaii International Conference on System Sciences (HICSS 2022)*, pages 6541–6550, Hawaii, HI, USA, 2022. Western Periodicals. DOI: 10.24251/HICSS.2022.792.

[123] Farah Ait Salaht, Frederic Desprez, and Adrien Lebre. An Overview of Service Placement Problem in Fog and Edge Computing. *ACM Computing Surveys*, 53(3):1–35, 2020. DOI: 10.1145/3391196.

[124] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Resource Provisioning in Fog Computing: From Theory to Practice. *Sensors*, 19(10):1–25, 2019. DOI: 10.3390/s19102238.

[125] Yuya Sasaki. A survey on iot big data analytic systems: Current and future. *IEEE Internet of Things Journal*, 9(2):1024–1036, 2022. DOI: 10.1109/JIOT.2021.3131724.

[126] Enrique Saurez, Kirak Hong, Dave Lillethun, Umakishore Ramachandran, and Beate Ottenwaelder. Incremental deployment and migration

of geo-distributed situation awareness applications in the fog. In *10th ACM International Conference on Distributed and Event-based Systems (DEBS 2016)*, pages 258–269, Irvine, California, 2016. ACM. DOI: 10.1145/2933267.2933317.

[127] Ali Shakarami, Hamid Shakarami, Mostafa Ghobaei-Arani, Elaheh Nikougoftar, and Mohammad Faraji-Mehmandar. Resource provisioning in Edge/Fog Computing: A Comprehensive and Systematic Review. *Journal of Systems Architecture*, 122(102362):1–23, 2022. DOI: 10.1016/j.sysarc.2021.10236.

[128] Prateek Sharma, Lucas Chaufournier, Prashant Shenoy, and Yong Chiang Tay. Containers and Virtual Machines at Scale: A Comparative Study. In *17th International Middleware Conference (Middleware 2016)*, pages 1–13, Trento, Italy, 2016. ACM. DOI: 10.1145/2988336.2988337.

[129] Jeff Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, and Matt Welsh. Hourglass: An infrastructure for connecting sensor networks and applications. Technical Report TR-21-04, Harvard University, 2004. https://www.bibsonomy.org/bibtex/2752c21f350903c8d68f63268df5e75b1/jpcik.

[130] Sukgpal Singh and Inderveer Chana. QoS-Aware Autonomic Resource Management in Cloud Computing: A Systematic Review. *ACM Computing Surveys*, 48(3):1–46, 2016. DOI: 10.1145/2843889.

[131] Steven S. Skiena. *The Algorithm Design Manual*. Springer, 2008. DOI: 10.1007/978-3-030-54256-6.

[132] Oussama Smimite and Karim Afdel. Containers Placement and Migration on Cloud System. *International Journal of Computer Applications*, 176(35):9–18, 2020. DOI: 10.48550/arXiv.2007.08695.

[133] Josef Spillner, Panagiotis Gkikopoulos, Alina Buzachis, and Massimo Villari. Rule-based resource matchmaking for composite application deployments across iot-fog-cloud continuums. In *13th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2020)*, pages 336–341, Leicester, UK, 2020. IEEE. DOI: 10.1109/UCC48980.2020.00053.

[134] Juraj Stacho. *Introduction to Operations Research. Deterministic Models*. Columbia University, 2021. Online; Accessed: Apr. 2023 `https://pdf4pro.com/view/introduction-to-operations-research-4366b.html`.

[135] Ivan Stojmenovic. Fog Computing: A Cloud to the Ground Support for Smart Things and Machine-to-Machine Networks. In *Australasian Telecommunication Networks and Applications Conference*

(ATNAC 2014), pages 117–122, Melbourne, Australia, 2014. IEEE. DOI: 10.1109/ATNAC.2014.7020884.

[136] Koen Tange, Michele De Donno, Xenofon Fafoutis, and Nicola Dragoni. A Systematic Survey of Industrial Internet of Things Security: Requirements and Fog Computing Opportunities. *IEEE Commununications Surveys and Tutorials*, 22(4):2489–2520, 2020. DOI: 10.1109/COMST.2020.3011208.

[137] John A. Tomlin. Technical Note - An Improved Branch-and-Bound Method for Integer Programming. *Operations Research*, 19(4):1070–1075, 1971. DOI: 10.1287/opre.19.4.1070.

[138] Rahul Urgaonkar, Shiqiang Wang, Ting Hea, Murtaza Zafer, Kevin Chan, and Kin K. Leung. Dynamic Service Migration and Workload Scheduling in Edge-Clouds. *Performance Evaluation*, 91:205–228, 2015.

[139] Robert J. Vanderbei. *Linear Programming. Fundations and Extensions*. Springer, 2008. DOI: 10.1007/978-0-387-74388-2.

[140] Konstantinos Vandikas and Vlasios Tsiatsis. Microservices in IoT Clouds. In *2nd International Conference on Cloudification of the Internet of Things (CIoT 2016)*, pages 1–6, Paris, France, 2016. DOI: 10.1109/CIOT.2016.7872912.

[141] Luis M. Vaquero and Luis Rodero-Merino. Finding your Way in the Fog: Towards a Comprehensive Definition of Fog Computing. *ACM SIGCOMM Computer Communication Review*, 44(5):27–32, 2014. DOI: 10.1145/2677046.2677052.

[142] Blesson Varghese and Rajkumar Buyya. Next Generation Cloud Computing: New Trends and Research Directions. *Future Generation Computer Systems*, 79:849–861, 2018. DOI: 10.1016/j.future.2017.09.020.

[143] Prateeksha Varshney and Yogesh Simmhan. Demystifying Fog Computing: Characterizing Architectures, Applications and Abstractions. In *IEEE 1st International Conference on Fog and Edge Computing (ICFEC 2017)*, pages 115–124, Madrid, Spain, 2017. IEEE. DOI: 10.1109/ICFEC.2017.20.

[144] Stephen J. Vaughan-Nichols. New Approach to Virtualization Is a Lightweight. *Computer*, 39(11):12–14, 2006. DOI: 10.1109/MC.2006.393.

[145] Michael Vögler, Johannes Michael Schleicher, Christian Inzinger, and Schahram Dustdar. Optimizing Elastic IoT Application Deployments. *IEEE Transactions on Services Computing*, 11(5):1–14, 2016. DOI: 10.1109/TSC.2016.2617327.

[146] Jianxin Wang, Ming K. Lim, Chao Wang, and Ming-Lang Tseng. The Evolution of the Internet of Things (IoT) over the Past 20 Years. *Computers and Industrial Engineering*, 155:1–14, 2021. DOI: 10.1016/j.cie.2021.107174.

[147] Zhiyu Wang, Mohammad Goudarzi, Jagannath Aryal, and Rajkumar Buyya. Container Orchestration in Edge and Fog Computing Environments for Real-Time IoT Applications. In *Computational Intelligence and Data Analytics*, volume 142, pages 1–21. Springer, 2023. DOI: 10.1007/978-981-19-3391-2_1.

[148] Charlotte Werndl. On Choosing between Deterministic and Indeterministic Models: Underdetermination and Indirect Evidence. *Synthese*, 190(12):2243–2265, 2013. DOI: 10.1007/s11229-011-9966-9.

[149] Darrell Whitley. A Genetic Algorithm Tutorial. *Statistics and Computing*, 4:65–85, 1994. DOI: 10.1007/BF00175354.

[150] Roel Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014. DOI: 10.1007/978-3-662-43839-8.

[151] David P. Williamson and David B. Shmoys. *The Design of Approximation Algorithms*. Cambridge University Press, 2010. DOI: 10.1017/CBO9780511921735.

[152] Cecil Wöbker, Andreas Seitz, Harald Mueller, and Berndt Bruegge. Fogernetes: Deployment and Management of Fog Computing Applications. In *IEEE/IFIP Network Operations and Management Symposium (NOMS 2018)*, pages 1–7, Taipei, Taiwan, 2018. IEEE. DOI: 10.1109/NOMS.2018.8406321.

[153] Yong Xiao and Marwan Krunz. QoE and Power Efficiency Tradeoff for Fog Computing Networks with Fog Node Cooperation. In *International Conference on Computer Communications (INFOCOM 2017)*, pages 1–9, Atlanta, GA, USA, 2017. IEEE. DOI: 10.1109/INFOCOM.2017.8057196.

[154] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014. DOI: 10.1109/TII.2014.2300753.

[155] Minxian Xu, Wenhong Tian, and Rajkumar Buyya. A Survey on Load Balancing Algorithms for Virtual Machines Placement in Cloud Computing. *Concurrency and Computation Practice and Experience*, e4123:1–16, 2017. DOI: 10.1002/cpe.4123.

[156] Qiaozhi Xu and Junxing Zhang. piFogBed: A Fog Computing Testbed Based on Raspberry Pi. In *38th IEEE International Performance Computing and Communications Conference (IPCCC 2019)*, pages 1–8, London, England, 2019. IEEE. DOI: 10.1109/IPCCC47392.2019.8958741.

[157] Mahendra Pratap Yadav, Nisha Pal, and Dharmendra Kumar Yadav. Resource provisioning for containerized applications. *Cluster Computing*, 2021(24):2819–2840, 2021. DOI: 10.1007/s10586-021-03293-5.

[158] Marcello Yannuzzi, Rodolfo Milito, Rene Serral-Gracià, Diego Montero, and Mrio Nemirovsky. Key ingredients in an iot recipe: Fog computing, cloud computing, and more fog computing. In *19th IEEE International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD 2019)*, pages 325–329. IEEE, 2014. DOI: 10.1109/CAMAD.2014.7033259.

[159] Zhen Ye, Xiaofang Zhou, and Athman Bouguettaya. Genetic Algorithm Based QoS-Aware Service Compositions in Cloud Computing. In *16th International Conference on Database Systems for Advanced Applications (DASFAA 2016)*, pages 321–334, Hong Kong, China, 2011. Springer. DOI: 10.1007/978-3-642-20152-3_24.

[160] Qing-Ming Yi, Min Shi, Ming-Min Chen, and Geng Wang. Research and Design of Embedded Microprocessor based on ARM Architecture. In *13th International Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP 2016)*, pages 463–467, Chengdu, China, 2016. IEEE. DOI: 10.1109/ICCWAMTIP.2016.8079895.

[161] Shanhe Yi, Cheng Li, and Qun Li. A Survey of Fog Computing: Concepts, Applications and Issues. In *16th ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2015)*, pages 37–42, Hangzhou, China, 2015. ACM. DOI: 10.1145/2757384.2757397.

[162] Emre Yigitoglu, Mohamed Mohamed, Ling Liu, and Heiko Ludwig. Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing. In *IEEE International Conference on AI Mobile Services (AIMS 2017)*, pages 38–45, Honolulu, HI, USA, 2017. IEEE. DOI: 10.1109/AIMS.2017.14.

[163] Myungryun Yoo. Real-time Task Scheduling by Multiobjective Genetic Algorithm. *Journal of Systems and Software*, 82(4):619–628, 2009. DOI: 10.1016/j.jss.2008.08.039.

[164] Jia Yu, Rajkumar Buyya, and Kotagiri Ramamohanarao. Workflow Scheduling Algorithms for Grid Computing. *Studies in Computational Intelligence*, 146:173–214, 2008. DOI: 10.1007/978-3-540-69277-5_7.

[165] Uwe Zdun, Rafael Capilla, Huy Tran, and Olaf Zimmermann. Sustainable architectural design decisions. *IEEE Software*, 30(6):46–53, 2013. DOI: 10.1109/MS.2013.97.

[166] Zhi-Hui Zhan, Xiao-Fang Liu, Yue-Jiao Gong, Jun Zhang, Henry Shu-Hung Chung, and Yun Li. Cloud Computing Resource Scheduling and a Survey of Its Evolutionary Approaches. *ACM Computing Surveys*, 47(4):1–33, 2015. DOI: 10.1145/2788397.

[167] Changhao Zhang. Design and application of fog computing and internet of things service platform for smart city. *Future Generation Computer Systems*, 112:630–640, 2020. DOI: 10.1016/j.future.2020.06.016.

[168] Decheng Zhang, Faisal Haider, Marc St-Hilaire, and Christian Makaya. Model and Algorithms for the Planning of Fog Computing Networks. *IEEE Internet Things Journal*, 6(2):3873–3884, 2019. DOI: 10.1109/JIOT.2019.2892940.

[169] Wenyu Zhang, Zhenjiang Zhang, and Han-Chieh Chao. Cooperative Fog Computing for Dealing with Big Data in the Internet of Vehicles: Architecture and Hierarchical Resource Management. *IEEE Commununications Magazine*, 55(12):60–67, 2017. DOI: 10.1109/MCOM.2017.1700208.

[170] Lin Zhu, Junjiang Li, Zijie Liu, and Dengyin Zhang. A multi-resource scheduling scheme of kubernetes for iiot. *Journal of Systems Engineering and Electronics*, 33(3):683–692, 2022. DOI: 10.23919/JSEE.2022.000063.

Appendix A. Curriculum Vitae

e-mail:	o.skarlat@infosys.tuwien.ac.at
www:	https://dsg.tuwien.ac.at/team/oskarlat/
Google Scholar:	https://scholar.google.com/citations?user=SVJf2XkAAAAJ&hl=en

Education

2015-01 – Ongoing	**PhD Studies** **Resource Provisioning in Fog Computing** *Distributed Systems Group, TU Wien – Vienna, Austria*
2008-09 – 2010-07	**Master of Science with honors** **Information Control Systems and Technologies** *National University of "Kyiv Mohyla Academy" – Kyiv, Ukraine*
2004-09 – 2008-07	**Bachelor of Science with honors** **Computer Science** *National University "Kyiv Polytechnic Institute" – Kyiv, Ukraine*
2008-09 – 2010-07	**Bachelor of Science** **Finance And Entrepreneurship** *International University of Finance – Kyiv, Ukraine*

Job History

2021-09 – Ongoing	**Senior Data Scientist** *Business Transformation & Execution* *A1 Austria Telekom Group – Vienna, Austria* Driving ML projects, formulating business goals Data specification and data engineering for projects Development of ML models and MLOps

Development of ML applications, deployment, maintenance of applications
Work with Azure Machine Learning Service and other Azure components
Presentations, dissemination, and deliverables
Supervision of junior data scientists

2018-12 – 2021-08 **Machine Learning Engineer**
Solution Architecture
A1 Digital International GmbH – Vienna, Austria
Data transformation and analysis
Development of ML models and MLOps
Development of ML applications, deployment, and maintenance of applications
Work with the BigML platform, deployment and maintenance of ML applications
Automated UI/API/MQTT testing for web services
CI/CD for ML projects, scalable Kubernetes services for ML
Workshops for customers, presentations, deliverables, blog posts

2018-07 – 2018-11 **Data Science Intern**
Solution Architecture
A1 Digital International GmbH – Vienna, Austria
Implementation of ML projects and PoCs
Descriptive statistics and data transformation
Development of ML applications, deployment, and maintenance of applications

2015-01 – 2018-06 **Research Assistant**
Distributed Systems Group, TU Wien – Vienna, Austria
Research in the areas of cloud, fog and edge computing, the Internet of Things
Published multiple scientific articles on fog computing and cloud manufacturing
Development of Spring services for the Horizon 2020 project Cloud-based Rapid Elastic Manufacturing (CREMA), management of a component "Service virtualization and abstraction"

2010-11 – 2014-09 **Research Assistant**
Glushkov Institute Of Cybernetics Of NAS Ukraine – Kyiv, Ukraine
Research in the areas of grid and cloud computing
E-signature and e-services

	Technical Committee for Standardization TC-20 “Information Technologies”, adaptation and translation of international standards Deliverables and technical reports
2013-09 –2014-06	**Business Analyst** *AT Consulting – Kyiv, Ukraine* Project requirements specification Technical specifications and project documentation Pre-sales presentations and writing sales proposals
2010-01 - 2010-09	**Assistant Project Manager** *Newtonideas Inc. – Kyiv, Ukraine* Project management activities Automated testing

Awards

- President’s Scholarship for Young Scientists. Kyiv, Ukraine, 2012.

Publications

Journals

- Olena Skarlat and Stefan Schulte. FogFrame: A Framework for IoT Application Execution in the Fog. *PeerJ Computer Science*, 7:e588, 1-45, 2021. eCollection. DOI: 10.7717/peerj-cs.588.
- Olena Skarlat, Matteo Nardelli, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Optimized IoT Service Placement in the Fog. *Service-Oriented Computing and Applications*, 11(4), 1–17, 2017. Springer. DOI: 10.1007/s11761-017-0219-8
- Philipp Waibel, Johannes Matt, Christoph Hochreiner, Olena Skarlat, Ronny Hans, and Stefan Schulte. Cost-Optimized Redundant Data Storage in the Cloud. *Service-Oriented Computing and Applications*, 11(4), 411–426. 2017. DOI: 10.1007/s11761-017-0218-9
- Vladislav Falfushinsky, Olena Skarlat, and Vadim Tulchinsky. Integration of Cloud Computing Platform to Grid Infrastructure. *International Journal of Computing*, 12(4), 333–339, 2014. CiCj. DOI: 10.47839/ijc.12.4.613

Conference and Workshop Proceedings

- Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. A Framework for Optimization, Service Placement, and Runtime Operation in the Fog. In *11th International Conference on Utility*

and Cloud Computing (UCC 2018), pages 164–173, Zurich, Switzerland, 2018. IEEE. DOI: 10.1109/UCC.2018.00025

- Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards QoS-aware Fog Service Placement. In *1st International Conference on Fog and Edge Computing (ICFEC 2017)*, pages 89–96, Madrid, Spain, 2017. IEEE. DOI: 10.1109/ICFEC.2017.12
- Olena Skarlat, Stefan Schulte, Michael Borkowski, and Philipp Leitner. Resource Provisioning for IoT Services in the Fog. In *9th International Conference on Service Oriented Computing and Applications (SOCA 2016)*, pages 32–39, Hong Kong, China, 2016. IEEE. DOI: 10.1109/SOCA.2016.10
- Olena Skarlat, Michael Borkowski, and Stefan Schulte. Towards a Methodology and Instrumentation Toolset for Cloud Manufacturing. In *1st International Workshop on Cyber-Physical Production Systems (CPPS 2016)*, pages 1–4, Vienna, Austria, 2016. IEEE. DOI: 10.1109/CPPS.2016.7483920
- Michael Borkowski, Olena Skarlat, Stefan Schulte, and Schahram Dustdar. Prediction-Based Prefetch Scheduling in Mobile Service Applications. In *5th International Conference on Mobile Services (MS 2016)*, pages 41–48, San Francisco, CA, 2016. IEEE. DOI: 10.1109/MobServ.2016.17
- Olena Skarlat. Elastic Manufacturing Process Landscapes. In *8th ZEUS Workshop (ZEUS 2016)*, pages 45–48, Vienna, Austria, 2016. CEUR Workshop Proceedings. Online. Accessed: Apr. 2023. http://ceur-ws.org/Vol-1562/paper6.pdf
- Olena Skarlat, Philipp Hoenisch, and Schahram Dustdar. On Energy Efficiency of BPM Enactment in the Cloud. In *Workshop on Process Engineering (IWPE) at the 13th International Conference on Business Process Management (BPM 2016)*, pages 489–500, vol. 256. 2016. LNBIP. DOI: 10.1007/978-3-319-42887-1_39
- Stefan Schulte, Michael Borkowski, Christoph Hochreiner, Matthias Klusch, Aitor Murguzur, Olena Skarlat, and Philipp Waibel. Bringing Cloud-based Rapid Elastic Manufacturing to Reality with CREMA. In *Workshop on Intelligent Systems Configuration Services for Flexible Dynamic Global Production Networks (FLEXINET) at the 8th International Conference on Interoperability for Enterprise Systems and Applications (I-ESA 2016)*, pages 407–413, Guimaraes, Portugal, 2016. ISBN: 978-1-84704-044-2.
- Vladislav Falfushinsky, Olena Skarlat, and Vadim Tulchinsky. On Power Efficiency of Distributed Computing. In *1st International Black Sea Conference on Communications and Networking (BlackSeaCom 2013)*, poster, Batumi, Georgia, 2013.
- Vladislav Falfushinsky, Olena Skarlat, and Vadim Tulchinsky. Cloud Computing Platform within Grid Infrastructure. In *7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS 2013)*, pages 717-720 Berlin, Germany. IEEE. DOI: 10.1109/IDAACS.2013.6663018