TU WIEN Informatics

Resource Management and Elasticity Control in Edge Networks

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Ilir Murturi, MSc
Matrikelnummer 11742795

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Univ.Prof. Dr. Schahram Dustdar

Diese Dissertation haben begutachtet:

Prof. Dr. Harald Gall	Assoc.Prof. Dr. George Pallis

Wien, 20. Jänner 2022

Ilir Murturi, MSc

TU WIEN Informatics

Resource Management and Elasticity Control in Edge Networks

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Ilir Murturi, MSc
Registration Number 11742795

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dr. Schahram Dustdar

The dissertation has been reviewed by:

Prof. Dr. Harald Gall	Assoc.Prof. Dr. George Pallis

Vienna, 20th January, 2022

Ilir Murturi, MSc

Erklärung zur Verfassung der Arbeit

Ilir Murturi, 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, 20. Jänner 2022

Ilir Murturi, MSc

TO MY PARENTS

for their endless love, support, and encouragement.

Acknowledgements

I owe thanks to numerous people that helped me throughout my Ph.D. Looking back, I realize how lucky I have been to join Distributed Systems Group (DSG) and have had the chance to meet remarkable people who have helped me to develop professionally. I am grateful to the DSG colleagues for making every day a pleasant working environment. The work I have done would never be possible without their friendly support.

I would like to express my sincere and most profound gratitude to my supervisor Univ.Prof. Dr. Schahram Dustdar, who gave me the opportunity to pursue my Ph.D. in his group and then patiently guided me throughout the entire thesis. I am thankful for every moment and the numerous pieces of advice he gave me during my Ph.D. His friendly tips over the years did not just help me improve professionally but also helped me grow personally. Furthermore, I would like to thank Assoc.Prof. Dr. George Pallis, and Prof. Dr. Harald Gall for reviewing and helping improve this thesis. Many thanks to Univ.Prof. Dr. Uwe Zdun and Univ.Prof. Dr. Wolfgang Kastner for agreeing to be in my proficiency evaluation committee, giving me very valuable feedback in the earlier stages of my research. Additionally, I would like to thank my collaborators for the knowledge exchange and contributions.

Finally, my deepest and sincerest thanks are dedicated to my family for their unconditional love and endless support. I am eternally grateful for everything they have done for me to accomplish my life goals.

The research presented in this thesis was funded partially by the "Smart Communities and Technologies (Smart CT)" and it has received funding from the EU's Horizon 2020 Research and Innovation Programme under grant agreement No. 871525. EU web site for Fog Protect: https://fogprotect.eu/

Kurzfassung

Edge Computing wurde vor kurzem als zwischengeschaltete Computing-Einheit zwischen Internet of Things (IoT)-Implementierungen und der Cloud eingeführt und stellt den teilnehmenden IoT-Geräten Daten oder Kontrollmöglichkeiten bereit. An der Schnittstelle dieser Domänen sind neue moderne Anwendungen entstanden, die einen Echtzeitzugriff auf Sensordaten aus der Umgebung mit geringer Latenz erfordern. Traditionell erfolgt die Ausführung solcher Anwendungen in ressourcenreichen Umgebungen wie der Cloud. Die enorme Menge an Datenübertragungen, heterogenen Geräten und Netzwerken wirken sich jedoch auf die Latenz aus, was wiederum zu einer hohen Latenz in IoT-Systemen führt. Um diese Engpässe der aktuellen Cloud-zentrierten Systeme zu überwinden, haben Forscher aus Wissenschaft und Industrie vorgeschlagen, IoT-Anwendungen mit strengen Anforderungen näher am Rand des Netzwerks bereitzustellen, um so ihre unterschiedlichen Anforderungen wie Hochverfügbarkeit, Leistung oder Datenschutz besser zu erfüllen. Dennoch ist ein zentralisiertes Ressourcenmanagement – typischerweise in der Cloud und in den heutigen IoT-Cloud-Architekturen offensichtlich – eine Lösung, erfordert jedoch, dass Cloud-Kontrollstrukturen immer verfügbar sind und eine geringe Latenz aufweisen. Dies erfordert daher neuartige Ressourcenverwaltungstechniken, die in Edge-Netzwerken bereitgestellt und ausgeführt werden und verschiedenen Endbenutzern helfen, eine Anwendung im Ziel-Edge-Netzwerk bereitzustellen und zu verwalten. Dennoch wurden Edge-Umgebungen als dynamisch, ressourcenbeschränkt, gekennzeichnet durch Unsicherheit und heterogene Computergeräte identifiziert. Insbesondere diese Eigenschaften von Edge-Netzwerken haben tiefgreifende Auswirkungen auf die Ausführung und Verwaltung von Anwendungsfunktionen zur Laufzeit.

Diese Dissertation bietet neuartige Methoden und Edge-basierte Ressourcenmanagement-Frameworks zur Unterstützung und Verwaltung von Laufzeitaspekten von Anwendungen (d. h. Edge-Anwendungen), die auf ressourcenbeschränkten Edge-Netzwerken ausgeführt werden. Genauer gesagt entwickeln wir ein neuartiges Edge-basiertes System, das Ressourcenmanagementfunktionen basierend auf drei Perspektiven bereitstellt: (1) Ressourcenerkennung auf dezentrale Weise, (2) Kontrolle der Elastizität von Edge-Anwendungen am Edge und (3) Ressourcenkoordination bei die Kante. Unser vorgeschlagenes Edge-basiertes System ermöglicht und unterstützt die Ausführung von Edge-Anwendungen in verschiedenen Edge-Netzwerken und behält ihre korrekte Funktionalität während der gesamten Ausführungszeit bei. Das vorgeschlagene Edge-basierte System ist hinsichtlich der Funktionalitäten skalierbar und erweiterbar und entworfen, um unter verschiedenen

Umständen zu arbeiten, die in Edge-Netzwerken auftreten können (z. B. Dynamik). Wir untersuchen und analysieren zunächst Edge-Szenarien aus mehreren Domänen, ermitteln Anforderungen und argumentieren dann für die Notwendigkeit, Edge-basierte Systeme zu dezentralisieren. Aufgrund der breiten Palette von Edge-Anwendungen und deren Ressourcenanforderungen argumentieren wir, dass Edge-Geräte miteinander interagieren und Ressourcen in einem Edge-Netzwerk automatisch erkennen müssen. Durch Experimente an dem von uns gebauten Prototyp zeigen wir zunächst die Machbarkeit der Bildung eines Edge-Netzwerks mit leistungsarmen Edge-Geräten. Anschließend demonstrieren wir die Möglichkeit, Ressourcen zu entdecken, indem wir Ressourcenmetadaten zwischen Edge-Geräten auf dezentrale Weise replizieren. Anschließend erweitern wir unsere vorgeschlagene Lösung um einen leichtgewichtigen Mechanismus, der es ermöglicht, Edge-Anwendungen in einem Edge-Netzwerk bereitzustellen und deren Skalierbarkeit/-Elastizität zur Laufzeit zu kontrollieren. Genauer gesagt findet die Planer-Komponente geeignete Pläne zur Bereitstellung von Edge-Anwendungen, während dezentrale Elastizitätskomponenten automatisch die Funktionalität der Anwendungen aufrechterhalten, indem sie die zur Entwurfszeit festgelegten elastischen Anforderungen kontinuierlich überwachen. Unser vorgeschlagener Elastizitätskontrollmechanismus berücksichtigt mehrere elastische Perspektiven (d. h. Qualität, Kosten und Ressourcen), um die korrekte Funktionalität von Edge-Anwendungen zu gewährleisten. Durch Experimente an dem von uns gebauten Prototyp zeigen wir die Machbarkeit der Ausführung von Elastizitätsmerkmalen in einem ressourcenbeschränkten Kantennetzwerk. Darüber hinaus zeigen wir die Möglichkeit, die korrekte Funktionalität von Edge-Anwendungen angesichts dynamischer Belastungen zur Laufzeit zu kontrollieren und aufrechtzuerhalten. Schließlich unterstützen wir Edge-Anwendungen oder Endgeräte mit einem Mechanismus, der ihnen hilft, nicht triviale Ressourcen zu erhalten. Wir präsentieren ein neuartiges technisches Framework für die Koordination von Engineering-Ressourcen für Edge-fähiges IoT. Unser vorgeschlagener Ansatz garantiert die Optimalität und Korrektheit der generierten Pläne, um nicht-triviale Ressourcen zu erhalten. Durch Experimente zeigen wir die Leistungsfähigkeit und Realisierbarkeit unseres Ansatzes auf ARM-basierten Edge-Geräten mit geringem Stromverbrauch.

Abstract

Edge computing has been recently introduced as an intermediary computing entity between Internet of Things (IoT) deployments and the cloud, providing data or control facilities to participating IoT devices. New modern applications have emerged at the intersection of these domains that require real-time access to sensor data from the environment with low latency. Traditionally, executing such applications happens in resource-rich environments such as the cloud. However, the massive amount of data transfer, heterogeneous devices, and networks involved affect latency, which in turn causes high latency in IoT systems. To overcome these bottlenecks of the current cloud-centric systems, researchers from academia and industry have suggested deploying IoT applications with stringent requirements closer to the edge of the network - thus, better satisfying their various demands such as high availability, performance, or privacy. Nevertheless, centralized resource management — typically in the cloud and evident in today's IoT-cloud architectures is one solution but requires cloud control structures to be always available and within low latency. Therefore, this calls for novel resource management techniques deployed and executed on edge networks and aids various end-users in deploying and managing an application in the target edge network. Nonetheless, edge environments have been identified as dynamic, resource-constrained, characterized by uncertainty, and heterogeneous computing devices. Specifically, these characteristics of edge networks have profound implications for execution and managing application functionality at runtime.

This thesis provides novel methodologies and edge-based resource management frameworks to assist and manage runtime aspects of IoT applications (i.e., edge applications) executed on resource-constrained edge networks. More precisely, we develop a novel edge-based system that provides resource management features based on three perspectives: (1) resource discovery in a decentralized manner, (2) controlling elasticity of edge applications at the edge, and (3) resource coordination at the edge. Our proposed edge-based system enables and supports the execution of edge applications on various edge networks and maintains their correct functionality throughout the execution time. The proposed edge-based system is scalable and expandable in terms of functionalities and designed to work under various circumstances that can appear in edge networks (e.g., dynamicity). We first examine and extensively analyze edge scenarios from multiple domains, elicit requirements, and then argue the necessity to decentralize edge-based systems. Because of the broad range of edge applications and their resource demands, we argue that

edge devices need to interact with each other and automatically discover resources in an edge network. Through experiments on the prototype we have built, we first show the feasibility of forming an edge network with low-powered edge devices. We then demonstrate the feasibility of discovering resources by replicating resource metadata among edge devices in a decentralized manner. Afterward, we extend our proposed solution with a lightweight mechanism that allows deploying edge applications on an edge network and controlling their scalability/elasticity at runtime. More precisely, the planner component finds eligible plans to deploy edge applications while decentralized elasticity components automatically maintain applications' functionality by continuously monitoring elastic requirements specified at design time. Our proposed elasticity control mechanism considers multiple elastic perspectives (i.e., quality, cost, and resources) to guarantee the correct functionality of edge applications. Through experiments on the prototype we have built, we show the feasibility of executing elasticity features on a resource-constrained edge network. Furthermore, we show the feasibility of controlling and maintaining the correct functionality of edge applications in the face of dynamic loads at runtime. Lastly, we support edge applications or end devices with a mechanism to help them obtain non-trivial resources. We present a novel technical framework for engineering resource coordination for edge-enabled IoT. Our proposed approach guarantees regarding optimality and correctness of generated plans to obtain non-trivial resources. Through experiments, we show the performance and realizability of our approach on low-powered ARM-based edge devices.

Contents

Publications

The research presented in this thesis is partly based on the following peer-reviewed publications (i.e., journals, conferences, and book chapters). A full list of publications can be found in Google Scholar Profile[1]

- Ilir, Murturi, and Schahram Dustdar. "A Decentralized Approach for Resource Discovery using Metadata Replication in Edge Networks." *IEEE Transactions on Services Computing*, pages 1-12, 2021.
- Ilir, Murturi, and Schahram Dustdar. "DECENT: A Decentralized Configurator for Controlling Elasticity in Dynamic Edge Networks." *ACM Transactions on Internet Technology (TOIT)*, pages 1-21, 2022.
- Ilir, Murturi, Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. "Edge-to-edge resource discovery using metadata replication." In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pp. 1-6. IEEE, 2019.
- Ilir, Murturi, Mohammadreza Barzegaran, and Schahram Dustdar. "A decentralized approach for determining configurator placement in dynamic edge networks." In *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, pp. 147-156. IEEE, 2020.
- Tsigkanos, Christos, Ilir Murturi, and Schahram Dustdar. "Dependable resource coordination on the edge at runtime." *Proceedings of the IEEE* 107, no. 8 (2019): 1520-1536.
- Dustdar, Schahram, and Ilir Murturi. "Towards Distributed Edge-based Systems." In *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, pp. 1-9. IEEE, 2020.
- Dustdar, Schahram, and Ilir Murturi. "Towards IoT Processes on the Edge."*In Next-Gen Digital Services. A Retrospective and Roadmap for Service Computing of the Future*, pp. 167-178. Springer, Cham, 2021.

[1] `https://scholar.google.at/citations?user=OL_Y_mgAAAAJ`

- Ilir, Murturi, Egyed, Adam, and Schahram Dustdar. "Utilizing AI Planning on the Edge" in *IEEE Internet Computing*, vol. 26, no. 2, pp. 28-35, 1 March-April 2022, doi: 10.1109/MIC.2021.3073434. (2022).
- Ilir, Murturi, Jia, Chao, Kerbl, Bernhard, Wimmer, Michael, Dustdar, Schahram, and Tsigkanos, Christos. "On Provisioning Procedural Geometry Workloads on Edge Architectures." *In Proceedings of the 17th International Conference on Web Information Systems and Technologies*, ISBN 978-989-758-536-4, ISSN 2184-3252, pages 354-359., 2021
- Alkhabbas, Fahed, Ilir Murturi, Romina Spalazzese, Paul Davidsson, and Schahram Dustdar. "A goal-driven approach for deploying self-adaptive iot systems." In *2020 IEEE International Conference on Software Architecture (ICSA)*, pp. 146-156. IEEE, 2020.

Furthermore, several topics were investigated together with students through their Master's and Bachelor's thesis under my supervision. A full list of topics investigated with students can be found in Distributed Systems Group website[2]

- Boris Sedlak, "Specification and Operation of Privacy Models for Data Streams on the Edge", Master Thesis (2022)
- Cem Bicer, "Design and Implementation of a Blockchain-based Zero Trust Architecture on the Edge", Master Thesis (2022 - in progress)

[2] `https://dsg.tuwien.ac.at/team/imurturi/`

CHAPTER 1

Introduction

In recent years, the Internet of Things (IoT) has been diffused into society, and many applications are constructed on top of IoT technologies in various industries such as Industrial Manufacturing, Healthcare, Lifestyle, Automotive, and Smart Building, just to name a few. Satisfying stringent requirements of IoT applications (i.e., low-latency and high-availability) has become challenging for cloud-centric architectures, resulting in latencies higher than the expected response for IoT applications. For instance, an IoT application with stringent requirements running on the cloud may face high latency since a massive amount of sensory data must be transferred to the cloud. In addition, centralizing resource management — typically in the cloud and evident in today's IoT-cloud architectures is one solution, but requires cloud control structures to be always available and within low latency. At the same time, it is well accepted that a centralized architecture does not scale well regarding the enormous number of devices even though the system infrastructure of central computers processing those data have improved by cloud computing technologies. Furthermore, cloud environments are characterized by the uncertainty of availability all the time.

To address these shortcomings of cloud-centric solutions, recent advancements within distributed systems have resulted in the architectural placement of a computing entity closer to the network edge. Available computing devices in proximity satisfy applications' stringent requirements, such as high availability, low latency, performance, or privacy [SCZ+16, SD16]. Such computation entities perceived as edge devices are used to offer computation resources to local devices [RGXZ17] and to bridge the gap between the cloud and IoT domain in order to facilitate low-latency and highly resilient applications. Within a neighborhood, for example, IoT devices may utilize resources of a local edge device, benefiting from high connectivity to it as well as its awareness of other IoT devices within its scope. This allows edge devices to act as mediators among other IoT devices, locally process incoming data and satisfy IoT device needs. Consequently, these devices

reduce reliance on cloud-based environments while aiming to reduce network congestion, high-latency, and enforce privacy by processing the data near end-users.

Research efforts associated with edge computing are still at a relatively early stage of development. It is worth noting that researchers both from academia and industry proposed two new paradigms, called fog computing [BMZA12] and edge computing [SCZ+16, SD16], which bring the computational resources (i.e., storage, networking, and processing) closer to the edge of the network. In general, the vision of the two paradigms overlaps to make more computation resources available at the edge of the network. Both paradigms have been proven to provide significant advantages for many various use cases such as video stream analytics [YHZ+17], storage [LDMB17], deep learning for IoT [LOD18], IoT application deployment [AD19], among others. Hence, a lot of studies have been performed, leading to various computing platforms that strive to leverage the edge of the network to meet application demands (e.g., latency and bandwidth) and enhance user experience [DMB18].

We recognize that edge computing means different things to different people; we vision edge computing as a bridge between IoT things and the nearest edge device to the user. Moreover, we identify an edge device as a low-powered ARM-based computer with a limited software stack and attached IoT resources. Nevertheless, heterogeneous edge devices are increasingly added at the edge of a network to acquire faster response and increased privacy [TNL+19]. Such edge environments bring seamless opportunities for deploying edge applications[1] (i.e., IoT applications) in proximity to IoT domains and end-users. However, edge networks' dynamicity and uncertainty bring new challenges where resource management techniques and elasticity control mechanisms are needed to fully utilize available resources and maintain edge applications' functionality and desired service quality at runtime. Thus, using various computational resources established by multiple heterogeneous devices in edge networks requires novel resource management techniques. In addition, the broad range of edge applications and their resource demands requires introducing flexible and modular edge-based frameworks that allow adding various functionalities at the edge.

1.1 Problem Statement

The vision of edge computing and IoT is a composite system where connected edge devices provide and manage a massive number of resources in proximity to end-users. Accordingly, the convergence between two concepts changes the way we conceptualize IoT as a large number of things connected to the Internet, which are often distributed [TNL+19]. We advocate that IoT needs to be thought of as an ecosystem of discrete computing resources, where generated sensory data is processed in the nearest available device in the computing infrastructure (i.e., the Edge-Cloud continuum). More precisely, IoT resources must benefit from available computational resources to process sensory data streams with low latency at the edge of the network [SD16, BM18]. This has opened up the development

[1] In this thesis, we use the terms edge applications and IoT applications interchangeably.

of novel edge applications in different fields, such as building automation, transportation, logistics, and healthcare [AIM10, GIMA10]. Thus, this led to a plethora of application-specific solutions developed over the past few years. As a result, this intuition led us to envision a world where general-purpose devices are sold to users and enable them to build their edge networks (i.e., homes, factories, hospitals, etc.). In this context, edge networks would allow users to customize their environment by deploying various edge applications in their edge network. In addition, edge applications and their functionality dependencies can be downloaded from a centralized IoT cloud platform [DAM19, MATD19].

This thesis aims to provide various resource management techniques executed on resource-constrained edge networks. More precisely, we provide several resource management functionalities to enable and support the execution of edge applications from different domains on resource-constrained edge networks and maintain correct functionality throughout execution time. We envision an ecosystem comprising a cloud-based IoT platform and an edge-based framework to achieve such objectives. The former is an environment where application developers or domain experts develop resource management techniques and edge applications. The cloud-based IoT platform essentially provides features for developers to specify high-level application requirements for their edge applications. The latter is an edge-based system[2] that provides mechanisms and methodologies to form an edge network, enable efficient resource discovery, efficiently utilize available resources, and maintain applications functionality throughout their execution time. Edge applications are executed in a runtime that considers the heterogeneity of edge resources giving applications seamless access to the targeting edge network. Based on the runtime reports (i.e., infrastructure and application metrics), the proposed edge-based system re-configures edge applications as necessary to meet their specified application requirements. Nevertheless, research literature shows that edge solutions operate in isolation, are mainly hard-coded, inflexible, and limited for future changes. However, we require a flexible edge-based system that adds novel functionalities at runtime to support executing a wide range of edge applications on an edge network.

In our envisioned ecosystem, edge computing is positioned as one critical architectural layer/stack in addition to fog and cloud. Unlike in cloud computing, there is no clear definition in research literature about what an edge infrastructure is precisely. The research literature shows that the concrete infrastructure and network topology entirely depend on the targeted use case and domain. Furthermore, edge devices within edge infrastructures are more diverse and resource-constrained with limited resources, referring to their different computational capabilities, including storage or processing facilities. Accordingly, executing an edge application (e.g., image processing) on a single edge device poses many limitations and a set of challenges in terms of processing capabilities, storage, and communication bandwidth. As a result, the resource-constrained nature of edge networks has led to the development of modern edge applications from a monolithic architecture to edge applications comprised of a set of independently deployable

[2]In this thesis, we refer to this new system as an edge-based system since it aims to place resource management features on an edge network.

software components (i.e., microservices). Furthermore, research literature shows several recommendations such that edge devices must collaborate with other edge devices in proximity to fulfill resource demands for each edge application component. Currently, we lack several proper mechanisms that will enable forming an edge network and discovering available resources, maintaining the desired functionality of edge applications at runtime while considering edge network characteristics.

Furthermore, research literature shows that recent edge solutions tend to operate with centralized architecture. In this context, we currently lack a proper solution that deals with edge networks characteristics (i.e., dynamicity, heterogeneity, and uncertainty). Taking edge networks characteristics into account, enabling resource management techniques, and executing edge applications introduces several challenging tasks. In this setting, one critical task is to dynamically determine the most suitable edge device responsible for the deployment process and maintain edge applications' functionality throughout their life cycle. Furthermore, research literature shows that edge applications can be different and with complex requirements. Generally speaking, a plethora of point-to-point solutions shows that edge applications may require various resource management features or functionalities in order to run on an edge network. In this thesis, we consider some of the critical functionalities not addressed by researchers and which are required for operationalizing edge applications at the edge. For instance, some edge applications may experience different workloads at their runtime. In other words, resource demands for a specific edge application component may change over time. Therefore, it is a significant challenge to detect when an edge application component requires more resources or less such that resource demands are fulfilled at runtime. In this context, we currently lack a proper solution that deals with multi-dimensional elasticity (i.e., resource, quality, and cost) to guarantee the correct functionality of edge applications running on an edge network, respectively, on the entire computing infrastructure. Furthermore, edge applications may require a resource that cannot be trivially obtained from available resources. To solve such complex tasks within the edge system, we require a proper methodology and framework that combines readily available resources to achieve the goal. In the following, we present investigated research questions and the appropriate solutions to overcome non-addressed challenges in edge computing.

1.2 Research Questions

The research presented in this thesis is driven by three overarching research questions:

(**RQ**1) *What are appropriate system architectures that enable resource management in resource-constrained edge networks?*

As previously mentioned, traditional edge solutions tend to operate in isolation and with centralized architecture. Compared to cloud settings, the peculiarities of edge environments enhance the total complexity of operating resource management techniques. This is because edge networks are very volatile environments where

resource-constrained devices may fail easily. In cloud computing, complex functionalities can operate on a single computing entity; meanwhile, such a claim is hardly achievable in edge environments since low-powered edge devices have limited resource capabilities. Furthermore, unlike the cloud, the literature lacks a clear definition and consensus on what edge infrastructure is in concrete, how computing devices are organized on the edge, and how they communicate. Nonetheless, it is currently unclear (a) whether decentralized resource management techniques can overcome challenges introduced by edge environment characteristics, (b) what are essential resource management functionalities required to maintain the desired edge application functionality throughout its entire execution, (c) which would be the most suitable application runtime platform to run application software components on resource-constrained edge networks, (d) which are the most suitable approaches for resource coordination when a requested resource cannot be obtained trivially, and (e) how modern edge-based systems can support a wide range of edge applications to run correctly on edge networks.

(**RQ**2) *What is an appropriate way to automatically discover and utilize heterogeneous resources in resource-constrained edge networks?*

A fundamental aspect of any modern edge-based system is resource discovery, which is complex in heterogeneous edge networks and IoT settings. As previously mentioned, there is no clear definition and consensus on what edge infrastructure is precisely. We hypothesize that edge devices in proximity form an edge network, or as we refer to it, as an edge neighborhood[3]. Accordingly, the resource discovery process in such settings plays a crucial role in supporting other resource management techniques (i.e., resource allocation, resource migration, and resource coordination) to achieve their objectives. However, it is currently unclear (a) which would be the most suitable communication protocol to enable forming edge networks with resource-constrained edge devices, (b) which would be the most suitable way to represent resources (i.e., IoT resources such as sensors, actuators, etc., and edge devices), (c) which would be the most efficient way to enable automatically discovering resources within an edge neighborhood, and (d) which would be the most suitable network organization type (i.e., hierarchical, P2P, etc.) to support scalability aspects.

(**RQ**3) *How can elasticity features be executed on low-powered computational resources in edge networks?*

Over-provisioning or manual resource allocation in edge networks is highly unsatisfying concerning application performance and utilization of available resources at the edge. As we may have various edge networks in different domains (e.g., smart building, smart home, smart factory, smart city, drone network, etc.), elasticity features at the edge would not target only just resources and their capacity to scale but also their relations with the different types of costs and quality. The complexity

[3]In this thesis, we use the terms edge network and edge neighborhood interchangeably.

increases furthermore when different metrics, stakeholders, and perspectives are considered. However, it is currently unclear (a) whether elasticity features can be executed on resource-constrained and low-powered edge devices, (b) which would be the most suitable way to specify elasticity requirements for edge applications, (e) which would be the level of granularity in edge applications to specify elastic requirements, and (d) which would be the most suitable way to control edge application elastic requirements while taking into account edge network characteristics.

1.3 Scientific Contributions

This thesis aims to advance the current state of edge computing platforms that target edge applications to enable more straightforward configuration, deployment, and management on heterogeneous and dynamic edge infrastructures. We do this by developing novel resource management mechanisms with decentralized design, self-adaptation, and resilience. Accordingly, the proposed mechanisms aim to significantly lower the barrier to enable running edge applications in volatile edge networks. Furthermore, the proposed mechanisms are evaluated through the experiments conducted on testbeds comprised of low-powered ARM-based edge computers. Our contributions to state-of-the-art are by addressing the research questions of this thesis. These contributions are:

(**C**1) *Towards a decentralized edge-based system.*

The broad range of edge application requirements concerning latency, software and IoT resource dependencies combined with edge networks' heterogeneous and dynamic nature requires novel methodologies and flexible resource management frameworks that allow deploying, executing, and managing such applications running on resource-constrained devices. The contribution in this stage is twofold: (1) we introduce a conceptual platform that envisions developers to provide an environment to design edge applications and specify their requirements, and (2) we propose a decentralized edge-based framework capable of deploying and managing edge applications at the edge. First, we identify the core resource management functionalities required at the edge to efficiently utilize available resources and maintain the desired application functionality throughout its entire execution. Then, we investigate several aspects regarding edge system architectures, edge infrastructure architectures, and application runtime platforms. Lastly, through the presented arguments, we advocate the necessity to design modern edge-based systems into a decentralized architecture that is scalable in numbers and flexible and easily expandable in terms of its functionalities (i.e., C1.1, and C3 contributions). We present the contribution originally in [DM20, DM21] and provide an extensive description in Chapter 2.

(**C1.1**) *A methodology and technical framework for engineering resource coordination for the edge-enabled IoT.*

This contribution proposes a methodology and technical framework for engineering resource coordination for the edge-enabled IoT. The contribution is twofold: (1) we provide a design-time methodology to specify semantic annotations to arbitrary resources, and (2) a boolean satisfiability problem (SAT)/satisfiability modulo theories (SMT) solver placed on a low-powered edge device leverages bounded model checking as the foundational technique to compute coordination plans that satisfy device, edge, and system goals. For the former, by introducing the semantic annotations, we enable specifying what a resource requires to be operational and what effects its invocation has on other resources. For the latter, each edge device active within the edge-based system has its objectives captured in a goal model. Within our proposed approach, we adopt a form of discrete goal-modeling to capture the objectives of edge devices. As the primary coordinator within the system, the edge device receives device requests seeking to achieve some goals that may depend on other resources. The coordination process (i.e., resource matchmaking) occurs on edge, which figures out how to combine available resources to produce a plan for the requester device. We present the contribution originally in [TMD19], and provide an extensive description in Chapter 5.

(**C2**) *A technical framework to automatically discover resources in resource-constrained edge networks.*

Resource discovery is among five critical objectives of resource management in edge-based systems [TNT18]. Resource allocation techniques (i.e., at the application deployment phase) and resource migration techniques (i.e., at the application runtime) require information regarding the edge infrastructure and available resources. This led us to provide a twofold contribution to handle the complexity to discover resources automatically in edge networks: (1) we propose a solution to build edge networks as a flat model with edge devices organized into clusters, and (2) we propose a decentralized resource discovery mechanism that enables discovering resources by replication their metadata information in an automatic manner in edge networks. For the former, the resource discovery complexity is handled by introducing the configurator of the system and other cluster coordinators. Assigning the configurator and coordinator roles dynamically overcomes challenges with network scalability and uncertainty of edge networks. For the latter, available resources are discovered by replicating resource metadata by participating edge devices in an edge network. The proposed solution is characterized by the absence of centralization, as edge devices exchange metadata about available resources within their scope in a peer-to-peer manner. The resource discovery mechanism also increases the number of eligible deployment plans generated by the resource allocation mechanism for edge applications with IoT resource dependencies. The contribution is a notable step towards a better understanding of edge computing

requirements as well as edge infrastructure architecture requirements. We present the contribution originally in [MD21b], and provide an extensive description in Chapter 3.

(**C3**) *A technical framework for controlling elasticity in edge applications running on resource-constrained edge networks.*

Edge application developers or domain experts must specify high-level operational and non-functional application requirements to satisfy the QoS demands inherent to a specific edge-based system. Developers must express the context in which applications are allowed to run, resource requirements, and elasticity requirements (i.e., resource, quality, cost) in a high-level way [DGST11]. To realize the underlying premise of edge computing in IoT scenarios, we must develop a solution that hides all internal complexities both from users and application developers.In this contribution, we extend the prototype [MD21b] with a lightweight mechanism that enables deploying and controlling elasticity in edge applications at the edge. Such a mechanism allows for easy configuration, deployment, and operation of applications on top of heterogeneous edge infrastructure. Edge applications are executed in a runtime that considers the heterogeneity of edge resources giving applications seamless access to existing edge infrastructure. The proposed framework re-configures edge applications as necessary to meet their specified requirements (i.e., elasticity requirements). We present the contribution originally in [MD21a], and provide an extensive description in Chapter 4.

1.4 Thesis Structure

This thesis is based on the contributions of the original research papers published in journals, conferences, and book chapters. Some sections may have been extended or reworked in order to fit the overall context of the thesis. The remainder of this thesis is structured as follows:

- Chapter 2 investigates and discusses some of the research challenges in enabling resource management in a decentralized manner at the edge. First, this section explores resource management objectives and identifies those that are not yet thoroughly researched in the literature. Then, through real-life use case scenarios, three design goals that need to be established by modern edge-based systems are identified. Lastly, this section proposes a platform and edge-based framework that enables executing resource management functionalities in a decentralized manner at the edge.
- Chapter 3 presents a decentralized approach for resource discovery in edge networks. Furthermore, this section offers a feasible solution to build edge networks as a flat model with low-powered edge devices organized in clusters.

- Chapter 4 presents a lightweight mechanism for controlling resource elasticity in edge applications running at the edge. Furthermore, the proposed solution enables developers to characterize their edge applications by specifying elasticity requirements.
- Chapter 5 presents a methodology and framework for engineering resource coordination at runtime.
- Chapter 6 concludes this thesis, briefly discusses scientific contributions, discusses limitations, and provides future research.

CHAPTER 2

Towards Decentralized Edge-Based Systems

In the past few years, researchers from academia and industry stakeholders have suggested adding more computational resources (i.e., storage, networking, and processing) closer to end-users and IoT domains, respectively, at the edge of the network. Such computation entities perceived as *edge devices* by processing IoT sensory data aim to overcome high-latency issues between the cloud and IoT domains. Processing IoT data streams closer to end-users and IoT domains can solve several operational challenges. Since then, many application-specific IoT systems have been introduced, mainly hard-coded, inflexible, and limited extensibility for future changes. Additionally, most IoT systems maintain a centralized design to operate without considering edge networks characteristics such as dynamicity, heterogeneity, and resource-constrained devices. In this chapter, we investigate and discuss some of the research issues, challenges, and argue potential solutions to enable: (1) resource management in a decentralized manner at the edge and (2) deploying and scaling edge applications on-premises of Edge-Cloud infrastructure. We present a comprehensive discussion of the three-tier Edge-Cloud architecture, edge-based system architectures, infrastructure architectures, and application runtime platforms. Then, we outline our vision by introducing a conceptual platform that aims to enable easy configuration and deployment of edge applications on top of heterogeneous. In addition, we present a decentralized edge-based system responsible for deploying, executing, and managing edge applications at the edge. Furthermore, after careful analysis of various IoT scenarios, we identify the core resource management functionalities required to support the execution of various edge applications from different domains.

The rest of the chapter is structured as follows. Section 2.2 gives an overview of Edge-Cloud architecture. In Section 2.3, we describe four use cases by emphasizing the benefits of the Edge-Cloud continuum and the need for distributed edge applications at the edge. Section 2.4, presents our conceptual framework to deploy and manage edge

applications in the Edge-Cloud infrastructure. In Section 2.5, we extensively discuss resource management categories, edge system and infrastructure architectures, and edge application runtime platform. The summary in Section 2.6 concludes the chapter.

2.1 Introduction

The Internet of Things (IoT) is becoming more prominent in our daily lives and today's society overall. Many services in various domains such as Industrial Manufacturing, Healthcare, Lifestyle, Automotive, and Smart Building are built on top of IoT technologies. At the same time, it is well accepted that a centralized architecture does not scale well regarding the enormous number of devices. Even though central computers' system infrastructure processing those data has improved by cloud computing technologies, satisfying edge applications'(i.e., IoT applications) stringent requirements has become challenging for a cloud-centric architecture. Transfering continuously vast amounts of sensory data for processing in the cloud results in high latency. In sharp contrast to a fully distributed and decentralized architecture (e.g., peer-to-peer network), many IoT services need to maintain a partially centralized design to operate the service. Research literature shows that a significant portion of decentralization is often achieved by delegating several functionalities from central servers to edge computing devices [SD16].

Edge computing has introduced edge devices as an intermediary entity between applications and IoT deployments, providing data or control facilities to the participating IoT devices. Edge devices are essentially computers close to the edge of the network, hence, closer to the sensors, which create the data streams to be processed later. Accordingly, these edge devices can be utilized to process data streams promising to satisfy the stringent requirements prevalent in IoT systems, including high availability, performance, and privacy [LTJ+19]. However, edge devices are usually considered resource-constrained with limited resources, referring to their different computational capabilities, including storage or processing facilities. For instance, providing a service for image processing or deploying edge applications (i.e., IoT applications) on a single edge device poses many limitations and a set of challenges in terms of processing capabilities, storage, and communication bandwidth. To this end, edge devices do not exist in isolation and must be able to collaborate with other edge devices. Thereby, interaction with other edge devices enables extending the scope of available resources and satisfying the computational requirements of real-time edge applications at the edge of the network.

The heterogeneity of IoT components is another significant aspect of the IoT era. Many IoT vendors provide different products across different layers and for the same type of components. Essentially, different products may have different operating systems, available support for programming languages, resource constraints, etc. In contrast to our expectations, each edge device's functions are hard-coded in most current implementations. This causes inflexibility and limited extensibility for future changes. This leads to a plethora of point-to-point solutions being developed based on proprietary protocols. For instance, any change being made to one IoT component leads to many possible changes

in many other components or in the whole IoT system [BM18]. Thus, leading to a situation that resembles the state of software infrastructure in the age of enterprise computing before the emergence of Web services and Service Oriented Architectures and their respective middleware infrastructure, such as Enterprise Service Bus. This intuition led us to design novel edge systems into a distributed, decentralized architecture that is scalable in numbers and concerning its heterogeneity, expandability and enhancement in terms of edge functionalities.

Nowadays, computer scientists and system engineers have been mainly focused on proposing multiple resource management techniques (resource allocation [LB20, DS21, PKBK20], resource discovery [MCB+21], or resource monitoring [FGB21]) at the edge. Existing resource management techniques are often deployed on cloud entities or statically placed on a powerful device such as a local server or a fog device residing at the edge [SKR+18, YHZ+17, ATD19]. Generally speaking, cloud-centric resource management solutions fail to satisfy stringent requirements of a latency-sensitive edge application since high volumes of IoT data streams must be transferred to the cloud. In sharp contrast to the cloud and fog devices which are considerably more powerful computation entities, distributed edge devices with embedded resource management features enable faster processing of such data streams as well as control applications' runtime aspects. However, executing complex resource management features on edge devices with limited computing stacks causes device failures when their computation resources are overloaded. To address such challenges, we identify three design goals that need to be established by modern edge-based systems, respectively resource management techniques:

- **Latency-aware and Proximity.** The edge computing paradigm aims at providing low-latency services for endpoint devices and end-users [GD18]. As we explore modern edge-based systems and their operating environments, placing a set of complex resource management functionalities statically on a low-powered edge device is impractical and may produce higher latencies and easily cause edge device failures. As a result, the decentralization aspects should be considered when designing edge-based systems. Furthermore, dynamically placing resource management functionalities in proximity to end-users becomes an increasingly inevitable requirement.

- **Expandability, Heterogeneity, and Dynamic Network.** Edge networks are scalable and dynamic environments consisting of heterogeneous devices that frequently join/leave over time. Edge-based systems should be able to flexibly utilize newly added resources at the edge. Furthermore, modern resource management techniques must deal with the dynamicity and uncertainty of the edge environment, as well as support expandability with new edge features (i.e., functionalities).

- **Adaptability.** Edge-based systems should be able to adapt to unexpected changes that might occur in the edge network. Thus, the edge-based system components, respectively resource management techniques deployed at the edge, should be

dynamically placed among available edge devices and continuously re-evaluated their placement decisions.

Recent developments within distributed systems have led to emerging commercial cloud-based IoT and cloud/edge integration solutions. Edge computing platforms such as EdgeX Foundry[1], AWS IoT Greengrass[2], or Google IoT Edge[3] promise to bridge the gap between the IoT and the cloud by providing a flexible runtime for applications running at the edge. However, these systems are extremely limited in their operational capabilities, missing elasticity features, and lack of self-adaptive mechanisms required in dynamic edge and IoT settings. Based on the discussion, we argue that decentralized and dynamic resource management techniques are appropriate to overcome challenges introduced by edge environment characteristics (i.e., edge network dynamicity, heterogeneity, and resource-constrained devices). Nevertheless, we foresee several challenges in engineering decentralized edge-based systems.

To that end, this chapter identifies some of the research issues and discusses challenges that are not yet thoroughly investigated, such as (1) resource management in a decentralized manner at the edge and (2) deploying and scaling edge applications on-premises of the Edge-Cloud infrastructure. These derive mainly from the fact that many aspects of edge computing require further investigation to realize edge computing's underlying premise. Thus, to assist in this process, we propose a conceptual platform and an edge-based framework that aims to enable easy configuration, deployment, and execution of edge applications on top of heterogeneous edge infrastructure. Our proposed approach shifts resource management features closer to the edge and dynamically places them in the most suitable edge devices. Furthermore, the proposed approach supports the systems to cope with the environment's dynamicity and uncertainty, including changes in their deployment topologies. Finally, we identify and discuss two critical edge features required to support edge applications execution properly.

2.2 An Overview of Edge-Cloud Continuum

In addition to fog and cloud, edge computing is seen as a critical architectural tier. The model makes it easier to deploy distributed, latency-aware applications at the edge. Furthermore, it can be considered as paramount to systems including (but not limited to) IoT deployments and the cloud, providing data and control facilities to participating IoT devices. So far, several surveys have been conducted to explain the model of edge computing, and its challenges [SCZ+16]. However, recent studies emphasize three-tier architecture as a useful way to enable more devices closer to IoT domains [AMD20]. To that end, the Edge-Cloud architecture [DAM19] is split into three tiers: the cloud tier, fog tier, and edge tier (as illustrated in Figure 2.1). In this section, a comprehensive description of each tier is presented.

[1]EdgeX Foundry, https://www.edgexfoundry.org/
[2]AWS IoT Greengrass, https://aws.amazon.com/greengrass/
[3]Google IoT Edge, https://cloud.google.com/solutions/iot

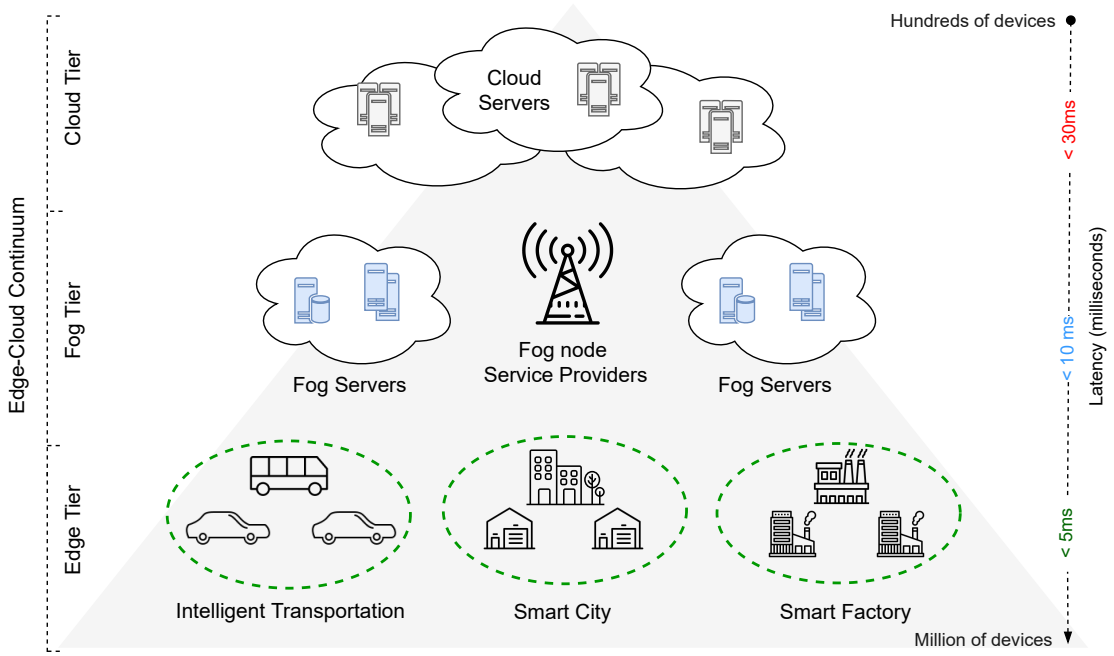


Figure 2.1: An overview of Edge-Cloud architecture.

2.2.1 The Edge Tier

The edge tier represents the lowest tier of the three-tier architecture. In essence, this tier represents the closest available computation entities called *edge devices* to IoT domains. As can be denoted in Figure 2.1, various domains such as smart city, intelligent transportation, or smart factories can benefit from available computation devices closer to IoT domains. Essentially, this leads to having various edge networks formed for different contexts. Generally speaking, edge networks are highly dynamic, heterogeneous, and resource-constrained environments. Such environments are composed of a set of low-powered edge devices with various computation capabilities (e.g., smartphones, smartwatches, Raspberry Pis, etc.) and IoT resources (e.g., sensors, actuators, etc.) connected to them or available in their surroundings.

Edge devices provide their computation and storing capabilities to process IoT data streams within very *low latency* generated by IoT resources. Essentially, these devices allow data streams to be processed as close as possible to data sources and to handle the most significant network traffic that may occur. Resource-constrained edge devices, however, cannot process large volumes of data. Consequently, processing such an amount of data on a single edge device may cause device overloading, resulting in high latency and poor overall performance. To address these challenges raised by resource-constrained devices, distributing various processes (e.g., edge application tasks) among edge devices at the edge is a critical requirement. More precisely, the placement of such processes

among edge devices becomes possible with the increased spectrum of resources provided by peer-to-peer edge networks.

An *edge network* provides a seamless opportunity for deploying edge-based systems (i.e., resource management techniques) and various edge applications closer to end-users. Accordingly, placing resource management techniques at the edge results in creating more *autonomous environments* and less dependent on the external settings (i.e., cloud or fog). For instance, in a smart home, residents should control their devices and process data locally without depending on cloud resources. This means that the edge-based system must provide functionalities locally to achieve such a user goal. Furthermore, a user may deploy an edge application (e.g., a smart health app) to provide services for processing health data transmitted from their wearable devices. However, it is worth noting that even though extending resource scope through edge-to-edge collaboration provides many benefits, there are still many complex tasks that cannot be divided into sub-tasks and processed at this tier. In such situations, edge devices must forward their processing to the upper tiers (i.e., fog or cloud).

2.2.2 The Fog Tier

The fog tier includes a collection of powerful (physical or virtual) devices responsible for managing, communicating, and exchanging resources between different edge networks. The core component of the fog tier is a *fog node*. This tier essentially represents the fog infrastructure (e.g., smart city network) where several fog devices are connected, offering various services, including computation and storage resources for edge networks and roaming end-devices in proximity. In sharp contrast to the edge networks, the fog infrastructure appears to be composed of stationary and powerful devices (typically managed and provided by Telco operators). This tier can handle more complex functions like processing large data streams, caching, device management, and privacy protection.

According to [IFB+18], fog computing provides a similar service model implementation as in the traditional cloud computing model. Thus, the following types of service models can be implemented:

- **Software as a Service (SaaS)**. Fog service providers may host various services (i.e., similar to the cloud computing Software as a Service (SaaS)) on fog infrastructure. Essentially, a fog service end-users (i.e., customers) may use providers' service running on fog infrastructure, specifically, on a cluster of federated nodes managed by the provider.

- **Platform as a Service (PaaS)**. Fog service providers allow end-users to deploy their applications onto the platforms (i.e., similar to the cloud computing Platform as a Service (PaaS)) on fog infrastructure. In essence, end-users control and configure their deployed applications and running environment. However, they do not control and manage fog platform(s) and infrastructure(s).

- **Infrastructure as a Service (IaaS)**. Fog service providers allow end-users to provision processing, storage, and other infrastructure-specific resources available on fog infrastructure (i.e., similar to the cloud computing Infrastructure as a Service (IaaS)). A customer can essentially deploy and run various software, while a consumer may control and manage the storage, operating system, and deployed applications.

One can notice that edge and fog tiers provide almost similar features. Both paradigms foresee enabling more computation resources in proximity to end-users and IoT domains. However, the most significant difference between the two tiers is *administrative differences* and *responsibilities*. Furthermore, fog nodes (e.g., deployed in base stations) may provide their services for larger geographical areas. For instance, intelligent transportation systems may benefit from connecting and processing vehicle data in fog infrastructure. Nonetheless, both tiers provide low-latency services since the end devices are closer to the source where the data is produced and consumed.

2.2.3 The Cloud Tier

The cloud tier provides "unlimited" computational and storage resources. This tier includes cloud servers deployed far away from the end devices and IoT domains. In essence, cloud-based servers perform computer-intensive operations obtained from the architecture's lower tiers. These environments have advanced features for both service providers and service consumers. For instance, service consumers can configure their runtime environment, configure deployed applications, security control, and so on. Along these lines, the cloud computing utility has been seen as a critical component for designing, deploying, and executing IoT platforms that promise to meet the general population's daily needs.

Despite the numerous resources and advanced features provided, this paradigm faces increasing challenges in meeting new IoT applications' stringent requirements. Specifically, geographically distributed IoT devices with intensive data generation cannot efficiently utilize resources available in cloud environments [DAM19]. On the one hand, transferring such intense and large amounts of data to a centralized cloud over Wide Area Networks (WAN) generates latencies. Additionally, service unavailability due to the non-persistent connectivity or eventually scheduled system maintenance (i.e., cloud-side) is another critical challenge that IoT systems may face during their runtime. On the other hand, real-time distributed apps require quick response time, high-availability, and increased privacy, often not met by a centralized environment such as the cloud.

2.3 Edge-Based IoT Use Cases

Understanding current and future requirements of edge applications from various domains is a critical challenge for the success of any modern edge-based system. Thus, through careful analysis of real-world IoT scenarios from multiple disciplines, we show the

importance of shifting various functionalities closer to the edge and dynamically placing them in the most suitable edge device. Furthermore, we identify multiple resource management features required at the edge by analyzing various use case scenarios.

2.3.1 IoT Safety Application

Consider emergencies such as natural disasters (e.g., earthquakes, fires, floods) in a city [DM21]. Natural disasters such as earthquakes can affect various city areas, destroy infrastructure, cause injury or death, and trap people under buildings. In such situations, time is valuable, and drones could be used to analyze the situation and assist rescue teams in locating and communicating with victims trapped under a collapsed building. In such a scenario, multiple connected drones are essentially edge devices forming an *edge neighborhood*. Drones flying over the city's affected areas (i.e., neighborhoods) assist rescue teams in locating people trapped under a collapsed structure. Each drone has various computation capabilities and integrated sensors (e.g., radar sensors, infrared cameras, electronic noses, etc.). We consider drones as multipurpose devices where the rescue teams may deploy various services depending on the emergency. Furthermore, we consider three-tier Edge-Cloud infrastructure (i.e., cloud, fog, and edge) [DM20]. Fog devices (i.e., server-graded hosts) placed in base stations provide computational and storage capabilities to edge neighborhoods. In addition to that, base stations may provide docker charge stations for charging drones. Cloud hosts can be used to store data for long terms.

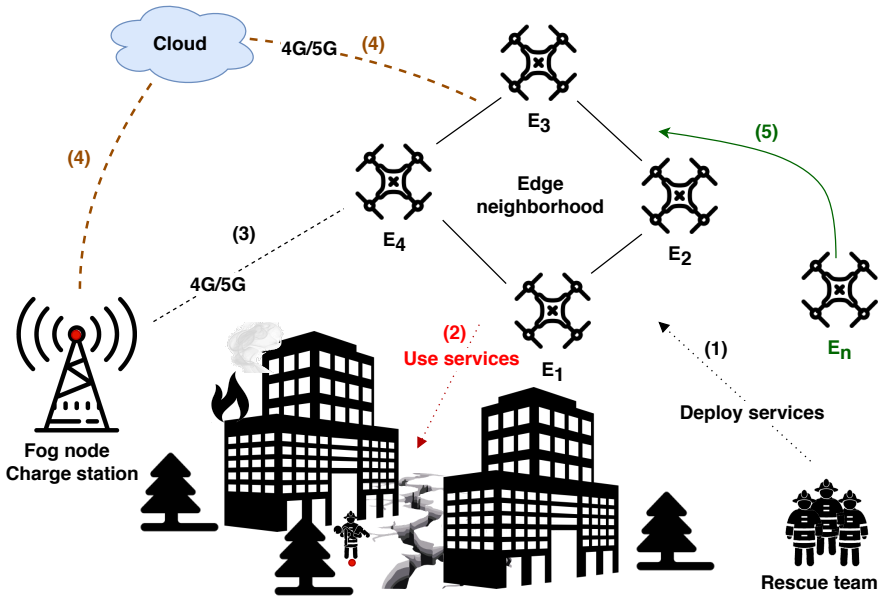


Figure 2.2: IoT safety application

We assume that drones are connected via a wireless connection provided by ground users (i.e., rescue team) or by drones [MSBD18] covering a particular city area (e.g.,

neighborhood). Based on the situation seen in Figure 2.2, we assume that the rescue team deploys (1) a public safety IoT service that detects a dangerous zone in the affected area (i.e., discovering cracks, smoke, hazardous gases, etc.). Such a service aims at helping rescue teams (2) find a safe path and avoid danger zones. The service depends on various resources such as multiple infrared cameras, radar sensors, and an electronic nose integrated into different drones. Since each drone is a potential candidate to run the service, it is evident that each edge device should automatically discover resources in a decentralized manner and make them available at runtime. In such use case scenarios, we cannot depend on the service availability [SUS14] offered from physically static entities (3-4). Additionally, edge-based systems with centralized architecture cannot run properly due to the network dynamicity (i.e., drones may join (5) and leave often). This use case represents the motivation and running example used throughout Chapter 3.

2.3.2 Smart Health Application

The falling asleep problem has been discussed previously in many research papers [SCZY17, FSH17, MBP13] and used as a motivating example for various proposed solutions. This problem generally belongs to the scenarios categorized into residents' comfort and convenience category. Suppose a resident has a smart health application running on his phone responsible for monitoring the resident's vital signs. The health application essentially collects data from a wearable ECG sensor attached to the human body through a smartwatch [GD18]. Specifically, the health application monitors the incoming data generated by the wearable device and reacts in real-time when a critical event occurs. Usually, a system architecture composed of a smartwatch and a smartphone represents a reasonable choice for this type of application. However, since smartphones have limited battery energy, shifting data (i.e., health applications [PAG+19]) to the nearest edge device may be required when the battery level decreases at a critical level. Nonetheless, transferring data to the cloud consumes smartphone energy even faster.

In this scenario, it is assumed that a person in a living room, sitting on a sofa and watching TV, gradually falls asleep. Multiple edge devices deployed in rooms are responsible for changing surroundings accordingly to get a comfortable living. In this case, the edge device's responsibility in the living room is to alter its surroundings and ensure that the resident can sleep comfortably. These changes include turning off the TV and lights, changing air conditioner settings, and locking the entrance door (e.g., see resource coordination [TMD19]). Meanwhile, as the resident forgets to plug the smartphone into power, the battery nearly runs out. Therefore, the smartphone requests the nearest edge device to handle and process the generated data from wearable devices (see Figure 2.3).

Our nearest edge device in the living room utilizes almost all its processing capabilities to ensure that the resident sleeps comfortably (i.e., measuring room temperature, etc.) as well as runs a similar health application $\mathtt{A}_1$. Since the health application at the edge was idle, it requires more processing power and memory after the wearable device pumps the data into the system. Nevertheless, the edge device doesn't have enough computation capabilities to execute both applications concurrently. Thus, the health application faces

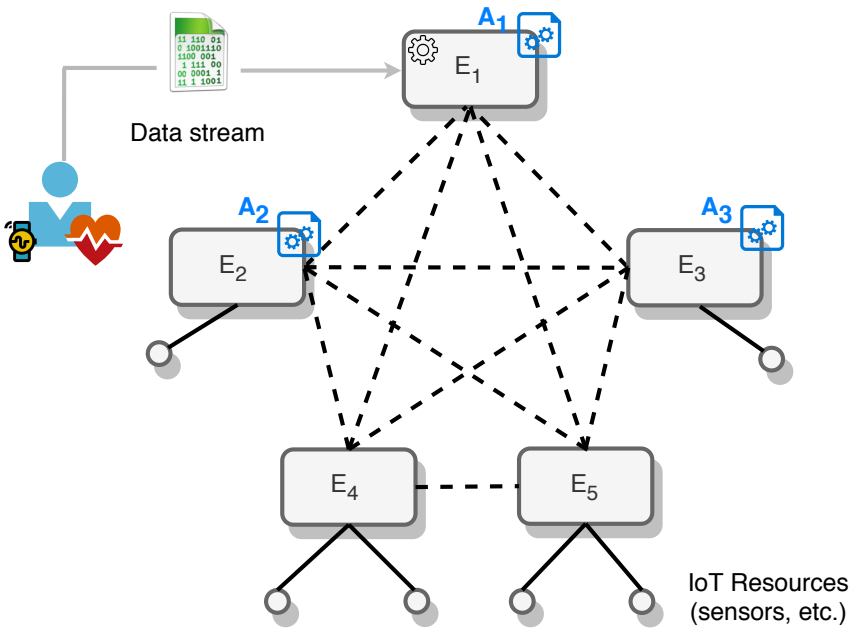


Figure 2.3: Smart home health application.

higher latency in accepting and processing incoming data. Meanwhile, other edge devices in the smart home may remain idle. To overcome such a gap, the health application should create new instances (e.g., A_2, and A_3) and deploy them on other edge devices or on-premises of Edge-Cloud infrastructure to meet application demands at runtime. Thus, it is evident that bringing the *elasticity features* at the edge is crucial. This use case represents the motivation scenario considered in Chapter 4.

2.3.3 Smart City Application

Consider a modern city containing various neighborhoods and parks where various devices are embedded, providing intelligent functionalities. Naturally, waste bins are located in neighborhoods as well as parks, and traffic lights may be deployed to facilitate municipal vehicles; ambulances or recycling trucks should be presented with green traffic lights when applicable. Moreover, irrigation facilities situated in city parks should automatically be operational when the detected soil moisture is below a threshold of 20%. However, this should not occur when the park is crowded with visitors. The city administration may also impose other limits on the overall system's operation; for example, to minimize citizen interruption, irrigation and rubbish collection should not occur at the same time while every neighborhood should have empty waste bins.

Notice that a smart city is an instance of an edge-based system; several sensing or actuating devices are needed to realize it. Devices or certain *scopes* in the city (e.g.,

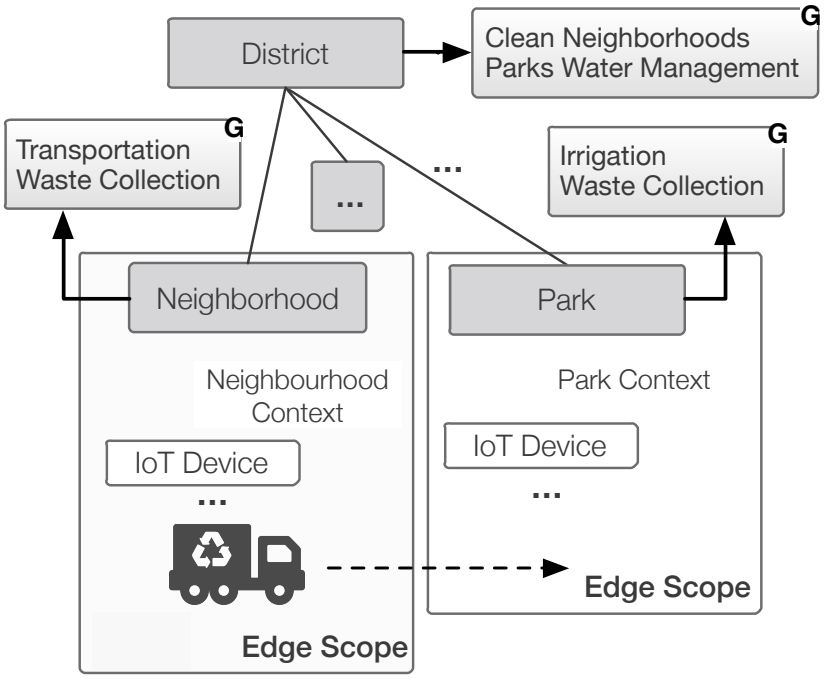


Figure 2.4: Pervasive resources in a smart city.

neighborhoods or parks) have various goals, which, when the system is in operation, may conflict (e.g., moisture may drop while a recycling truck arrives in a park). Moreover, the particular conditions and configurations of IoT devices are unknown at design time; we require no knowledge of recycling trucks, presence or not of irrigation in parks, for example. It is then evident that coordination among IoT devices is required to fulfill their various goals. Such goals can be achieved through utilizing various AI techniques (e.g., [TMD19]) which are well-known approaches developed to solve planning autonomously and without human intervention. This use case represents the motivation scenario considered in Chapter 5.

2.4 Towards Decentralized Edge-Based Systems

As we are acquainted with the requirements of various edge applications, in this section, we introduce a conceptual platform that aims to enable easy configuration and operation of edge-based systems on top of heterogeneous edge infrastructure. We present a decentralized edge-based resource management mechanism called the configurator responsible for deploying and managing edge applications at the edge. Finally, based on the analysis of real-world scenarios, we identify and discuss two critical functionalities required for edge-based systems to support the execution of applications at the edge.

2.4.1 A Platform and Runtime for Edge Computing in the Internet of Things

Edge neighborhoods offer a seamless opportunity for end-users to customize their environments with various services to support their daily activities and improve their living comfort. As illustrated in Figure 2.1, we may have multiple edge networks (e.g., smart building, smart home, smart factory, smart city, drone network, etc.). The end-users may deploy various edge applications and customize their edge neighborhoods based on their needs. For instance, in a smart home, residents may deploy an edge application that provides them a service to back up their smartphones automatically when they're home. Or, in a smart building (e.g., museum environments), the system administrator may deploy an edge application that provides a service for visitors to interact with their surrounding objects through virtual reality (VR) [SSL20].

Even though the computation scope expands with forming edge neighborhoods, one should note that edge neighborhoods cannot always provide enough resources to execute edge applications (e.g., image processing tasks, etc.). Edge environments are resource-constrained and with limited resources. Specifically, edge applications or their software components may experience various workloads over time (i.e., periodically, continuously, or unpredictable). For instance, an AI-based surveillance security application at the airport may analyze numerous cameras at once and automatically detect passengers with a fever. The number of passengers changes continuously as well as the application resource demands. Consequently, the AI-based surveillance security application requires to automatically maintain the desired service quality throughout its entire execution. Thus, we need a lightweight framework that can be easily deployed on low-powered edge devices and enable edge-based systems to expand with enhancements of new resource management features (i.e., via edge modules).

To fill this technological gap, we propose a platform comprises of two main parts: (1) a cloud-based IoT platform and (2) a decentralized edge-based framework (see Figure 2.5). A cloud-based IoT platform provides several types of services and contains *edge application models* that users can download to their edge neighborhoods. An edge application model essentially describes in detail application components, their resource requirements (i.e., hardware requirements and edge functionality dependencies (i.e., *edge modules*), and elastic requirements specified by the application developer at the design time ①. We assume that an edge module is a software component (i.e., serverless function [ATC+21] or container-based) that adds a specific feature to an existing edge-based system at runtime. For instance, the WebAssembly Runtime (Wasm)[4] is a potential solution that enables adding functionalities to an edge-based system at runtime. More specifically, Wasm is lightweight, high-performance, and optimized for serverless function executions and the execution of applications in a decentralized manner at the edge. Nevertheless, several approaches have been proposed to engineer modular systems [BDW13]. Thus, we do not explore this feature in this thesis, and further investigations remain as future work. In

[4]WebAssembly, https://developer.mozilla.org/en-US/docs/WebAssembly

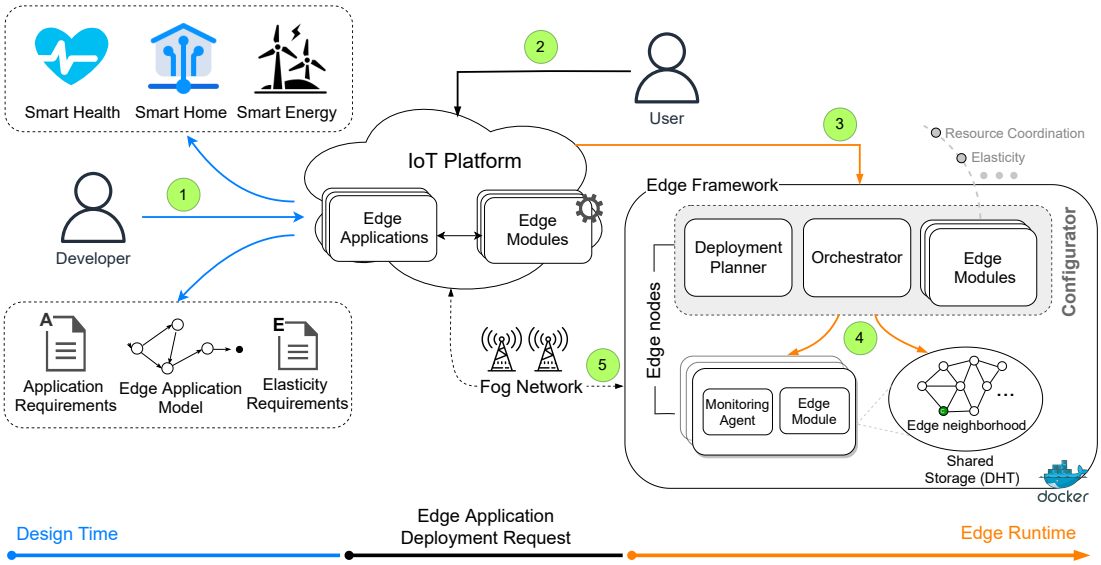


Figure 2.5: An overview of a conceptual IoT platform and edge framework.

this thesis, we investigate and provide our contribution to enable two edge-based system functionalities such as (1) elasticity features (C3) and (2) resource coordination features (C1.1). Furthermore, the cloud tier assists edge neighborhoods in identifying the closest fog devices (e.g., location coordinates) as well as providing storing and computation resources. In the smart city context, fog services enable sharing of resources among nearby edge neighborhoods as well as authenticating them to the city's edge network backbone (i.e., fog network). For instance, a fog-based blockchain platform may provide service to edge neighborhoods to share their available resources with others along with the payment term ⑤.

Within this thesis, our focus resides in the proposed decentralized edge-based framework that enables easy system configuration, deployment, and operation of edge applications on top of heterogeneous edge infrastructure. Because resource management is challenging in dynamic environments with multiple heterogeneous edge devices, we advocate that each edge device should provide resource management functionalities. Furthermore, deploying an edge application requires real-time infrastructure-specific information. Accordingly, it is computationally and network demanding to monitor resources from each edge device throughout the network. Therefore, first, we need to determine which edge device must take the responsibility to act as a *control mechanism* for controlling deployment aspects and managing applications' runtime aspects at the edge. We refer to this mechanism as the *configurator*. The configurator is self-adaptable and configurable through the *configuration file* which is shared system-wide. The system designer specifies and configures several parameters specific to a particular edge-based system. We discuss the configuration aspects continuously throughout the following two chapters.

As illustrated in Figure 2.5, the proposed edge-based framework equips edge devices

with the same functionalities such as (1) deployment planner, (2) orchestrator, (3) edge monitoring agent, and (4) edge modules. There is one and only one edge device in an edge neighborhood that operates as the configurator. The configurator is responsible for monitoring the edge infrastructure, deploying, managing, and monitoring edge applications at the edge. The core component of the edge-based framework is the orchestrator component which is enabled on all edge devices. The orchestrator's first critical task is to determine the most suitable edge device where the configurator should be placed (deployed) and run (executed) at the edge. Second, the orchestrator enables the communication between edge devices peer-to-peer, provides a resource discovery mechanism, resource manager to monitor resources, and provides a gateway component to connect the edge neighborhood with external networks (i.e., cloud and fog). The following chapter investigates and shows our contribution (C2) on how to enable resource discovery and determine the most suitable edge devices to run the configurator.

After the configurator is determined, the user may download ②-③ an edge application from the cloud and deploy it to the edge neighborhood. The deployment planner is responsible for analyzing application requirements and generating valid deployment plans. The scheduler considers all the requirements mentioned above, get the resource manager's infrastructure state, and generates an eligible deployment plan (if it exists) ④. Thus, it determines where the edge application components and data must be processed to fulfill their requirements. To generate such deployment plans in the Edge-Cloud infrastructure, we consider the approach in [BF17]. Furthermore, the monitoring agents deployed at each edge device periodically update the resource manager with infrastructure-specific metrics (i.e., CPU, storage, etc.) and application-specific metrics (i.e., non-functional parameters). Furthermore, some edge applications may require additional and specific system functionalities. For instance, resource coordination service [TMD19] at the edge requires specific functionalities (e.g., SMT [BCD$^+$11a]) to generate valid coordination plans that will enable the service to achieve its goals. Or, an edge application to run properly may require an elasticity feature running in the edge neighborhood. Thus, such edge modules are automatically downloaded after user approval to deploy an edge application.

2.4.2 Elasticity as an Edge Feature

Future edge-based systems for the described Edge-Cloud architecture need to hide their operational complexity from application developers as well as from end-users. Specifically, application developers should not have to deal with the edge network setting's heterogeneity, dynamicity, and expandability. Developers should be able to express the context in which edge application components are allowed to run, their system dependencies, and their requirements (e.g., QoS, elastic requirements, etc.) in a high-level way [DGST11]. For instance, an application developer may specify the elastic requirements of an edge application (e.g., health application). To interpret these requirements and enforce required operations in order to keep the desired service quality, we require the *elasticity feature* at the edge. Specifically, the elasticity feature provides a

runtime mechanism that interprets elastic requirements specified by the developer. Such feature is deployed and enabled automatically at the edge-based system after the user requests to deploy an edge application with elasticity dependency.

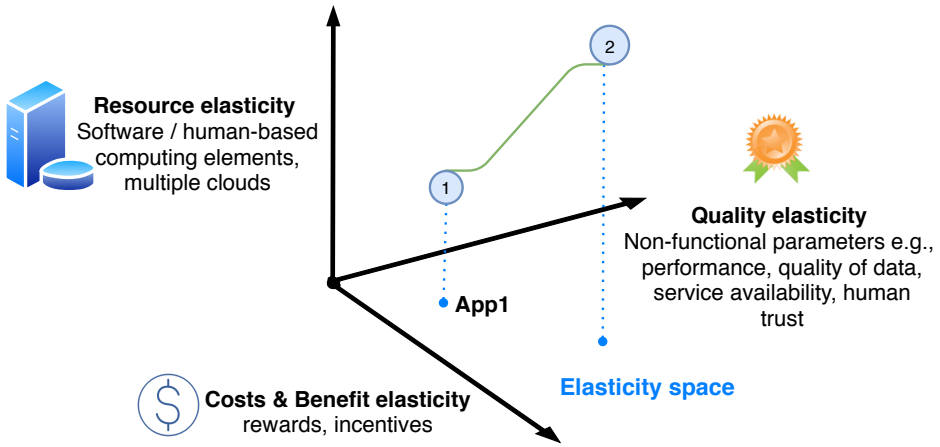


Figure 2.6: Elasticity space at the Edge-Cloud architecture.

Bringing elasticity properties to the edge is crucial for the future of edge applications. In the Edge-Cloud architecture, elasticity targets not just resources and their capacity to scale, but also their relations with the different types of costs and quality (as illustrated in Figure 2.6) [DGST11]. In this context, various stakeholders may be involved in specifying elastic requirements. For instance, the developer could specify that the latency between application components must not reach 20ms without determining how many resources should be used to achieve the desired state. The edge network provider could specify its resource utilization schema; for example, when overall utilization at the edge is higher than 90%, it enables scaling out the edge. As potential candidates for defining these requirements are : Simple Yet Beautiful Language [CMTD13] and SLOC: Service Level Objectives for Next-Generation Cloud Computing [NMP+20]. In this thesis, we consider the SYBL language for specifying the elasticity requirements of edge applications. Chapter 4 shows our contribution (C3) on how to enable elasticity features to be executed on low-powered computational resources in edge neighborhoods. We propose a methodology and a framework to control application elasticity at runtime.

2.4.3 Resource Coordination as an Edge Feature

IoT has enabled objects and devices, such as sensors, actuators, and other appliances, to connect and collaborate to achieve user goals. Accordingly, engineering edge-based systems is a challenging task partly due to the *dynamicity* and *uncertainty* of the environment, including the involvement of the human in the loop. Through resource

coordination techniques, users, edge applications, or IoT resources should achieve their goals seamlessly in different environments.

Consider situations similarly as is presented in the smart city application (see Section 2.3.3). Suppose an edge application requests a nearby edge device for a resource that cannot be trivially obtained from resources readily available. In that case, some combination of resources of other IoT devices must be derived. This thesis proposes a novel approach to achieve such objectives. The proposed approach utilizes SAT/SMT solver [BT18] situated on a low-powered edge device which leverages bounded model checking techniques [BCC+03] at runtime to fulfill the objectives of local IoT devices or edge applications by coordinating available resources in its scope based on the currently active context. A SAT/SMT solver is an efficient algorithm that determines whether or not a formula (i.e., problem instance) is satisfiable or not. The biggest challenge is modeling problem instances in boolean logic for SAT solvers and first-order logic for SMT solvers [End00]. Chapter 5 shows our contribution (C1.1) on how to achieve resource coordination for the edge-enabled IoT.

2.5 Resource Management and Edge Architectures

To achieve the goals mentioned in the proposed conceptual framework, we first discuss resource management categories and investigate edge computing systems and infrastructure architectures. Then, we explore communication types among edge devices and open challenges for developing and deploying edge-based systems in a decentralized manner on edge networks. Afterward, we research application runtime platforms responsible for executing edge applications on heterogeneous edge devices or in available devices within the computing continuum. Finally, potential and feasible solutions are given for each of the challenges this thesis addresses.

2.5.1 Resource Management

Resource management is a critical challenge for the overall edge application performance and QoS. According to [TNT18], resource management is divided into five categories:

- *Resource estimation*: The first requirement in resource management is to estimate how many resources are needed to execute a specific edge application. Resource estimation is an essential feature to handle fluctuations in resource demand in order to maintain the quality of service for end-users. However, the biggest challenge in resource-constrained environments is that resources are often over-provisioned to handle the possible incoming workload at any point in time. Thus, introducing elasticity features at the edge solves several challenges in avoiding undesired situations with resource over-provisioning or under-provisioning.
- *Resource discovery*: On the contrary to resource estimation, resource discovery enables discovering resources that are available for use. Information about re-

sources, such as their availability, location, and status, is among the data that edge-based systems require at their runtime. For instance, when deploying an edge application at the edge, a management system needs to know where resources are located and their availability status. Thus, resource discovery provides the core information required during the application deployment phase. Nevertheless, due to the system characteristics (i.e., dynamicity, heterogeneity, resource-constrained devices), resource discovery is among resource management categories that are not well-researched. In fact, a few research works that exist in literature employs a locally centralized strategy for resource discovery at the edge [AV15, AADP15].

- *Resource allocation*: Two previously mentioned resource management categories help us to gather information about available resources and the edge application requirements. Based on this knowledge, the resource allocation techniques aim to find the most suitable deployment plan to deploy edge applications at the edge. Therefore, such methods aim to find where to place application components such that their requirements are satisfied.

- *Resource sharing*: In this category, edge devices within the edge-based system are assumed to collaborate to fulfill various application requirements or to achieve a common goal. Nevertheless, resource sharing is not limited only to edge devices. For instance, in a smart city, we may have various edge networks, respectively edge-based systems. In such environments, edge-based system owners may allow sharing their unused resources with the neighbor edge systems along with the payment term.

- *Resource migration*: Edge-based systems are very volatile environments where heterogeneous devices may join or leave without prior notification. The mentioned system characteristics require methods to decide how to change initial application deployment at runtime such that the deployment continues to satisfy the initial application's resource demands. Based on the literature review [TNT18], edge application's components are migrated based on the adaptation processes, which rely on the context information or re-using the resource allocation techniques to find new deployment plans.

This thesis combines resource discovery, resource allocation, and resource migration to deploy and manage edge applications on top of a heterogeneous and resource-constrained edge network.

2.5.2 Edge System Architectures

Researchers and computer scientists both in edge and fog computing have been mostly focused on proposing multiple techniques for resource allocation problems aiming to minimize various trade-offs such as latency, bandwidth, energy consumption, or maximizing the utilization of resources at the edge. In general, IoT systems can be deployed according to the following models [ADS18, RRL+14]:

1. *Everything in the cloud model*: In this model, the software components of IoT systems are placed in the cloud [Ray16]. It is suitable when the systems require significant elastic processing and storage capabilities or when their constituents are scattered in various areas. It has higher latency rates compared to the edge-based model.

2. *Everything in the edge model*: The software components of IoT systems are placed in networks of more constrained devices with respect to the cloud (e.g., local servers and gateways) that are at the edge of the network [AMD20]. In this model, the computational capabilities are lower than those within the cloud-based model.

3. *Hybrid Edge-Cloud model*: The software components of IoT systems are distributed across the cloud and the edge of the network [MJK+21, MRB20, VFD+16]. Thus, it enables exploiting the advantages of the other two models. For instance, it supports deploying the components that should perform processes with low latency in the edge of the network, and deploying those that perform resource-demanding processes in the cloud.

These deployment models have different properties in terms of response time, availability, privacy, and other quality characteristics [ADS18]. A common approach for resource management in edge-based systems is to assign components across the computing continuum by considering several factors such as processing capability, bandwidth, or energy. For instance, an edge-based system composed of a set of system components (i.e., controlling module, scheduler, resource manager, etc.) can be deployed based on the above mentioned models. Regardless of the deployment models, edge-based system control can be exerted by just one entity or shared among several independent computing entities. Throughout this thesis, we focus only on edge-based systems deployed based on everything in the edge model. Thus, we analyze and discuss the pros and cons of three main edge system architectures: (i) *centralized*, (ii) *distributed*, and (iii) *decentralized*.

Edge-based systems with centralized architecture have a single edge device that acts as the master device for the entire system. The master device is usually responsible for monitoring and distributing tasks among other available computation entities in the Edge-Cloud continuum. Specifically, the master device may provide a set of functionalities such as a gateway, monitoring resources, or scheduling algorithms for generating application deployment plans. Based on the resource management taxonomy [TNT18] and surveyed articles, resource management objectives (see Section 2.5.1) are deployed statically on the edge servers [THL16, FAS17, RSNK16, FGRT16] or on the cloud [MCPR14, LNST14, QLW+16]. Generally speaking, edge-based systems designed with central architecture may be feasible in the context of small and non-dynamic edge networks (e.g., smart homes)[YHZ+17]. For instance, Skarlat et al. [SKR+18, SS21] proposes a framework called FogFrame, which aims to deploy and execute various workloads in the fog infrastructure. The scheduler and monitoring components are placed statically in a powerful device at the highest level of the system's hierarchy. Similarly, in [CK18], fog nodes in the network are organized hierarchically. However, such approaches determine a

coordinator statically, which resides in the cloud. Nevertheless, recalling the three design goals for edge-based systems, building edge-based systems with centralized architecture has many limitations, such as (1) systems do not scale easily (i.e., due to the resource-constraints), (2) the master device is statically determined (i.e., dynamicity of the edge network is neglected), and (3) the QoS between the master device and the other edge devices are not considered (i.e., QoS may degrade due to the high-utilization or high end-to-end latency between computing entities).

In sharp contrast to the centralized architecture, the distributed architecture treats all edge devices equally in terms of their system responsibilities. In an edge network without a master device, edge devices in proximity are coordinated and agreed upon to some Service Layer Agreements (SLAs) to execute software components. Essentially, the coordination among devices is achieved by using consensus-based algorithms. In practice, distributed solutions may face latency issues when edge devices need consensus to distribute software components. Due to the latency issues, edge-based systems with distributed architecture consider a small number of edge devices in their network topology (e.g., see [AD19]). Nevertheless, regardless of the approach, both solutions may have plenty of advantages in various IoT scenarios.

In the decentralized architecture, the master device functionalities may be placed *statically* (i.e., at design time) or *dynamically* (i.e., with a control device). In decentralized edge-based systems, system control components are shared among several independent computing entities. For instance, a decentralized approach [ATD19] inspired by the functionality of an auction house has been proposed to enable IoT application deployment at the edge of the network. The proposed solution focuses on application placement but does not address how the system components are decentralized. The second category, dynamic control devices or coordinator devices, are usually mobile devices with less computational power and bandwidth. For instance, fog colonies solution [SNSD17] with a control node enables to select edge coordinators in the system such that they offer some edge functionalities. However, the selection process is dependent on a central node. Nevertheless, recalling the three design goals for edge-based systems, we again advocate that resource management components require a continuous re-evaluation of their placement.

A feasible approach to overcome such a gap is placing functionalities dynamically at the edge via leader-based algorithms [MWV00]. For instance, through initiating an *election* between edge devices, the most suitable node (e.g., in terms of computation power) is elected as a master device. By communicating election results, edge devices in the edge network reach the same conclusion independently. Such a solution denotes a very decentralized approach to elect the master device automatically. Moreover, the elected master device must overcome several challenges, such as delegating various functionalities to other edge devices to handle computation and network overheads. A possible solution to the aforementioned challenges is to introduce new coordinators (i.e., superpeers [JMB06]) in the edge-based system. A coordinator can be responsible for a set of edge devices within the system and provide resource management functionalities

such for instance assisting in the resource discovery process. As the network size expands, new coordinators need to be introduced to the system; respectively, new clusters need to be formed to handle the network and computing complexities introduced with the scalability. Chapter 3 provides our contribution on how to determine system roles within an edge network.

2.5.3 Edge Infrastructure and Communication Types

The architecture of cloud computing and the computational infrastructure is widely researched and well understood. Powerful computers and storage appliances followed by high-performance networking devices are placed within massive data centers. In literature, there is no consistency on what edge infrastructure is when compared to cloud computing. Usually, infrastructures and network topologies are determined based on the use case scenario considered in research papers. For instance, in Mobile Edge Computing (MEC) scenarios, telecom companies may place computation entities at base stations, allowing cloud service providers to place services close to end-users [MB17]. Or, in healthcare system scenarios, several edge devices placed near patients enable processing and monitoring of IoT data streams generated from smart devices [HHI19]. Essentially, edge networks can exist in different sizes and settings; thus, edge-based systems must be configurable by the system designer or self-configurable when the network size changes.

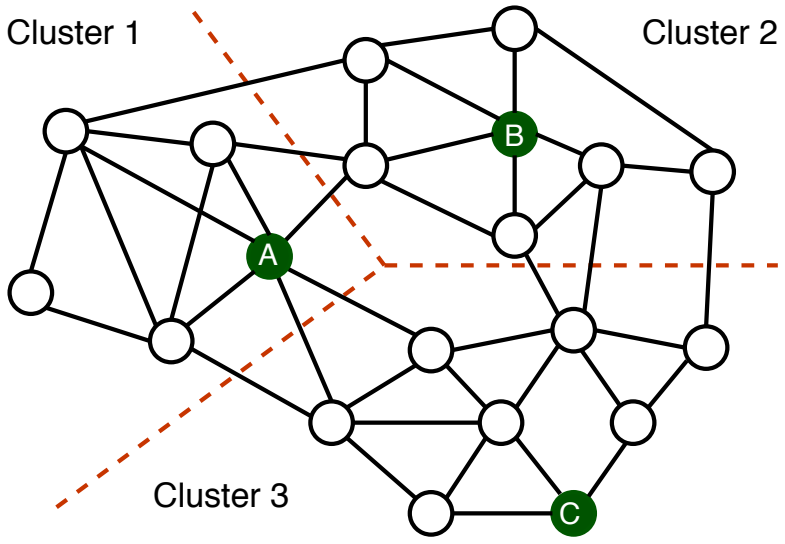


Figure 2.7: An example of an edge neighborhood comprised of twenty nodes organized in three clusters.

A typical edge infrastructure includes heterogeneous, resource-constrained, and geographically distributed computing resources [AMD20]. Current literature in edge computing recognizes and briefly discusses types of communication [MKB18]. Several approaches have been proposed aiming to build and organize computation devices in the edge network

[Kar19, RSB+17, AV15]. Notably, P2P approaches have shown great potential to handle edge infrastructures in a scalable manner [LPSD17]. In [KSL+19] for instance, edge devices are equipped with a mechanism that allows themselves to organize into clusters by considering the proximity aspect. We consider a similar organization type of the edge network proposed in [KSL+19] where nodes are organized in clusters, and the edge network can easily scale. We hypothesize that a group of edge devices in proximity form a cluster, and multiple connected clusters create an edge network or, as we refer to it, an edge neighborhood (see Figure 2.7). We advocate such a network organization since the edge networks can be different in size and setting depending on the use case scenario. Accordingly, grouping edge devices into clusters and assigning a coordinator to each group prevents edge devices from storing information about the entire edge network. In terms of the network responsibilities, coordinators maintain information about edge devices within their cluster and other coordinators in the system. Implementing decentralized systems requires decentralized protocols which are designed to scale by not relying on global knowledge of the system [KTD06]. Chapter 3 briefly discusses our proposed approach to build edge networks and enable automatic resource discovery within the system.

2.5.4 Edge Runtime Platform

The cloud infrastructures usually are comprised of enormous resources which are abstracted by hardware virtualization using Hypervisors. The resources within the cloud are easily managed and accessible as a utility by end-users or other management software [HV19, EPM13]. Unlike cloud computing, edge networks are heterogeneous environments, and executing software components (e.g., system components and edge applications) is another significant challenge. An essential requirement is to enable the execution of software components on a homogeneous runtime platform that follows the "*run once, run anywhere*" model and technology agnostic. Furthermore, the platform must enable easy interaction with other developed system components responsible for accessing, governing, and managing edge applications.

To overcome challenges with resource-constrained edge networks and to fully utilize available resources at the edge, edge applications (i.e., *services*) are divided into a set of independently deployable software components (i.e.,*microservices*). Docker[5] is one of the containerization platforms that allows packaging software components or edge applications and their dependencies in the form of containers. Docker is a platform designed to create, deploy, run, and manage applications using containers. Unlike traditional virtualization, containers are lightweight because they're usually small in size, fast, and easily portable between runtime environments. Additionally, containers compared to virtual machines (VMs) have many benefits such as: (1) containers do not need to include a guest OS in every instance (i.e., containers share the machine OS kernel), (2) containers are platform-independent (i.e., containers contain all their dependencies and they can efficiently run in Edge-Cloud computing continuum), and

[5]Docker, https://www.docker.com/

(3) containers support modern development and application patterns (i.e., microservices, serverless, DevOps).

Another important aspect is the container orchestration tool. Docker Swarm[6] and Kubernetes[7] are the most well known and widely used open-source container-management tools. Both Docker Swarm and Kubernetes support the inclusion of multiple manager nodes responsible for controlling and monitoring all resources in the cluster. Essentially, multiple manager nodes are required to ensure high availability in case the leader manager node fails. Along these lines, both platforms may have plenty of advantages when using them in various IoT scenarios. Generally speaking, Kubernetes, compared to Docker Swarm, provides more advanced features for the management of service deployment to nodes. However, deploying a Kubernetes master on resource-constrained and low-powered edge devices is challenging due to the resource requirements. Even though there are few lightweight versions of Kubernetes dedicated to edge computing (e.g., KubeEdge[8]), running master functionalities at the edge remains challenging due to the low processing capabilities in low-powered edge devices. Essentially, running these functionalities on edge devices like Raspberry Pi 3 consumes high processing power, while starting services may take longer. On the other hand, Docker Swarm is the native orchestration tool for containers, and resource consumption remains within acceptable limits for low-powered edge devices. Thus, considering the edge network's dynamic nature and resource-constrained devices, we advocate the Docker Swarm orchestration tool is much more suitable for edge-based systems. Nevertheless, we do not claim that running (i.e., executing) Kubernetes master or KubeEdge at the edge is unsuitable. As we previously mentioned, edge computing means different things to different people; therefore, edge devices at the edge can be more powerful processing capabilities than devices considered within this thesis.

2.6 Summary

Edge computing is considered a fundamental enabler for the proliferation of IoT applications. Compute and storage resources placed at the edge of the network are used to bridge the gap between the cloud and IoT domains. Such computation entities perceived as *edge devices* can be used to analyze high-volume IoT data streams and, at the same time, provide control facilities to participating IoT devices. A wide range of available resources at the edge provides a seamless opportunity to deploy various edge applications providing their services with low latency. However, different edge functionalities are needed at the edge (e.g., elasticity, resource coordination, etc.).

We outlined the role and the importance of decentralizing the edge-based systems running on the edge. We discussed some of the research issues, challenges, and potential solutions for enabling resource management in a decentralized manner, controlling and managing

[6]Docker Swarm, https://docs.docker.com/engine/swarm/
[7]Kubernetes, https://www.kubernetes.io/
[8]KubeEdge, https://www.kubeedge.io/

edge applications and their dependencies. Specifically, we propose a self-adaptive and decentralized configurator to allow easy configuration, deployment, and operation of such applications on top of heterogeneous edge infrastructure. To achieve the mentioned system objectives, in the remainder of the thesis, we investigate and evaluate three systems aspects (1) enabling resource discovery in a decentralized manner and a configurator that self-adapts to the dynamic and uncertain edge network, (2) enabling developers to specify elastic requirements for edge applications followed by a technical framework to control the elasticity at runtime, and (3) enabling resource coordination at runtime. Nevertheless, to fully operationalize the introduced edge framework, we must further investigate each module and develop additional functionalities.

CHAPTER 3

Resource Discovery using Metadata Replication in Edge Networks

Recent advancements in distributed systems have enabled deploying low-latency edge applications (i.e., IoT applications) in proximity to end-users, respectively, in edge networks. The stringent requirements combined with heterogeneous, resource-constrained, and dynamic edge networks make the deployment process challenging. Besides that, the lack of resource discovery features make it particularly difficult to fully exploit available resources (i.e., computational, storage, and IoT resources) provided by low-powered edge devices. To that end, this chapter proposes a decentralized resource discovery mechanism that enables discovering resources in an automatic manner in edge networks. Through replicating resource descriptions (i.e., metadata), edge devices exchange information about available resources within their scope in a peer-to-peer manner. We propose a solution for building edge networks as a flat model and organizing edge devices in clusters to handle resource discovery complexity. Our approach supports the system in coping with the dynamicity and uncertainty of edge networks. We discuss the architecture, processes of the approach, and the experiments we conducted on a testbed to validate its feasibility on resource-constrained edge networks.

The rest of the chapter is structured as follows. The related work is presented in Section 3.2. Section 3.3 presents our approach to organize edge devices in edge neighborhoods and resource modeling. Section 3.4 describes in detail the proposed algorithm in charge of determining system coordinators and the framework for automatic resource discovery in edge neighborhoods. Section 3.5 provides the experiment results to evaluate the proposed solution followed by discussion in Section 3.5.4. Finally, Section 3.6 summarizes the chapter.

3.1 Introduction

In recent years, one prominent approach that has emerged to overcome latency delays, especially in pervasive applications, suggests utilizing computation entities in proximity to end-users. Edge computing is a distributed computing paradigm that places resources (e.g., compute and storage) at the edge of the network [SD16]. Edge resources (i.e., perceived as edge devices) are typically resource-constrained and heterogeneous devices that can process, store, and analyze data and deliver efficient and low-latency user-facing services. Such edge devices can support contemporary applications by (1) running decision functions close to data-producing end-users, (2) processing various computational workloads, and (3) minimizing the need to transmit data to the cloud. A wide range of available resources at the edge has introduced new opportunities such as deploying low-latency, privacy-awareness, and resilient *edge applications* (e.g., IoT applications). In this regard, many studies have been carried out to exploit edge networks for various purposes (i.e., from processing sensory data streams to EdgeAI applications) [DM20]. Notably, we consider edge networks as resource-constrained, heterogeneous, and dynamic environments where multiple low-powered edge devices in proximity are connected. In this sense, we may have various edge networks (e.g., smart building, smart home, drone network, etc.) where end-users may deploy different edge applications.

Computer scientists have been mostly focused on proposing multiple techniques for resource allocation problems to minimize latency and maximize resource utilization at the edge. Notably, today's applications are not monolithic; they are divided into multiple independent deployable tasks. Each task may have various resource requirements that need to be fulfilled by available edge devices upon deployment. Tasks are characterized by requirements such as computational (i.e., processing, memory, storage), energy, or bandwidth. However, resource allocation approaches often overlook the dependence between tasks and IoT resources (e.g., sensors and actuators)[AMD20]. Additionally, despite the numerous advantages introduced by edge networks, communication between edge nodes[1] and the network organization has been neglected by many research papers [AD18, TMD19]. According to the paper [Kar19], the communication and network organization type of a platform affects the functionality of the final applications deployed at the edge infrastructure.

In literature, very few research works consider IoT resources as an application requirement that needs to be fulfilled when deploying them at the edge [BF17, ATD19]. For example, computing a local weather forecast in a smart agriculture setting may require various IoT resources such as temperature and humidity readings from available sensors across a crop field [TMD19]. In such a scenario, application tasks dependent on particular IoT resources must be deployed on edge devices providing those resources. Notably, edge devices are not equipped with the same capabilities, and such a stringent constraint reduces the number of eligible deployments at the edge. For example, the allocation technique [BF17] tries to overcome the problem by enabling sharing of IoT resource information within

[1] In this thesis, we use terms edge nodes, nodes, and edge devices interchangeably.

neighbor nodes. Similarly, the proposed solution [ATD19] acknowledges the problem; however, it faces latency issues, and it considers a limited number of edge devices in the network topology. Nevertheless, both approaches do not address issues related to the communication between edge devices, network organization, and resource discovery.

To overcome these shortcomings, we discuss two major issues. First, edge networks should be designated to handle the complexity of discovering resources in a decentralized and automatic manner. Thus, we design edge networks in a flat model where edge devices in certain proximity are connected in a peer-to-peer (P2P) way. A set of edge devices form a *cluster*; while multiple connected clusters form an edge network, respectively *an edge neighborhood*. Besides that, we introduce system coordinators with their corresponding functionalities to organize edge devices and support the resource discovery process in an edge neighborhood. Second, resource management's fundamental objective is to discover resources available at the edge [TNT18]. Edge devices provide heterogeneous resources and are equipped with many IoT resources. We refer to heterogeneous resources as computational, sensing, context data, or other domain-specific resources. Naturally, performing a resource discovery algorithm for each resource on the entire network is possible. However, such a process is computationally intensive, and resources are discovered by querying the entire network based on a keyword [PP12]. Thus, we advocate that exchanging information about available resources between edge devices in an automatic manner enables: (1) sharing of resources across the whole system and (2) edge devices to perform complex queries locally.

This chapter proposes a framework and a methodology to build edge networks as a flat model and enable edge devices to be organized in clusters. Our proposed framework enables discovering heterogeneous resources and making them available at the runtime. A resource is described by providing certain core information about the functionality and its properties. This type of description, known as the resource's metadata, is replicated among edge devices and stored in a decentralized manner. Furthermore, we treat privacy aspects based on each edge device's resource preferences, ensuring that not all resources are advertised across the whole system. Specifically, our concrete contributions are as follows:

- We develop a prototype enabling edge devices to be connected in a P2P manner and form an edge network. Our approach is built on top of the Kademlia Protocol [MM02] used as the communication protocol between edges devices. To enable scaling of our approach, we propose a solution to break down edge networks into clusters as well as introduce new coordinators to handle the complexity to discover resources automatically.
- We advocate decentralization as the system can operate without a static entity for discovering available resources. Essentially, coordinators are placed dynamically and run on the most suitable edge devices providing various services. Our approach supports the system in coping with the environment's dynamicity and uncertainty and continuously re-evaluates coordinators' placements.

- To validate the approach's feasibility, we show that the prototype's footprint is limited to hardware resources and network bandwidth. We evaluate our prototype on a testbed composed of low-powered ARM-based edge devices.

3.2 Related Work

We divide related work into two categories. First, we review P2P approaches and communication types used at the edge. Afterwards, we review related techniques on resource management as they apply to IoT.

3.2.1 Communication in Edge Networks

Many studies have been carried out, leading to different approaches aiming at exploiting the edge and providing various allocation techniques to fulfill application requirements. In this context, many platforms use different communication types to achieve particular system goals. The current literature recognizes and briefly discusses communication types at the edge [MKB18, BMZA12]. According to the paper[Kar19], three main types of communications in edge computing are identified: hierarchical, P2P, and hybrid.

Notably, P2P approaches have shown great potential to handle edge infrastructures in a scalable manner [LPSD17]. Therefore, a lot of research has been conducted in this context, resulting in many approaches that aim at organizing edge devices using different communication types [RSB+17]. Due to their fully distributed nature, P2P architectures are both scalable, reliable [IM09], and fault-tolerant [SWVDT18]. Many edge-based platforms use P2P to organize edge devices within the edge network [TBT18]. Similarly, Cabrera et al. [F+16] propose a P2P-based fog platform that enables storing data generated from IoT devices. In the proposed approach, data is stored among fog devices in a distributed manner. A fog device can restore corrupted data by asking other fog devices in the network. In the mentioned works, devices are organized in a P2P manner and are assumed to have a partial view [TBT17] or full view of the network (i.e., $\mathcal{O}(1)$ protocols). In contrast to the mentioned works, our approach is build on top of the Kademlia Protocol and provides a decentralized mechanism to organize edge devices in clusters. Our solution through the proposed algorithms determines the most suitable edge devices to place coordinators in an edge network and adapts to the changes that may occur. Nevertheless, the proposed solution makes edge neighborhood autonomous environments less dependent on centralized nodes.

3.2.2 Resource Discovery

Performing a resource discovery algorithm for each resource on the entire edge neighborhood is computationally demanding. For instance, queries like discovering all cameras in a particular area are becoming highly desirable for IoT applications. In general, such system behavior in decentralized DHT protocols is hardly achievable. Even though some DHT-based approaches support discovering data through multi-attribute queries

[ZCX+13]; however, such methods remain unsuitable in IoT systems and edge networks due to the high content lookup latency. In addition to that, resource discovery is a critical challenge for IoT application performance.

Service-based discovery has been widely studied [KL14, WC16]. Paganelli et al. [PP12] introduce a DHT-based IoT service discovery that supports multi-attribute and range queries. Furthermore, the proposed solution enables real-time monitoring of resource positions since it updates resource locations periodically. Santos et al. [SWVDT18] propose a resource discovery service for resource provisioning in fog environments. The proposed solution is based on DHTs and enables exchanging information about the available resources (i.e., performance metrics, workloads, etc.). However, in contrast to our approach, the proposed solution does not address privacy aspects, does not consider discovering IoT resources, and discusses no actual provisioning mechanisms.

Resource allocation and management have been widely studied both in the cloud and fog computing [MKB18]. Up to now, many factors have been considered including time (e.g., computation [ZGG+16]), data size [SCD15], cost (e.g., networking and deployment[GZG+15], execution [HXWC15]), user-application context [HXWC15], etc., which have been found to play important roles in resource and service provisioning. Jain et al. [JT17] propose a solution where the IoT application is divided into multiple tasks annotated with location information. The application is decomposed into fragments and deployed to the corresponding individual compute nodes based on the annotation. In contrast to the mentioned research papers, the resource discovery aspect has been mostly ignored. Furthermore, none of the papers have addressed privacy aspects when considering resource discovery.

Our approach's second novelty and contribution to state-of-the-art lie in the introduced novel decentralized mechanism that enables automatic resource discovery in edge networks. Discovering resources at once represents a feasible and optimal solution for edge neighborhoods. We enable end-users or edge applications to perform various complex queries locally by replicating metadata between edge devices. Contrary to other works, our resource discovery mechanism considers resource privacy preferences ensuring that not all resources are advertised across the whole system.

3.3 Edge Networks and Resource Modeling

This section introduces our approach to organize edge devices in an edge neighborhood. We describe basic definitions and then discuss communication protocol between edge devices. Subsequently, we discuss architecture modeling and resource modeling in Section 3.3.4.

3.3.1 Definitions

We refer to an *edge neighborhood* as a resource-constrained edge network, which is comprised of edge devices placed close to each other (see Figure 3.1). Edge neighborhoods

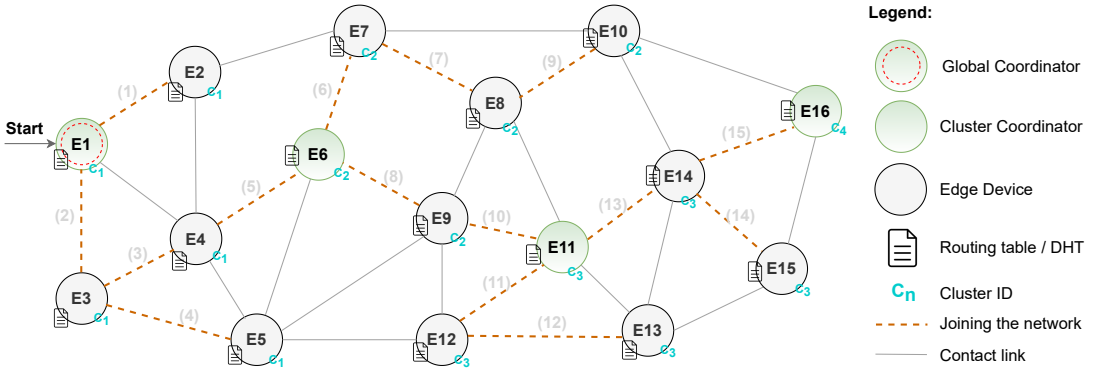


Figure 3.1: An example of an edge neighborhood consisting of sixteen edge devices organized in four clusters.

are formed in various geographical areas and within different contexts (i.e., smart homes, drones network, etc.). Notably, they may vary in size and settings; thus, our proposed system is configurable by the system designer through the configuration file. In our conception, edge devices are low-powered, heterogeneous, and resource-constrained computational entities in the system. Within the system context, edge devices may provide multiple functionalities (e.g., act as a client device, server device, and bootstrap device). Furthermore, edge devices are grouped in clusters to promote scaling, reduce bandwidth consumption, and manage the difficulty of discovering resources in an automatic manner in edge neighborhoods. Subsequently, edge devices within the same cluster are considered *neighbor devices*, as well as clusters close to each other are considered as *neighbor clusters*. Each cluster consists of a finite number of edge devices, and each device belongs to one and only one cluster at a time.

In the system context, each edge device may serve specific roles such as *i) cluster coordinator* and *ii) global coordinator (i.e., configurator)*. Such roles are dynamically assigned to the edge devices based on their resource capabilities (discussed in Section 3.4.1). In Figure 3.1, each cluster has a coordinator device (i.e., with a green circle) and a single global coordinator device (i.e., with red and green circles). We assume that cluster coordinators act as *superpeers* [JMB06]. Each cluster coordinator keeps track of the other coordinators and devices within the same cluster (i.e., IP addresses, port). Similarly, edge devices in the same cluster store information for one another and are always aware of their cluster coordinator and global coordinator. A cluster coordinator may have various responsibilities for a subset of devices (i.e., monitoring, discovering resources, etc.). The global coordinator is responsible for monitoring coordinators, exchanging resource descriptions between clusters, and managing edge applications in Edge-Cloud infrastructure (i.e., controlling elasticity, migrating tasks, etc.). However, future work remains to provide a complete solution for the coordinators introduced in this chapter. This thesis focuses on three main aspects: i) organizing edge devices in an edge neighborhood, ii) determining the most suitable devices to place the global and

cluster coordinators, and iii) enabling automatic resource discovery in heterogeneous and dynamic edge neighborhoods.

Each edge device may serve as the entrance door into the edge neighborhood. Essentially, edge devices provide core functionalities to assign newly added edge devices in the available clusters or create new clusters in the edge neighborhood. We introduce three functionalities to identify the maximum number of edge devices in a cluster. First, the system designer may define a system-wide parameter to bound the maximum size of clusters. Second, the system designer may configure cluster coordinators to generate random cluster size (i.e., not higher than the system-wide parameter). And third, the system designer may define a system-wide threshold value specifying the maximum CPU utilization for cluster coordinators. As a result, depending on the functionality enabled, we may have edge neighborhoods with different cluster sizes. The cluster number in an edge neighborhood is not bounded.

3.3.2 Communication Protocol

The communication between edge nodes in the proposed architecture is realized through implementing the Kademlia Protocol [MM02]. Our study revealed several distributed hash table solutions to consider for our framework such as Chord [SMK+01], Pastry [RD01], and Tapestry [ZKJ+01]. We suggest using Kademlia as the edge communication protocol since it spreads contact information automatically and uses minimal Remote Procedure Calls. Moreover, one of the most important features of the Kademlia network is the $\mathcal{O}(log(n))$ lookup time. Finding nodes and resource descriptions in this type of network is fast and efficient.

Kademlia is a distributed hash table (DHT) for decentralized peer-to-peer computer networks. A Kademlia network consists of nodes where each node has a unique 160-bit ID as an identifier. Nodes in the Kademlia network communicate using User Datagram Protocol. Moreover, the participating nodes exchange their information through node lookups. An overlay network is formed where each node is identified by its node ID. The provided ID will serve as a direct map to the file hashes and where to obtain the requested resource. Besides the unique ID, it maintains a routing table and a DHT. A routing table maintains a list for each bit of the node ID (e.g., node ID consists of 160 bits means that it will keep 160 such lists). A routing table is divided into created lists known as buckets - each bucket contains contact information and the distance from the current node. Contact information resides in one of the buckets in which it contains the node ID, IP address, and port number of the other node. Buckets in the routing table are updated when a new node joins the network. In addition, new edge nodes can be bootstrapped by knowing basic contact information of any other reachable DHT nodes in the network. The DHT segment stores key/value pairs where the key is the name of the public metadata document and the value is the network location of the document

Furthermore, DHT is a data storage that is kept consistent between all edge devices within the whole edge network. Essentially, when an edge device updates its local

DHT, the changes are propagated to all other devices, allowing them to be queried and manipulated again. Likewise, information about current cluster coordinators and the global coordinator is stored in DHT. Each edge device implements a distributed routing table and stores data in Kademlia buckets ordered by the local device's distance. The routing table size is bounded by $\mathcal{O}(log_2(l/k))$ where l is the number of edge devices in the edge neighborhood, and k is the bucket size. Once a bucket is full, it replaces unresponsive devices in favor of incoming devices. The routing table size in our proposed approach is configurable through the configuration file, and it depends on the expected edge neighborhood size. We consider edge devices as resource-constrained computers (i.e., in terms of computational and storage capabilities); thus, edge neighborhoods' size is not expected to be massive.

3.3.3 Forming an Edge Neighborhood

Figure 3.1 shows an illustration of how our solution organizes edge devices in an edge neighborhood. Initially, the edge neighborhood is formed with an edge device `E1`. Since it is the only device in the edge neighborhood, it is automatically assigned to cluster `C1` and determined as the cluster coordinator $\texttt{C1}_{\texttt{coord}}$ and the global coordinator $\texttt{G}_{\texttt{coord}}$. At the same time, `E1` serves as an entrance door into the edge neighborhood. We assume edge devices progressively join the edge neighborhood and each edge device also becomes another bootstrap device $\texttt{En}_{\texttt{b}}$ (i.e., `n` is a unique random ID). It is common to keep a list of always-running edge devices to allow new edge devices to join an edge neighborhood. In our case, `E1` is the first device contacted by `E2` to join the edge neighborhood. The edge device `E2` is assigned to cluster `C1` by `E1` in coordination with $\texttt{C1}_{\texttt{coord}}$. Furthermore, `E2` is supplied with DHTs and the complete routing table from `E1`. The process of adding new edge devices in an edge neighborhood is presented in Algorithm 3.1.

The process continues with adding a new edge device $\texttt{E}_{\texttt{new}}$ by contacting $\texttt{En}_{\texttt{b}}$. Once $\texttt{E}_{\texttt{new}}$ request to join the neighborhood, bootstrap device $\texttt{En}_{\texttt{b}}$ initially provides a unique random ID (i.e., 160-bit). Afterward, $\texttt{En}_{\texttt{b}}$ retrieves information from its cluster coordinator $\texttt{Cn}_{\texttt{coord}}$ where to assign $\texttt{E}_{\texttt{new}}$. A cluster coordinator $\texttt{Cn}_{\texttt{coord}}$ (i.e., referred as `this`) maintain information regarding: i) own cluster maximum size $\texttt{C}_{\texttt{M}}$ (line 1), the current cluster size $\texttt{C}_{\texttt{S}}$ (line 2), and other available clusters $\texttt{C}_{\texttt{N}}$ (line 3). $\texttt{C}_{\texttt{N}}$ provides clusters with available places where $\texttt{E}_{\texttt{new}}$ can be assigned.

The `MaxClusterSize()` function is configurable and enables the system designer to define the maximum number of edge devices per cluster. In the configuration file, such value can be determined by specifying (i) the system-wide parameter (e.g., five edge devices per cluster or random value) and (ii) the CPU utilization threshold (e.g., CPU utilization set to 35%). When the system-wide parameter is set, and random cluster size is disabled, edge devices are grouped into clusters of the same size (as illustrated in Figure 3.1). When a cluster exceeds the maximum allowed devices, more clusters with corresponding coordinators should be designated to handle the resource discovering complexity. Edge devices are grouped into clusters of different sizes when the CPU utilization threshold is set. More specifically, when $\texttt{C}_{\texttt{M}}$ and $\texttt{C}_{\texttt{S}}$ are equal, $\texttt{E}_{\texttt{new}}$ is assigned

Algorithm 3.1: Process of adding a new edge device

Input : E_{new}
Output : Adding E_{new} to the edge neighborhood

```
1  C_M = this.MaxClusterSize()
2  C_S = this.CurrentClusterSize()
3  C_N = this.OtherClusters()
4  boolean flag = false
5  if C_M < C_S then
6     En_b.addDevice(E_new,this.cluster())
7  else
8     if C_N != null then
9        C_clo = this.findMostClosest(C_N)
10       foreach c ∈ C_clo do
11          if c.size() < C_N.cluster(c).maxSize() then
12             En_b.addDevice(E_new,c.cluster())
13             flag = true
14             break
15          end
16       end
17    end
18 end
19 if flag = false then
20    En_b.addDevice(E_new,new Cluster())
21 end
22 this.updateRoutingTable()
23 this.updateDHT()
```

to one of the existing clusters, or a new cluster is created. Nevertheless, both options can be used at the same time. However, the violated option decides whether or not the cluster can scale further. Furthermore, if the condition in (line 5) is not violated, E_{new} is assigned to the current cluster of the En_b (line 6).

Neighbor clusters are found through using a system call (i.e., *traceroute command*), which estimates proximity with other cluster coordinators (i.e., using hop count and latency). The function `findMostClosest(`C_N`)` uses traceroute command and returns the most suitable clusters that E_{new} can be assigned (line 9). This is especially useful in edge neighborhoods running on different networks. Finally, Cn_{coord} provides information to En_b to assign the E_{new} to the particular cluster if and only if condition in (line 11) is not violated (lines 11-15). Otherwise, when there are no clusters with free places, then E_{new} is assigned to a newly created cluster by Cn_{coord} (line 20). Note that each cluster in the edge neighborhood has a unique ID generated by En_b when the cluster is created. Notably, the bootstrap device's task is to cooperate with the cluster coordinator to assign

a cluster ID to the newly joined devices. Finally, once $\texttt{E}_{\texttt{new}}$ joins the edge neighborhood, $\texttt{Cn}_{\texttt{coord}}$ updates the routing table and other DHTs (lines 22-23).

3.3.4 Resource Modelling

We assume that an edge device may contain multiple resources (i.e., computational, sensing, or context data) represented as microservices [BSH+17]. When invoked remotely, such microservices yield resources; however, resource information must be shared beforehand among edge devices in the edge neighborhood. An essential step towards discovering these resources in the edge network is resource modeling at design time. To ensure automatic resource discovery in an edge setting, two types of resources must be modeled: *i) IoT resources* (i.e., sensors, actuators, etc.) and *ii) edge devices*. We propose a resource representation structure that provides seven main properties as described below:

- Resource identification provides essential information on the resource, such as resource name, a unique resource identification ID, and the edge node ID, which handles the resource description. Resources are described by several keywords, which helps developers to search for resources within the edge node. For example, knowing the location of the resources is a way for the user to locate resources. For instance, the location parameter may be used to discover resources such as temperature sensors in a certain area in smart cities. It also may provide a product description (e.g., device brand, category, description, etc.).

- Resource connectivity provides information such as the IP address and port number to reach the resource through the overlay network. In addition, it provides a piece of information related to the connectivity status (e.g., connected or disconnected). For example, a user application might request a specific available resource from the edge node, which in turn gets the information on how to reach the desired resource.

- Resource capability describes the ability of a resource to perform some action (e.g., a resource may provide storage capabilities to others). In addition, specific resources may provide sensing operations to change the state of the resource (e.g., increasing/decreasing temperature).

- Resource accessibility provides information related to the resource access policy. We define two types of resources that participate in the edge network: i) public resources and ii) private resources. All edge nodes that join the network can discover and replicate public resources. Private resources are not discoverable by other edge nodes - such resources are accessible only locally.

- Resource output provides information related to the output data of the resource (e.g., unit). For example, a specific user application might require a temperature sensor that provides values only in Fahrenheit. However, we note that metadata does not contain any real-time value of the resource. Resource location describes

the properties of the resource where it is located. It can be described through a given geographical position or a logical location. A resource can only have a link to one location instance.

- The resource administration domain defines the resource's owner shared in the edge network. For example, the fire department can monitor the activity within the administrative domain to ensure that the monitored resources remain consistent.

Representation of resources is mainly focused on representation models using ontologies. We propose that a resource description needs to be formatted in a JavaScript Object Notation (JSON). The rationale for exchanging metadata over JSON is:

- Metadata using JSON may have a very rich description of the resource while being lightweight and machine readable.
- Metadata considered within a JSON file is very small; thus, the replication process across many edge devices organized in clusters is feasible and does not degrade the overall network performance. Hence, JSON documents considerably reduce the network traffic between edge nodes compare to the other formats (e.g., XML [SM12]).

Our resource structure is similar to the ontology-based structure proposed by Barnaghi et al. [DBBM11]. Unlike the ontology-based approach, we format resource descriptions in a JSON file. We advocate that exchanging metadata over JSON is a lightweight process, machine-readable, and provides rich information about resources. We assume that resource descriptions are constructed by the system designer or the manufacturer itself.

3.4 A Decentralized Edge-to-Edge Resource Discovery

In this section, we discuss the process of determining the global coordinator and cluster coordinators. Next, we explain the framework for sharing edge device resources based on privacy preferences. Finally, we discuss edge device failures, dynamicity, and uncertainty of edge neighborhoods.

3.4.1 The Process of Determining Coordinators

The process of determining coordinators in a distributed system should be carried out in the system background, encapsulated from the end user's perspective but indispensable for the correct and efficient execution of distributed tasks. System coordinators' role is versatile and can range between managing applications, monitoring resources, or distributing data between devices. We define two leading roles, such as *i) cluster coordinator* and *ii) global coordinator (i.e., configurator).* Such roles are dynamically assigned to edge devices based on their resource capabilities. Determining new coordinators is triggered by

an event when the global or a cluster coordinator experiences high utilization of specific hardware resources (i.e., CPU, memory, or storage) and requests to transfer leadership to other devices. Determining a new cluster coordinator occurs only between edge devices within the same cluster. The process of determining a new global coordinator occurs between cluster coordinators. The latter consists of two phases: first, cluster coordinators are determined; second, new cluster coordinators determine the global coordinator. In Algorithm 3.2, we present the process of determining system coordinators.

Algorithm 3.2: Process of determining coordinators

Input : ϕ_i , θ, σ_i
Output : C_{coord} , G_{coord}

```
1  t = 0
2  ν_this = GetDeviceMetrics(φ_i)
3  round = Random()
4  Initial_Message(round,σ_i)
5  Parameter_Message(ν_this, round,σ_i,γ)
6  I ← Receive_Parameter_Messages()
7  Solution_Found ← False
8  while t < θ or !Solution_Found do
9     ν = I.getBest()
10    if ν < ν_this and ν.round = round then
11       Solution_Found ← True
12       Set(E_this, EdgeMode)
13       Set(En, Coordinator, σ_i)
14 end
15 if !Solution_Found then
16    Set(E_this,Coordinator, σ_i)
17 end
18 E_this.updateDHT()
```

The proposed algorithm runs on each edge device separately. The algorithm takes three inputs: i) an edge device hardware metrics denoted with ϕ_i, ii) the deadline to find a solution denoted with θ, and iii) the process type denoted with σ_i. The process of determining coordinators is designed by considering hardware metrics and the bandwidth of edge devices. We consider hardware metrics both statically (e.g., CPU cores, storage capacity, etc.) and dynamically (e.g., current CPU load, current storage, etc.). Such hardware information can be monitored using *Hyperic Sigar*[sig] while *Assolo* [GRT09] enables collecting bandwidth probes. Notably, the algorithm gets only the current device metrics ν_{this} specified at design time (line 2). In our case, we consider metrics θ_i related to the CPU utilization degree. However, such a parameter is configurable based on system requirements. The deadline θ is given at design time (e.g., $\theta = 50$ms). The third input σ_i specify the process type: i) determining the global coordinator (σ = `global`) and ii) determining a cluster coordinator (σ = `cluster`).

A unique random signature (i.e., SHA-1) called `round` is used to make each process unique (line 3). In lines (4-5), we define two types of messages exchanged between devices: *i) initial message* and *ii) parameter message*. First, the initiating coordinator (i.e., process initiator device) probes which edge devices are up and running in the edge neighborhood. Then, it sends an initial message only to the edge devices that responded on time. The initial message essentially contains a list of all participating devices (i.e., only those edge devices that replied) and a unique `round` value assigned to a process round. Second, once receiving an initial message and the list of participants, the process initiator device sends a parameter message containing its local metrics to all participating devices. The parameter message contains local hardware metrics, utilization values in percentage, a process round value, and a timestamp when the process to determine the new coordinator is started γ. The timestamp γ ensures that the metrics are not older than the process initiation time. Each edge device responds to the initial message and sends its parameter message to other edge devices, including the initiating coordinator (line 6). Notice that the same proposed algorithm determines the global and cluster coordinators.

After exchanging performance metrics, edge devices develop the same result independently. Each device compares received parameter metrics and determines which device is most suitable for a cluster/global coordinator (lines 8-14). To find the most suitable edge device, we compare the number of CPU cores, clock speed, and current utilization (line 9). $\mathtt{E}_{\mathtt{this}}$ changes the status to the `EdgeMode` (i.e., edge device only shares resources) only when an edge device `En` with better performance metrics is found. Line 16 is executed only when a solution is not found within the time θ, and the coordinator role remains in the current edge device $\mathtt{E}_{\mathtt{this}}$ since no better edge device is found. Furthermore, each edge device maintains a complete overview of all processes and saves the information in a DHT. The result is propagated to all edge devices within the edge neighborhood using the DHT. The last step is executed by all edge devices, which on the one side creates some redundancy but also improves the stability of propagating results (line 18).

3.4.2 System Architecture and Resource Discovery

We propose a framework for enabling automatic discovery of heterogeneous resources in edge networks [MATD19]. Figure 3.2 illustrates three main components of the framework: Edge-to-Edge Communication, the Metadata Container, and the Local Search Engine facility. The Edge-to-Edge Communication component implements communication between edge devices (i.e., Kademlia Protocol), organizes edge devices in clusters, determines coordinators, and exchanges resource descriptions in an edge neighborhood (1). The Metadata Container is responsible for analyzing resource descriptions based on their privacy preference defined by the system designer at design time. Besides that, the component is responsible for sharing resource descriptions system-wide (2), processing received resource descriptions from other edge devices (3), and storing resource descriptions locally (4). The Local Search Engine component (4) is based on CouchDB document-oriented NoSQL database [Cou]. This component subsequently stores data

locally and, through APIs, enables users to query stored content (6).

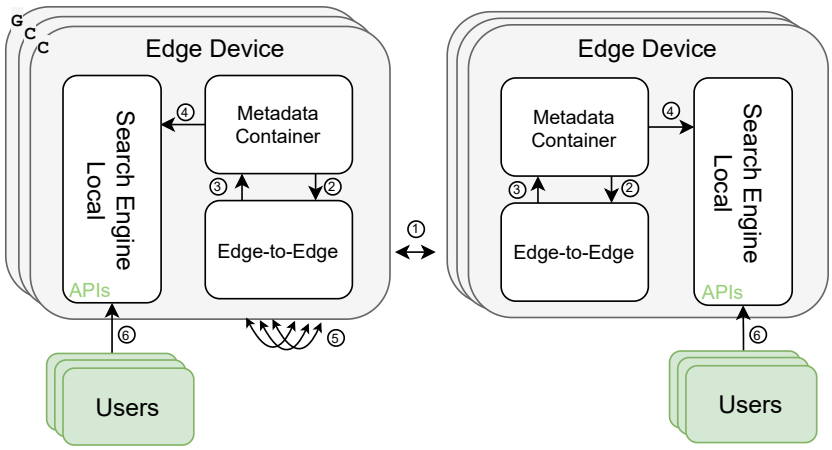


Figure 3.2: Resource discovery through metadata replication: High-level architecture.

Referring to Figure 3.2, our resource discovery mechanism allows edge devices within the same cluster to exchange metadata through their cluster coordinator. Once coordinators are determined, edge devices are ordered to share their public metadata document with their cluster coordinator (1). Other edge devices contact their cluster coordinator to retrieve all public metadata documents contained within their cluster. This may happen once a new edge device is connected to the edge neighborhood or when an edge device wants to refresh its current storage. The frequency to refresh metadata documents is also configurable at design time. Besides that, the global coordinator regularly exchanges metadata documents with the cluster coordinators (5).

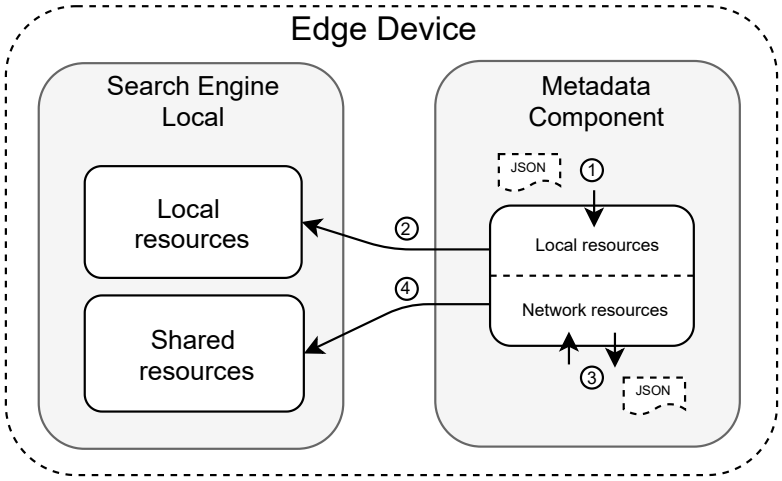


Figure 3.3: Processing metadata on the edge.

In Figure 3.3, we present the process of analyzing resource descriptions and the process

of sharing them in an edge neighborhood. As mentioned before, we consider edge devices as resource-constrained devices with a set of built-in sensors and actuators. Each resource is described by providing certain core information about the functionality and properties (see Section 3.3.4). We assume that edge device manufacturers provide JSON format resource descriptions (i.e., metadata). The Metadata Container is responsible for analyzing metadata in the edge device. The component stores resource descriptions based on privacy preferences i) public resources (i.e., shared resources (4)) and ii) private resources (i.e., local resources (2)). Public resource descriptions are merged into a single metadata document and named with the edge device ID. The public metadata document is shared with the corresponding cluster coordinator system-wide (3). The local search engine component separates resource descriptions into private ones and those shared system-wide.

3.4.3 Edge Device Failures

Consider a situation when a cluster coordinator triggers a new process to determine the new coordinator in the cluster. An edge device under a high workload may fail to determine cluster coordinators or the global coordinator. Even though edge devices do not participate in a determination process, they continuously update their local DHTs with the closest devices. Besides, those who have failed and re-joined the edge neighborhood may contain some obsolete information (i.e., DHT is not up-to-date, coordinator information is outdated). However, after joining the edge neighborhood, edge devices must update their local DHTs with the closest devices.

Another situation may arise when a specific cluster coordinator fails. The global coordinator is responsible for detecting such failure and triggering a new process to determine the coordinator. As discussed previously, edge devices within the cluster decide on their new coordinator. One of the cluster coordinators triggers a new global coordinator determination process when the global coordinator fails. Notably, each cluster coordinator verifies whether the global coordinator has failed and responds to the determination process. Furthermore, the resource discovery process is not repeated when edge devices frequently leave and join the edge neighborhood (i.e., due to connectivity issues). Moreover, when an edge device goes offline, the metadata documents remain stored in the other devices for some time. This is especially useful in unstable edge neighborhoods (e.g., wireless), prone to momentary loss of connection.

3.5 Evaluation

In this section, we first discuss our evaluation setup environment, prototype details, and limitations. Next, we evaluate the approach's effectiveness by running multiple experiments and checking the proposed solution's behavior in different situations. We assess our proposed solution regarding hardware and bandwidth consumption during runtime. Then, through a use case, we show how the proposed discovery mechanism

increases the number of eligible deployments when deploying edge applications in an edge neighborhood. Finally, we conclude with a discussion in Section 3.5.4.

3.5.1 Setup, Prototype Details, and Limitations

We developed a prototype that implements the Kademlia Protocol and core functionalities to enable the automatic discovery of heterogeneous resources in edge networks to show the feasibility of the proposed approach. The prototype is written in Java and tested on a testbed composed of (a) edge devices (i.e., Raspberry Pi 3 Model B V1.2) with 4×ARM Cortex-A53 CPU at 1.2 GHz, 1 GB of RAM, and 16 GB disk storage, and (b) virtual edge device instances. To evaluate our prototype, we exploited the testbed (i.e., edge neighborhood) composed of 60 edge devices placed close to each other. Edge devices contain multiple resource descriptions generated randomly at design time. Furthermore, edge devices in the testbed are connected through a wireless connection with a nominal speed of 10 Mbps and 5 Mbps in download and upload. We assume that edge devices trust all other devices in the edge network since they belong to the same local administrative domain.

In distributed systems, discovering and adding devices automatically in an edge network is a significant challenge. In our current implementation, edge devices have an open designation port that listens for possible future connections. To join the edge neighborhood, edge devices require to know at least one device (i.e., IP, port) currently up and running. We acknowledge that the current implementation represents a limitation, and further investigations are required to develop an advanced approach to discover running edge devices. We acknowledge that IoT resources (e.g., sensors) can also be connected to edge devices using various end-to-end communication protocols (e.g., Zigbee, etc.). However, in this thesis, we treat communication and operational aspects of IoT resources as orthogonal to our approach; we assume edge devices are equipped with a set of IoT resources. Furthermore, we acknowledge that using the Java Virtual Machine (JVM) environment is resource-expensive. However, this thesis shows the approach's feasibility in resource-constrained edge neighborhoods.

3.5.2 Experiments and Results

We evaluate our prototype on a testbed, whose size progressively increases to 60 edge devices as presented in Table 3.1. The cluster coordinators can determine their cluster sizes based on (i) the CPU utilization threshold (i.e., configured to 35%), (ii) the system-wide parameter (i.e., configured to 30 devices per cluster), and (iii) the random value (i.e., not bigger than the system-wide parameter). Furthermore, the routing table size is set to $k = 20$. We monitor edge devices through the *nmon* tool [IBM] and retrieve information regarding the hardware utilization and the data received and sent between edge devices.

The goal of the first experiment is to assess the prototype's footprint on hardware resources and bandwidth usage. Notably, we focus on resource consumption to determine

Table 3.1: Edge neighborhood

Neighborhood	Edge Devices (per cluster)	Total
C1	15	15
C2	20	35
C3	15	50
C4	10	60

the system coordinators and discover resources in the edge neighborhood. We monitor the global coordinator for each cluster for up to 60 seconds in the edge neighborhood (i.e., running on RPi3). We also monitor bootstrap devices (i.e., RPi3s) when a new request arrives to join the edge neighborhood. Notably, we observe that the time required to join the edge neighborhood and assign it to the existing cluster is between $0.07 - 0.2$ seconds in all test cases. However, when a new cluster is required to be created, the average latency is slightly higher between $0.5 - 1.02$ seconds, but reasonable for the resource-constrained edge neighborhood. Specifically, the overall CPU and memory utilization remains almost similar. Concretely, the average CPU consumption stayed around 2.5%, while the overall memory consumption stayed around 5 MB. In all test cases, edge devices have been successfully assigned to the corresponding clusters, or new clusters are created when required. Table 3.2 presents a detailed overview of the resource consumption and latency for edge devices to join the neighborhood.

Table 3.2: Average latency and resource consumption to join the edge neighborhood

Clusters	Latency (avg.)	CPU (avg.)	RAM (avg.)
C1	71.25 ms	2%	2 MB
C2	82 ms	2.5%	3 MB
C3	137.36 ms	2.5%	2 MB
C4	193.6 ms	3%	5 MB

Figure 3.4 plots the overall CPU utilization during the process to determine the global coordinator (i.e., the global and cluster coordinators are determined) and synchronization processes running in the background. Essentially, we simulate the situation when the global coordinator is overloaded, and the function to determine the new global coordinator is automatically triggered. We repeat the process more than ten times for each test case (i.e., clusters). As shown in Figure 3.4, the overall CPU utilization is slightly increased when adding more edge devices/clusters in the edge neighborhood. Specifically, the average CPU consumption while determining the system coordinators is around 10%, while the overall memory consumption is approximately 5 MB. The most important aspect is the accuracy of assigning edge devices to a particular cluster or forming new

clusters when needed (i.e., when the cluster size is exceeded). Creating new clusters was successful in all test cases, including newly joined edge devices added to their adequate clusters. Since the overhead imposed by our approach is small, the experiments show that the proposed approach is feasible to operate on low-powered and battery-powered edge devices. Notably, the proposed approach has shown a very contained impact on hardware resources and bandwidth usage. To obtain consistent results, we calculated an 83% confidence interval of means for each experiment.

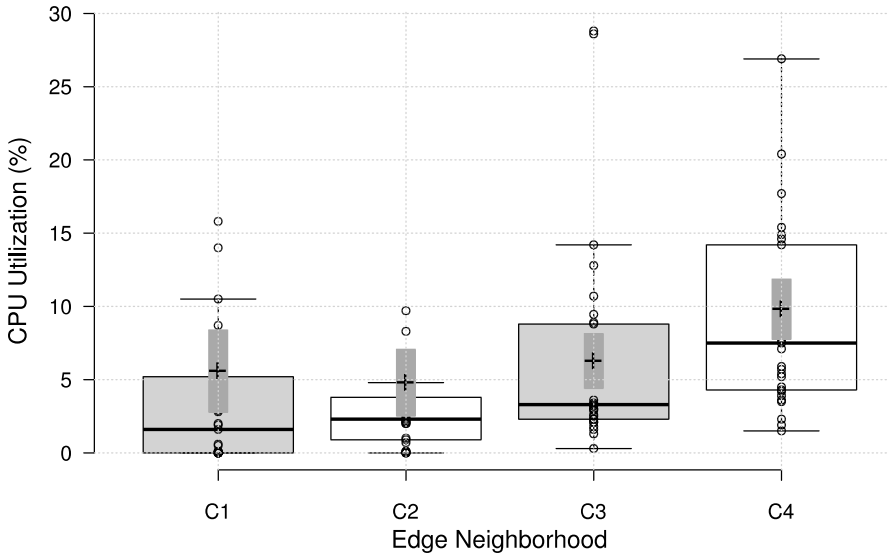


Figure 3.4: Analysis of the CPU utilization during the process to determine the system coordinators.

Figure 3.5 and Figure 3.6 plot the overall maximum and minimum data transfer/received by the global coordinator during the process to determine new system coordinators and the synchronization process. We consider analyzing the bandwidth due to the relationship between network traffic and the load on the memory system of an edge device [MSBD18]. Furthermore, in resource-constrained edge neighborhoods, the energy supply and bandwidth are among the main resource constraints of edge devices [BS07]. Therefore, it is necessary to know the total data size transferred/received (i.e., metadata sharing, processes to determine coordinators, synchronizing processes, etc.) between edge devices during the runtime. Notably, the maximum and minimum values vary depending on the number of edge devices in the neighborhood. As shown in Figure 3.6, the data transfer is slightly increased by adding more devices in the edge neighborhood. The slight increase occurs due to the increased number of edge devices that impose new coordinators. Thus, newly formed coordinators synchronize their routing tables and DHTs with the global coordinator.

The goal of the second experiment is to assess the time complexity to determine coordi-

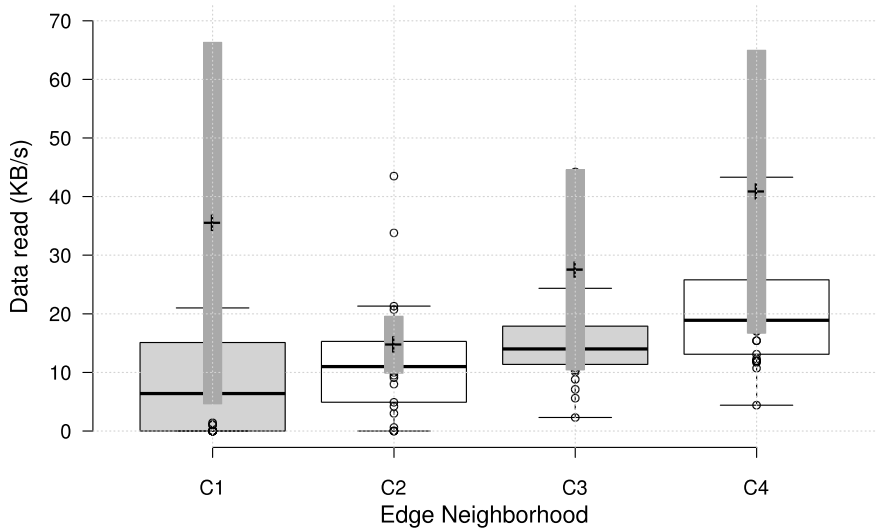


Figure 3.5: Analysis of the data received in kilobytes per second by the global coordinator.

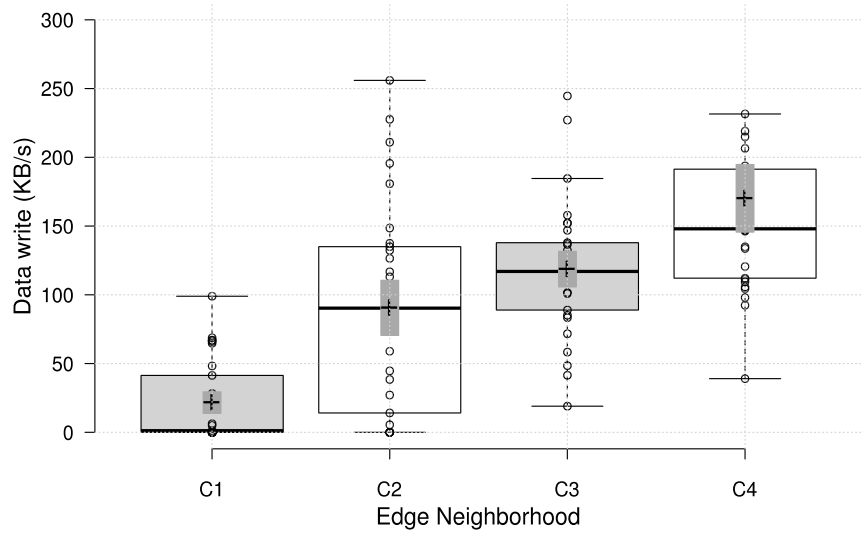


Figure 3.6: Analysis of the data transfer in kilobytes per second by the global coordinator.

nators and place them on the edge neighborhood's most suited edge devices. Concretely, we show how the proposed mechanism discussed in Section 3.3 and Section 3.4 performs to determine coordinators. The experiment results illustrated in Figure 3.7 show that

the proposed approach in all test cases finds an edge device with the most suitable computation resources to assign the global coordinator. Besides that, the proposed approach successfully determines cluster coordinators for each cluster.

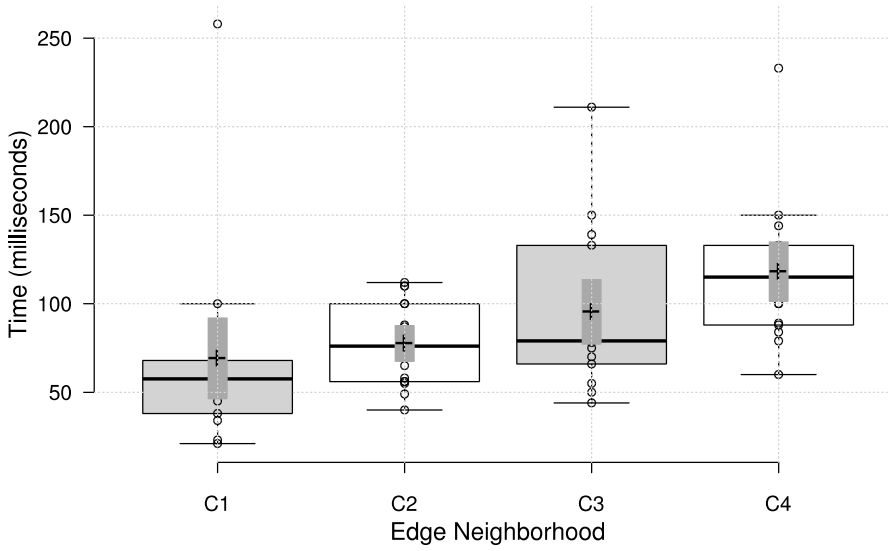


Figure 3.7: Time required to determine coordinators.

In Figure 3.8, we analyze the percentage of responsive edge devices during the process to determine the cluster coordinator on an edge neighborhood with a single cluster and different sizes (e.g., CS=15, etc.). We repeat the process more than five times for each test case. Notably, we observe that the time required to join the edge neighborhood and assign it to the existing cluster remains similar to the previous results (i.e., between $0.07 - 0.2$ seconds in all test cases). The number of edge devices that respond to this process is critical since it enables us to find the most suitable edge device to place the cluster coordinator. In clusters with more than 40 edge devices, the percentage of responsiveness is decreased to 92%. In contrast, clusters up to 30 edge devices show a higher responsiveness rate. To that end, we configure the system-wide parameter to bound the maximum number of edge devices per cluster (i.e., 30 edge devices).

3.5.3 Use case: Deploying IoT safety application

The third experiment demonstrates the proposed mechanism's function in the application deployment process. For the demonstration purpose, we adapted and extended parts of the FogTorchΠ simulator [BFGL19] to generate deployment plans in the edge infrastructure. FogTorchΠ was originally proposed to support IoT designers in deciding where to deploy application components at the edge. The approach allows specifying IoT resources as an application requirement that must be met before applications can be deployed.

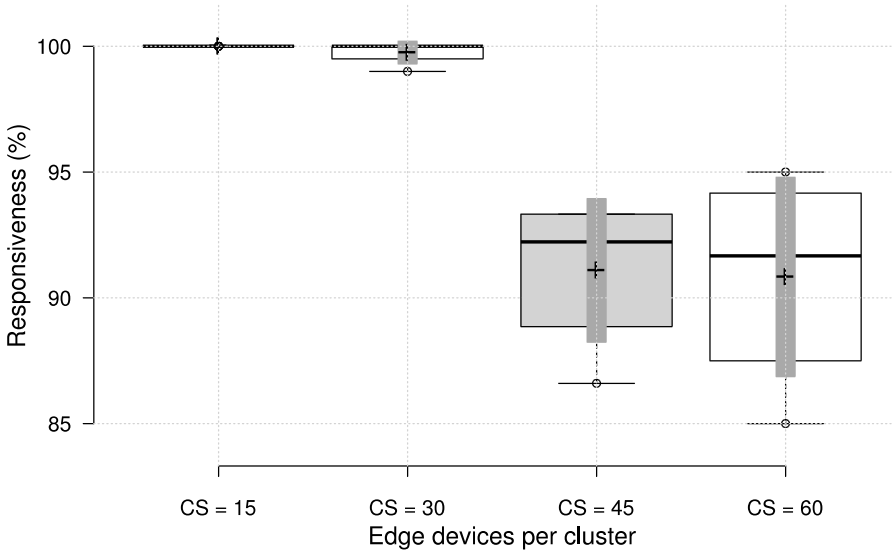


Figure 3.8: Percentage of responsive edge devices during the process to determine coordinator on clusters with different sizes.

We assume edge devices contain a set of built-in IoT resources (i.e., cameras, radars, gas sensors, etc.) that can be accessed remotely through APIs. Recalling our motivation scenario, consider the IoT safety application that provides a service to rescue teams (as discussed in Section 2.3.1). As illustrated in Figure 3.9, the IoT safety application comprises three components: i) the insights backend component (φ_0), ii) the monitoring component (φ_2), and iii) the processing component (φ_1). The insights backend component (φ_0) enables users to interact with the service. The monitoring component (φ_2) is responsible for monitoring resources specified at design time. The processing component (φ_1) analyzes collected data into meaningful results and provides it to the end-users through the insight component. We assume deploying IoT safety application in the edge neighborhood as presented in Figure 3.1. Notice that only a single application component can be executed in parallel on edge devices. The other application requirements depicted in Figure 3.9 are assumed to be met by all edge devices.

In our given edge neighborhood, the edge device E_2 provides the radar resource with privacy preference set to private. This means that the φ_1 can be deployed only in E_2. The rest of the IoT resources (e.g., cameras, gas sensors, etc.) are remotely accessible, and their privacy preference is public. Through the metadata replication process, edge devices exchange public resources in the edge neighborhood. To this end, each edge device is considered a potential candidate to execute one of the two other components. Meanwhile, each edge device shares private resources only with the global coordinator, responsible for generating deployment plans for the IoT safety application. Figure 3.10 shows the total number of eligible deployments plans generated through FogTorchΠ combined with our

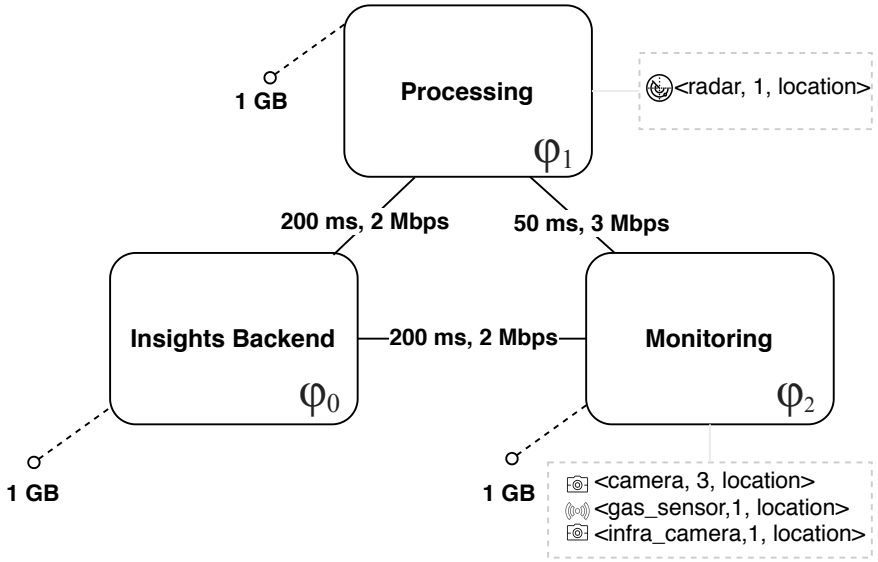


Figure 3.9: IoT safety application.

resource discovery mechanism. It is evident that φ_0 and φ_2 components dependent on particular resources can be executed on all other edge devices. More precisely, each edge device knows how to access resources on other edge devices due to metadata replication.

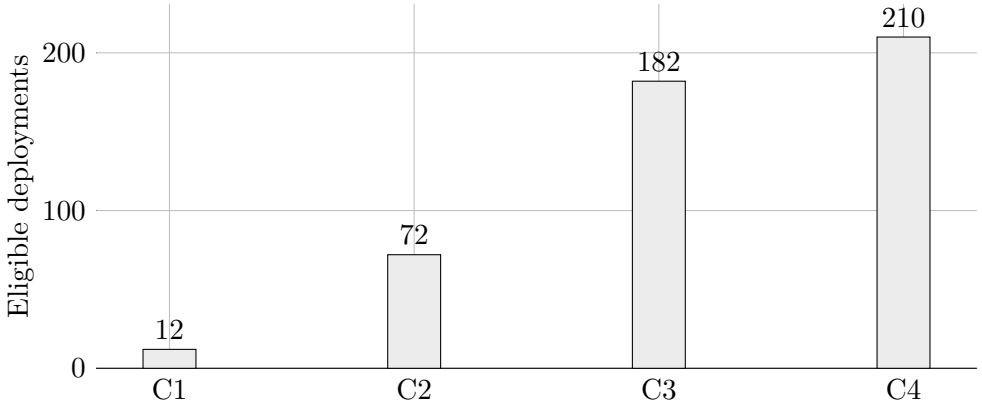


Figure 3.10: Generating deployments plans for the IoT safety application in an edge neighborhood (see Figure 3.1).

Unlike our approach, FogTorchΠ does not support resource privacy preferences, does not provide a mechanism to discover resources (i.e., supports resource sharing with neighbor devices), and does not support the dynamic changes in their environments. Besides that, evaluating the deployment mechanism is out of the thesis scope, as we use it to demonstrate that the discovery mechanism allows us to completely exploit available resources in edge neighborhoods.

3.5.4 Discussion

We have demonstrated that by using our resource discovery mechanism, discovering heterogeneous resources in an edge setting increases the number of eligible deployments for applications dependent on IoT resources. We showed that organizing edge devices in clusters and discovering resources through the edge replication process is performant and feasible on a testbed with low-powered ARM-based devices. Our results showed that the overhead introduced by the proposed decentralized resource discovery mechanism is very low for realistic edge neighborhood sizes.

Our approach within the given testbed can operate with larger cluster sizes. However, edge devices may not react in time to participate in processes to determine coordinators in large cluster sizes. In such cases, the Kademlia Protocol requires much more processing capabilities to process faster incoming requests (i.e., updating routing tables, updating DHTs, resource metadata, etc.). Nonetheless, we plan to improve and optimize the current mechanism to form larger clusters in edge neighborhoods. We plan to investigate the maximum edge neighborhood size and the routing table size acceptable for low-powered edge devices.

Our approach does not provide a mechanism to detect which edge devices fail to send their parameter metrics and ask them to resend their messages. In addition to that, since multiple edge devices in an edge network come to the same result independently, it is necessary to introduce an additional mechanism to verify the result. In our approach, we overcome such an issue by distributing results using the DHT. This creates some redundancy - but also improves the stability of propagating results. However, using consensus algorithms where edge devices pick random participants to verify their final result is highly desirable. Thus, some assumptions underlying our methodology must be further explored.

3.6 Summary

Edge networks provide a seamless opportunity for deploying various edge applications providing multiple services to the end-users and the surrounding IoT devices. However, we require novel lightweight and decentralized mechanisms to automatically discover heterogeneous resources at the edge to deploy edge applications dependent on IoT resources. To that end, we introduced a decentralized mechanism that enables edge devices to connect in a P2P manner, organize edge devices in clusters, and support the automatic discovery of heterogeneous resources in edge networks. The proposed approach support resource discovery based on resource privacy preferences. Furthermore, we evaluated our approach in a testbed composed of a set of low-powered edge devices. Throughout the experiments, we showed the feasibility of the proposed approach to run on low-powered edge devices. We believe that the proposed approach paves the way for utilizing available resources and accomplishing the promised high-quality and low-latency services deployed in edge networks (i.e., edge neighborhoods).

CHAPTER 4

On Controlling Elasticity of Edge Applications

The broad range of edge application requirements combined with heterogeneous, resource-constrained, and dynamic edge networks make it particularly challenging to configure and deploy them. Besides that, missing elastic capabilities on edge makes it difficult to operate such applications under dynamic workloads. To this end, this chapter proposes a lightweight, self-adaptive, and decentralized mechanism (DECENT) for (1) deploying edge applications on edge resources and on-premises of Edge-Cloud infrastructure, and (2) controlling elasticity requirements. DECENT enables developers to characterize their edge applications by specifying elasticity requirements, which are automatically captured, interpreted, and enforced by our decentralized elasticity interpreters. In response to dynamic workloads, edge applications automatically adapt in compliance with their elasticity requirements. We discuss the architecture, processes of the approach, and the experiment conducted on a real-world testbed to validate its feasibility on low-powered edge devices. Furthermore, we show performance and adaption aspects through an edge safety application and its evolution in elasticity space (i.e., cost, resource, and quality).

The rest of this chapter is structured as follows. Section 4.2 gives an overview of the platform along with a running example. Related work is considered in Section 4.3. Section 4.4 describes the edge application and system modeling, along with describing application requirements. In Section 4.5, we describe in detail the DECENT framework, the processes, and details regarding prototype implementation. Evaluation and results are presented in Section 4.6. Finally, Section 4.7 concludes the chapter.

4.1 Introduction

As the newly introduced paradigm, Edge Computing [SCZ+16] is a key enabler for IoT proliferation. In contrast to cloud infrastructures, edge networks are resource-

constrained environments. However, a wide range of available resources at the edge have introduced new opportunities such as deploying low-latency, privacy-awareness, and resilient *edge applications* (e.g., IoT applications). Besides many benefits introduced by edge devices, analyzing high-volume IoT data streams on a single device through monolithic applications poses many limitations and challenges in terms of processing capabilities, storage, energy, and communication bandwidth. To that end, modern applications are no longer monolithic [ABL15]; such edge applications (i.e., *services*) are divided into a set of independently deployable software components (i.e., *microservices*) and distributed over edge resources or on-premises of Edge-Cloud infrastructure. Similarly, resource management techniques must be designed as decentralized systems to run in resource-constrained environments. Thus, this brings altogether new challenges where novel lightweight resource management techniques are needed to fully utilize available resources at the edge. Nonetheless, the broad range of requirements concerning latency, Quality of Service (QoS), or fault-tolerance, combined with edge networks' heterogeneous and dynamic nature, make it particularly challenging to manage, configure, deploy, and operate such applications.

Over the past few years, researchers have been widely focused on proposing multiple resource allocation techniques at the edge [SCZY17]. However, less attention is given to provide elastic features at the edge [HV19, MKB18]. In most cases, elasticity refers to a system's capability to adapt to workload changes by (de)provisioning resources in an automatic manner [HKR13]. Resource demands for a particular running application or component may change over time. Consequently, this may cause poor overall performance and higher latency than the expected response time. For instance, consider a scenario where a health application running in an edge network (e.g., smart home) monitors residents' health through processing data streams created by the users' smartwatches. At the time t_0, a single edge device has sufficient resources to monitor only one resident's health. At the time t_1, there is more than one resident, and the edge device may not have enough resources to process produced data for all residents. To avoid such situations, edge applications should scale over edge resources or on-premises of Edge-Cloud infrastructure. Therefore, introducing *elasticity* features at the edge is crucial.

The elasticity concept is heavily related and used in cloud computing, and often is considered one of the main features of the cloud paradigm [DGST11]. In Edge Computing, very few works propose methods for controlling application elasticity [TTT+18, FACP18, NPP+20, FJFCSG+19]. Current approaches exhibit several limitations, such as they are built as centralized systems, application-specific, and enabling application scaling only by considering hardware resources and their capacity to scale. Furthermore, centralized approaches are sensitive to edge systems characteristics (i.e., resource-constrained, dynamic, and uncertain). Thus, edge networks' dynamic nature requires continuously re-evaluating placement decisions for edge functions. Nevertheless, in our conception, besides *resource* requirements, elasticity in three-tiered infrastructures should also target their relations with the different types of *costs* and *quality*.

To address the aforementioned challenges, we propose a lightweight framework called De-

centralized Configurator for Controlling Elasticity in Dynamic Edge Networks (DECENT) and its runtime mechanism for controlling elasticity requirements in edge applications. DECENT enables deploying and scaling edge applications in dynamic edge networks and on-premise of Edge-Cloud infrastructures. Essentially, the developer defines application requirements, elasticity requirements, and a scaling model for each edge application and its components. The DECENT interprets these requirements, deploys components in the three-tier architecture, and enforces various scaling operations at runtime to fulfill edge application demands. The system we propose enables easy configuration, deployment, and operation of edge applications on top of heterogeneous edge infrastructure. Furthermore, DECENT is a self-adaptive and decentralized mechanism that can be easily deployed and run on low-powered edge devices.

Our concrete contributions are as follows.

- We enable application developers or domain experts to specify high-level elasticity requirements for their edge applications in a declarative way. Besides that, the user specifies the deployment and scaling model inherent to a specific infrastructure configuration. To specify edge application elastic requirements, we consider the declarative language called *Simple Yet Beautiful Language (SYBL)* [CMTD13]. We extend the language and focus on developing novel constraints and enforcement strategies to support edge application characteristics. In our conception, besides *resource* requirements, elasticity in three-tiered architectures should also target their relations with the different types of *costs* and *quality*.
- We extend the prototype of [MD21b] with a lightweight mechanism that enables deploying and controlling the elasticity of edge applications in a decentralized manner at an edge network. Edge devices that run application components capture and interpret their elastic requirement through *Elasticity Interpreters* and report to the configurator device whether the requirements are violated. The configurator device takes action and re-configures the application to meet the specified elastic demands.
- To validate the approach's feasibility, we perform an experimental evaluation that shows that an edge application and its components can easily scale and be controlled by the mechanisms deployed at the edge of a network. Our prototype is evaluated on a real-world testbed comprising several low-powered edge devices.

4.2 Running Example

Referring to the situation illustrated in Section 2.3.1, respectively Figure 2.2, we assume that a rescue team deploys (1) the lost-person service (i.e., edge safety application) in the affected area. Such service aims at helping rescue teams solve missing person cases faster by finding their location in the affected zones (2). The service is dependent on camera resources, which are integrated into various drones. Specifically, the lost-person service

comprises components responsible for specific tasks (i.e., front-end, image processing, generating results, storing results, etc.). The service takes as input images provided by the flying drones in the affected area. Each drone every second generates various images to be processed by the service. However, with the increasing number of drones, the number of generated images is increased (5). To that end, application components may require more computing resources to process images such that application requirements are fulfilled. For instance, consider that the front-end component that accepts drones' images has a response time requirement that should be less than 100 ms. When the response time requirement is violated (e.g., 100 ms), the service component must scale to multiple instances on edge or on-premises (3-4) to meet the desired service quality. Thus, it is evident that to meet service demands at runtime and to avoid resource over-provisioning/under-provisioning, we require a lightweight mechanism that dynamically controls application elasticity at the edge.

4.3 Related Work

Research efforts associated with the elasticity at the edge are still at a relatively early development stage. Elasticity features in edge infrastructures mostly have been focused on scaling up/down *resources* to meet application demands. In some approaches, tasks/services are reallocated when devices are overloaded [SKR+18]. However, such practices, in turn, incur an overhead of resource usage, increased cost, and increased energy consumption. Even though resource over-provisioning can be considered feasible in resource-rich environments such as the cloud, such an assumption is highly impractical in resource-constrained edge networks. More specifically, reserving resources more than needed to support the intended task workload wastes available resources.

Very few approaches address these challenges at the edge. Furst et al. [FACP18] introduce a new framework that enables services to self-adapt and meet the current service demands of their Service Level Objectives (SLOs). A novel programming model called Diversifiable Programming (DivProg) uses function annotations as an interface between the service logic, its SLOs, and the execution framework to achieve such an adaption dynamically. Essentially, a third-party execution framework captures service configuration given by the developer through DivProg, interprets, and scales services that conform to changing SLOs. Tseng et al. [TTT+18] provide a lightweight autoscaling mechanism for fog computing in industrial applications. Lujic et al. [LT19] propose a novel, holistic approach for architecting elastic edge storage services, featuring three aspects such as data/system characterization (e.g., metrics, key properties), system operations (e.g., filtering, sampling), and data processing utilities (e.g., recovery, prediction). The authors [NPP+20] discuss how applications for a fog infrastructure can be packaged into containers and act elastically. Their approach is built on top of the container orchestration tool Kubernetes and extends it to the fog. In [ZFJB18], the authors investigate the benefits of virtualization to move and redeploy mobile components to the fog devices closest to the targeted end devices. By using geometric monitoring, the proposed approach dynamically scales and provisions the resources for the fog layer.

Any Edge-Cloud system's goal is to hide the complexity of edge applications deployment and operations in heterogeneous edge networks and enable developers to specify application requirements in a declarative way. A domain-specific language (DSL) specifies the high-level constraints of edge applications, such as QoS, application criticality, and elasticity requirements. The DSL essentially makes it easy for users to develop these specifications. Understanding the current and future requirements of edge applications from various domains remains a prominent challenge. A platform for the described IoT scenarios (e.g., as in our running example) needs to hide this operational complexity from application developers. In particular, programmers should not have to worry about the distribution of data and edge or cloud resources' heterogeneous capabilities. Developers should be able to express the context in which applications are allowed to run and the elasticity requirements in a high-level way [DGST11]. The platform should then take care of resource provisioning and data movement. However, this requires that the programming model and API are intuitive for developers but expressive enough to help the execution platform make runtime decisions on scheduling edge application components.

Finally, our work is an effort to advance edge computing platforms' current state and enable more straightforward configuration, deployment, and operation of edge applications on top of heterogeneous edge infrastructure. The above-mentioned systems are extremely limited in their operational capabilities and lack of self-adaptive mechanisms required in dynamic edge and IoT settings. In essence, such systems assume static configurations that do not change over time, provide no way to specify elasticity or QoS requirements, and do not have a mechanism to enact them. Our proposed approach aims to bridge this gap and ensure multi-dimensional elasticity control (i.e., cost, resource, and quality) for fulfilling edge application demands deployed on Edge-Cloud infrastructure. It enables edge applications and their components to adapt to dynamic changes in their workload. Finally, we enable developers to easily define elasticity requirements captured and executed by our lightweight mechanism in a decentralized manner.

4.4 Edge Modeling and Elastic Requirements

This section formally defines the concepts of edge applications, the edge system, deployment and scale policy, and elasticity requirements. First, we model edge applications and the system. Then, we describe application deployment and scaling models in Edge-Cloud architectures. Finally, we extensively explain application elasticity requirements, which enable developers to characterize their edge applications.

4.4.1 Edge Application and System Model

A service-based edge application a_i can be described as a set of components $H=\{h_1, h_2, \ldots, h_m\}$ divided by the developer prior to deployment. In fact, to enable the execution of such components on low-powered devices and for better utilization of distributed resources dividing edge applications into small components is crucial. To this end, edge application components may be distributed and executed on various

available devices such that their resource requirements are fulfilled upon deployment. Each component h_i can be modeled as a Directed Acyclic Graph (DAG) where vertices represent components and edges represent their dependencies as given in Figure 4.1.

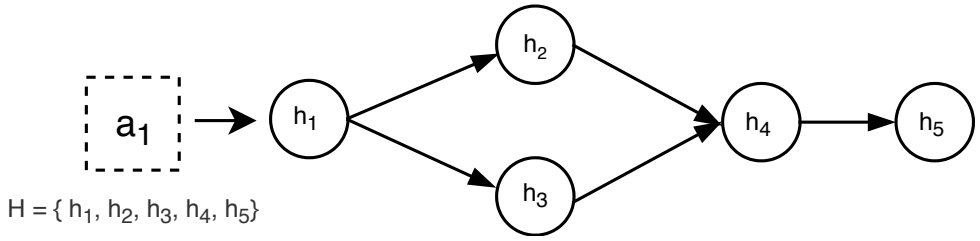


Figure 4.1: Edge application model.

Each component h_i is a black-box, representing a set of instructions aiming to provide specific functionalities. Various hardware-related requirements characterize each component (e.g., processing, memory, storage) and latency requirements between components [BF17]. Moreover, components are characterized by varying workloads, and their deployment should also be properly adapted at runtime. To that end, for each component, we have the following: (i) *the edge application resource requirements*, (ii) *elasticity requirements*, and *(iii) deployment and scaling policy*. In the following subsections, we have discussed both application requirements in detail.

The Edge-Cloud architecture consists of edge infrastructures (i.e., multiple edge devices connected in a peer-to-peer manner form an edge infrastructure), fog infrastructure, and cloud infrastructure. As mentioned in Section 3.2.1, our approach is built on top of an edge network, which is built as a Distributed Hash Table (DHT) network [MM02]. We assume that every edge device trusts all devices to establish a direct communication link; they belong to the same local administrative domain. Furthermore, in this work, our primary focus resides on edge infrastructures while we assume that the fog and cloud infrastructures are considered Infrastructure as a Service (IaaS) [MS14]. We assume that the system designer configures the edge network to connect to the IaaS services.

Executing components on heterogeneous environments (i.e., edge tier, fog tier, or cloud tier) is crucial. For instance, an application with multiple components can scale on multiple instances running on different locations in either edge or cloud, depending on current demands and the application's constraints. To overcome the challenges introduced with heterogeneous environments, we consider Docker[1] as our homogeneous application runtime platform that follows the *"run once, run anywhere"* model. A docker platform represents a lightweight, stand-alone, executable package that contains everything needed to run the specifically added component. The application runtime is essentially responsible for executing edge applications (i.e., container-based) on edge devices or on-premises. Thus, to deploy edge applications in Edge-Cloud infrastructure, components are packaged in individual docker containers.

[1]Docker, https://www.docker.com/

4.4.2 Deployment and Scaling Models

In our conception, edge applications can be thought of as a set of deployable software components running on-premises of the Edge-Cloud infrastructure. As mentioned in Chapter 2, edge application components can be deployed and scaled according to the following models: (1) Everything in the Cloud, (2) Everything in the Edge, and (3) Hybrid Edge-Cloud.

4.4.3 Elasticity Requirements

Elasticity properties at the edge are crucial to execute edge applications and fulfill their dynamic demands. In Edge-Cloud architectures, elasticity targets not only resources and their capacity to scale but also their relationships with various forms of costs and quality [DGST11]. In this context, multiple stakeholders may be involved in specifying elastic requirements. For instance, the developer could specify that the latency between application components must not reach 20ms without determining how many resources should be used to achieve the desired state. An edge network provider could specify its resource utilization schema; for example, when overall utilization at the edge is higher than 90%, it enables scaling applications toward fog or cloud infrastructure. To enable such a feature, we consider a declarative language called Simple Yet Beautiful Language (SYBL) to allow users to specify an edge application's elasticity requirements at the design time [CMTD13]. We extend the language and focus on developing novel constraints and enforcement strategies to support edge application characteristics running on Edge-Cloud infrastructures. In addition, we extend and optimize the language runtime engine to support controlling Docker-based edge applications and enable execution on low-powered edge devices. We provide a time-based mechanism that analyzes workloads generated by incoming requests to optimize and avoid unnecessary scaling operations. More specifically, the time-based mechanism controls for a few seconds (i.e., a configurable value, e.g., 20 seconds) if the increased load on a particular component is handled without executing any scaling operation. The feature mentioned above is useful when the increased or decreased load in a component is occasionally and not persistent.

SYBL enables users (i.e., developer or system user) to specify application elasticity requirements in a declarative way represented in the form of (i) *monitoring* (i.e., specifying which metrics to monitor), (ii) *constraints* (i.e., specifying the limits in which the monitored metrics can oscillate), (iii) *strategies* (i.e., specifying actions to be followed in case the constraint is violated or becomes true), and *priorities* (i.e., specifying constraints with higher priority than the other ones). The user can specify elastic requirements at different levels of edge application. Thus, elasticity controls can be achieved at the (i) *edge application level* (i.e., specifying high-level application elastic requirements), and (ii) *edge component level* (i.e., specifying low-level application elastic requirements). Listing 1 shows an example of elasticity requirements specified by the user. At the edge application level, the user may specify the maximum cost allowed for the entire edge application executed in Edge-Cloud infrastructures. The user could specify that the application needs to scale down when the cost is high, and CPU usage is below 20%. Or, when the

cost is below the predefined value (e.g., 5 euros) and the CPU usage is higher than 80%, the application needs to scale up. At the edge component level, for instance, elasticity requirements from the developer side can be applied regarding the quality, such that, e.g., if the edge device battery is less than 10%, the component must scale up to avoid application failure.

```
#ApplicationLevel.AppID
Mo1: MONITORING cpuUsage = cpu.usage
Mo2: MONITORING memUsage = mem.usage
Co1: CONSTRAINT totalCost > 5Euro AND cpu.usage < 20
Co2: CONSTRAINT totalCost < 5Euro AND cpu.usage > 80
St1: STRATEGY CASE Violated(Co1): ScaleIn
St2: STRATEGY CASE Violated(Co2): ScaleOut
#ComponentLevel.AppID.componentID
Co3: CONSTRAINT mem.usage < 80 AND cpu.usage < 20
Co4: CONSTRAINT battery.Level < 10
St3: STRATEGY CASE Violated(Co3): ScaleIn
St4: STRATEGY CASE Violated(Co4): ScaleOut
Pr1: Priority (Co4)>Priority(Co3)
```

Listing 4.1: An example of elastic requirements.

The SYBL elastic requirements can be easily injected/integrated into various description languages. For instance, the elastic requirements can be easily integrated into cloud application description language TOSCA standard [2], docker-compose files (i.e., YAML), JavaScript Object Notation (JSON), or specified separately through XML descriptions. In the current version of our prototype, we specify edge application elastic requirements in a JSON file. Nevertheless, future work remains to provide a mechanism that will inject elastic requirements easily in the YAML file (i.e., since our application runtime platform considered is Docker).

4.5 DECENT - Design and Processes

This section provides an overview of our approach to deploying service-based edge applications and controlling their elasticity on an edge network. We extensively outline the main components of the DECENT and the interaction of its components during the runtime.

[2]OASIS, Topology and Orchestration Specification for Cloud Applications (TOSCA), http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html

4.5.1 System Overview

An overview of our approach at design time and runtime of the system is illustrated in Figure 4.2. The developer defines the edge application model and its requirements at design time, as described in the previous section. Afterward, the developer specifies each component's application structure and resource requirements. The elasticity requirements and deployment policy can be determined by the developer as well as by the system user (i.e., owner) before the deployment phase. The deployment process starts when the user requests the system's configurator device to deploy an edge application. Exploring the module that enables users to interact with the system is beyond this thesis scope.

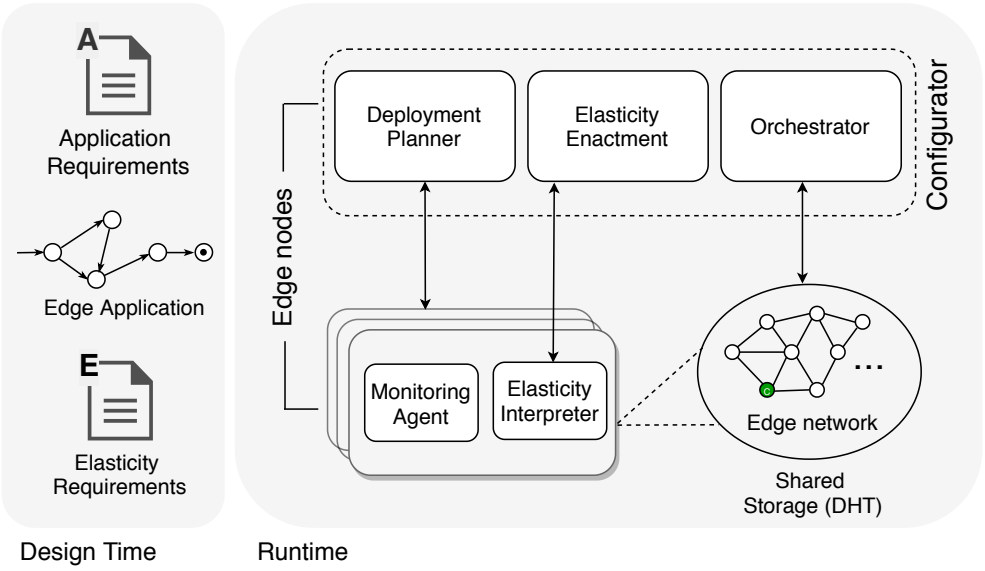


Figure 4.2: Overview of DECENT's components and their interaction during the runtime.

Each edge device consists of similar system components, as illustrated in Figure 4.2. However, the edge device that becomes the system's configurator provides the main features to control edge applications and runtime aspects. The architecture of the approach comprises five main modules, as described in the following.

`Deployment Planner(DP)`**:** This module aims to generate QoS-aware deployment plans for deploying edge applications on-premises of Edge-Cloud infrastructure. The `DP` module provides two main sub-modules: i) *Planner* and ii) *Resource Manager*. The Planner generates deployment plans for a given edge application by considering both its app requirements and deployment policy. The Planner is essentially responsible for finding all possible eligible deployment plans by considering application hardware requirements (i.e., CPU, RAM, and storage), bandwidth, and latency between components. Moreover, deployment and scaling policies tell the Planner which infrastructures can be considered when deploying application components. The Planner gets the current state of the

infrastructure(s) through the Resource Manager. The Resource Manager is responsible for monitoring and storing the infrastructure-specific metrics such as resource capabilities of edge devices (i.e., hardware), their current resource utilization, and link latencies between devices. Such information is provided by *Monitoring Agents* deployed at each edge device.

`Monitoring Agent (MA)` **:** This module measures a set of infrastructure-specific metrics and application performance metrics continuously. The infrastructure-specific metrics are CPU, RAM, storage, battery levels (when applicable), and their current utilization. Nevertheless, the `MA` module continuously measures application metrics such as hardware-related resource consumption, response time, application status, etc. However, note that our approach monitors only metrics specified in the elasticity requirements. The `MA` module periodically sends information to the Resource Manager. Such data is temporally stored locally on the configurator device, and it is regularly updated when an application is deployed. This thesis does not investigate how the monitoring agents are implemented in cloud and fog environments. Several studies [BFG19, FGB20] address relevant aspects for monitoring agents as well as several monitoring tools exist for providing the necessary information[3,4].

`Elasticity Enactment Engine (3E)` **:** This event-driven module handles the coordination between the application's desired state and the current application elasticity state. In the DECENT runtime, elasticity requirements are interpreted by the *Elasticity Interpreter* deployed on each edge device. When a component's elasticity constraint is violated, the Elasticity Interpreter communicates such information to the `3E` module. Afterward, the `3E` module enforces required actions such that the component requirements are fulfilled. Thus, the `3E` module is the central part of the runtime system, which manages edge applications. In contrast, Elasticity Interpreters are local runtime engines that capture application component elastic requirements, interpret and communicate necessary actions to the enactment engine. The following section briefly outlines the overall process of managing edge applications at runtime.

`Orchestrator` - This module provides several functionalities to support executing edge applications in dynamic edge networks. The Orchestrator functionalities are mostly addressed in Chapter 3 and are not evaluated in this Section. In addition to that, the Orchestrator is responsible for creating, controlling, and managing the cluster of Docker Engines called *swarm*. Essentially, the system's configurator device simultaneously is the *swarm manager*, and the other edge devices are *worker devices*. The swarm manager maintains the swarm state through Raft Consensus Algorithm[5]. On the contrary, the Orchestrator is responsible for keeping the quorum of managers in the system consistent.

At system design time, the Orchestrator is configured regarding the number of swarm manager devices that should be consistent at runtime and the expected size of the edge

[3]Nagios, https://www.nagios.com/
[4]Ganglia, http://ganglia.sourceforge.net
[5]Raft Consensus Algorithm, https://raft.github.io/

network. The system designer should consider a trade-off between performance and fault-tolerance when it defines the number of swarm managers. Having more swarm manager devices makes the system more fault-tolerant while writing performance is reduced (i.e., due to the network round-trip traffic). We configure an odd number of swarm manager devices to take advantage of swarm mode's fault-tolerance features. The Orchestrator promotes new swarm manager devices whenever the edge network doubles the expected network size. Notice that we may have a maximum of five managers in an edge network.

Nevertheless, the Orchestrator periodically monitors the desired swarm manager number (i.e., system designer perspective) and the current number of swarm managers. Thus, if the desired state is violated, it takes the required actions to keep swarm managers' quorum in the system. Notice that the Orchestrator configuration data (i.e., swarm managers, swarm cluster joining key, etc.) is stored as DHT, meaning that it is shared and kept consistent between all devices within the entire network.

4.5.2 The Process

Edge applications are multi-container Docker-based applications. This means that the developer defines components that make up an edge application, including their hardware requirements specified in the *docker-compose file*. Essentially, an edge application and each component have their unique name when deployed. Furthermore, at design time, the user specifies the deployment and scaling model and the elastic requirements (as presented in Listing 1). Both these requirements are formatted and stored as a single JSON file. Thus, we assume that the mentioned requirements are given at the design time.

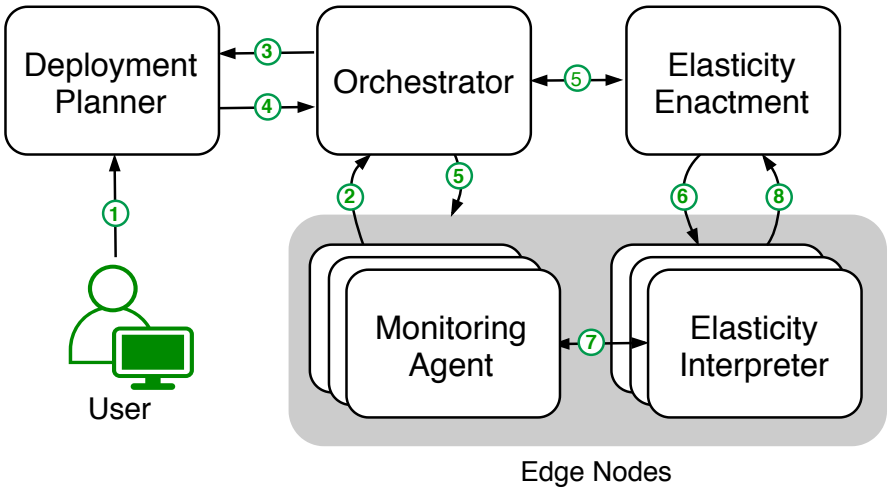


Figure 4.3: The process at runtime.

The process starts when the user requests (1) the configurator device to deploy an edge application at an edge network (as illustrated in Figure 4.3). At this phase, the deployment planner interacts with the Orchestrator to get the current hardware infrastructure status, available edge devices, resource information, resource utilization rates, and latency of the communication links between devices (2)-(3). Afterward, it gets the edge application docker-compose file and specified hardware requirements and generates all eligible deployment plans (3). To generate such plans, it runs the algorithm presented in [BF17]. In essence, for each application component, the `DP` module finds all possible devices that fulfill the component's hardware requirements. Furthermore, the `DP` module notifies the user whether the edge application can be deployed at the specified deployment and scaling model. For instance, if the deployment and scaling model is set to the only *edge*, it means that all application components must be deployed at the edge (if possible).

Suppose the `DP` module generates at least one or more deployment plans. For each component, we have a list of compatible devices that can run them. Afterward, the `DP` module gets the list and updates the docker-compose file by adding the placement constraints. This means that for each component, it specifies devices where the component can be deployed. After this process is finished, the configurator executes Docker-based commands to deploy the edge application. Deploying and starting components (i.e., containers) can take several seconds and depend on edge device hardware capabilities. However, in this thesis, we do not investigate performance aspects when deploying and starting containers. A study that acknowledges the problem and addresses relevant aspects is presented in [AP18].

Through the Orchestrator module, the configurator shares elastic requirements with edge devices (5). Elastic requirements are shared by using DHT. Essentially, each edge device automatically identifies when the configurator assigns a container (e.g., named φ_1) to them. Thus, when φ_1 is running, elasticity interpreters on each device query DHT to receive elastic requirements for the running applications. The user can change elastic requirements at runtime. The changes made on elastic requirements (i.e., in the configurator device) are automatically updated by other edge devices and captured by corresponding Elasticity Interpreters. Afterward, before starting elasticity monitoring (7), the elasticity interpreter first checks whether it is the only device running φ_1. The configurator device provides information (6), and such information is required to avoid situations where multiple elasticity interpreters start monitoring φ_1 (i.e., when φ_1 runs on several devices). Moving on, consider a situation when an elasticity interpreter monitors a constraint that says the edge application component requires to scale up when it uses 80% of the edge device CPU (e.g., see Listing 1). Thus, when the specified constraint is violated, the interpreter communicates it to the `3E` module (8), which is responsible for enforcing the scaling operation. Note that monitoring agents provide hardware-related metrics and container-based metrics.

The `3E` module is triggered when it receives information to enforce a specific strategy in the particular application component. In essence, enforcement operation for the

violated constraint is specified in elastic requirements. Before executing the action, the 3E module checks whether the application component should scale up or down. If the application component requires scale-up, it requests the DP module to check whether the component can scale in the current infrastructure state. We apply this strategy to avoid enforcing scaling operations in infrastructure with insufficient resources. Otherwise, the Docker runtime environment will continuously try to scale the application component without the mentioned strategy, causing network congestion and computation overload. Moreover, the configurator device for each edge application periodically monitors elastic requirements at the edge application level. The overall resource usage (i.e., for all running components) is considered and analyzed to determine whether constraints are violated at the application-level requirements.

In case the configurator device fails, edge devices hold an election to find a new configurator device as presented in [MD21b]). Elasticity interpreters contact other swarm managers to enforce scaling operations until a new configurator device is elected. Since all edge devices keep consistent data through DHTs, the newly elected configurator is initialized quickly by considering the locally stored data. Nevertheless, each device in the network knows the system's current configurator device at any time. Note that the aspects mentioned above are primarily addressed in Chapter 3 and are not evaluated in this Section. Furthermore, edge applications running at the edge network are not affected by possible configurator failure.

4.6 Evaluation

This section first presents details about the prototype implementation, setup environment, and limitations. Furthermore, we experimentally evaluate the approach's effectiveness and present the evolution of an edge application in elasticity space. We conclude with a discussion in Section 4.6.3.

4.6.1 Prototype Implementation, Setup, and Limitations

To assess the proposed approach, we extend the prototype of [MD21b] with a lightweight mechanism that enables deploying and controlling the elasticity of edge applications in a decentralized manner at the edge. The prototype is partially developed and written in Java. The prototype is tested in a realistic environment on edge devices (i.e., Raspberry Pi 3 Model B V1.2) with 4×ARM Cortex-A53 CPU at 1.2 GHz, 1 GB of RAM, and 16 GB disk storage. The prototype is deployed on each edge device, and each edge device runs the Docker Engine as the edge application runtime platform. To implement the deployment generator, we refined and extended parts of the FogTorchΠ [BF17] simulator to generate all eligible deployment plans for an edge application. In addition, the simulator does not consider dynamic environments, runtime aspects, does not provide elasticity features, does not implement monitoring tools, and does not implement any communication protocol between computation entities at the edge. Thus, the extensions we refer to are further functionalities developed to support the runtime aspects of edge

applications (e.g., elasticity, etc.) in a realistic testbed. Along these lines, the planner is fully integrated into the prototype. It gets the infrastructure state (i.e., available devices, network structure) and generates plans by considering real-time infrastructure-specific metrics.

The 3E module is implemented as a thread that runs continuously and listens for requests generated by elasticity interpreters. This module enforces various operations by Docker Java API[6] that allows building, controlling, updating, and running containers. Docker Engine REST APIs allow us to configure and update containers (i.e., running services) whenever it is needed. Additionally, some features that the mentioned APIs do not support were implemented by executing docker commands using the command-line interface. Elasticity Interpreters are deployed on each edge device, and for each running container with its elastic requirements, a single thread is executed to interpret those requirements. To implement the monitoring agent, we used *Hyperic Sigar*[7] to collect hardware information on edge devices. The network latency between devices is collected by using *ping* command. Furthermore, Docker Java API is used to collect hardware utilization data about running containers (i.e., state, CPU, memory, storage, etc.). For each container with elastic requirements, a single thread is executed to monitor specified elastic metrics. Thus, the thread number is dependent on the number of running containers on an edge device.

To evaluate our prototype, we exploited the testbed edge network comprised of ten edge devices placed close to each other. Edge devices in the testbed are connected through a wireless connection with a nominal speed of 10 Mbps and 5 Mbps in download and upload. Furthermore, the prototype's main limitation is being executed in the Java Virtual Machine (JVM) environment. We acknowledge the JVM is resource-expensive; however, we aim to show the approach's feasibility within this thesis.

4.6.2 Use Case, Experiments, and Results

We are considering an edge application (i.e., edge safety application) providing a service as described in our motivation scenario (Section 2.3.1). The edge safety application is partially developed, and it is made out of five components, with three of them written in Python (as illustrated in Figure 4.4). The front-end component φ_1 enables edge devices (i.e., drones) to interact with service and continuously upload real-time images, including location coordinates. The Redis component φ_2 collects new images and stores them in binary format. The processing component φ_3 consumes data, processes (i.e., image analysis), and stores them in the database component φ_4 (i.e., Postgres). Finally, the results component visualizes the safe path for the rescue team member residing in the affected zone.

Software components are containerized (docker images). Each container is configured with specific resource requirements (i.e., 1 CPU (1.2 GHz) and 60 MB memory) resources

[6]Docker Java API, https://github.com/docker-java/docker-java
[7]Hyperic Sigar, https://github.com/hyperic/sigar

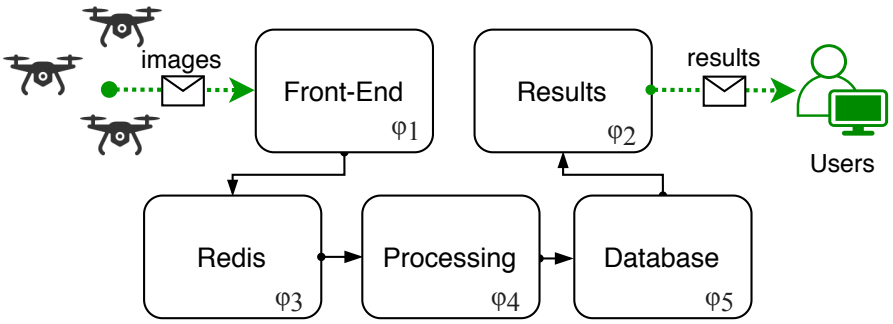


Figure 4.4: Edge safety application.

that containers can use on the hosting edge device. Notice that to reserve and use a various number of CPU resources per container, the RPi3 must be upgraded to the latest firmware[8]. Furthermore, we assume that the component images are already available on each edge device. This assumption is made due to the latency issues introduced when images are downloaded from centralized devices. In [AP18], the authors acknowledge the problem and address relevant aspects to improve deployment time.

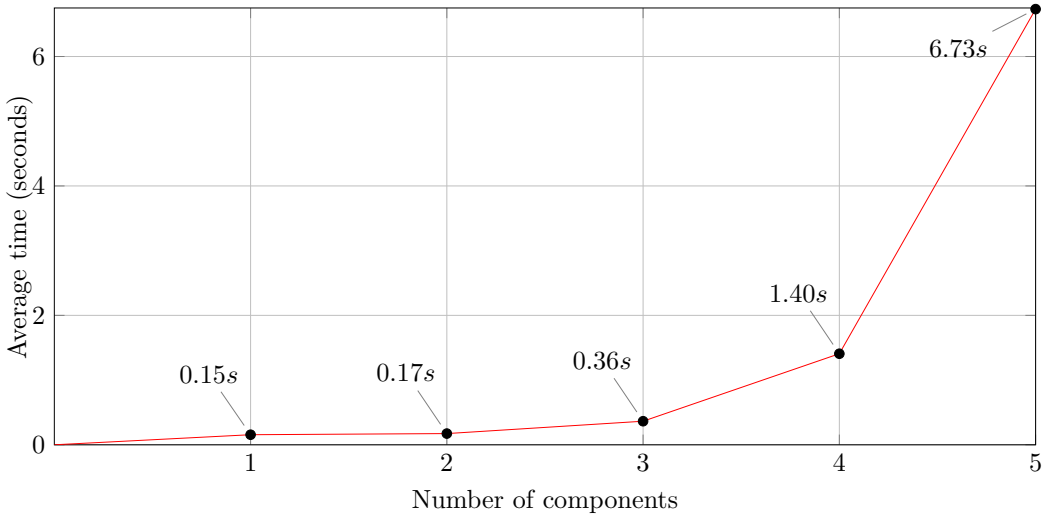


Figure 4.5: Edge safety application deployment at an edge network with ten low-powered edge devices.

Figure 4.5 illustrates the average time required to generate all possible valid deployment plans for each edge safety application component when needed to be deployed and scaled. We simulate the generation of deployment plans ten times and illustrate their maximum

[8]Raspberry Pi, https://github.com/raspberrypi/firmware

Table 4.1: Number of eligible deployments plans (i.e., in a testbed with ten edge devices)

Edge Safety Application	Generated Plans	Time (seconds)
One Component	10	0.15 s
Two Components	100	0.17 s
Three Components	1000	0.36 s
Four Components	9720	1.40 s
Five Components	84,960	6.73 s

average time requirements. For the edge safety application with five components and the given testbed, we may have up to 84,960 generated valid deployment plans (i.e., the maximum average time is 6.73s). Nevertheless, once a single valid plan is founded, the process is interrupted, and the application component(s) is deployed or scaled. Notice that when the infrastructure changes, the DP module must generate all valid deployment plans. Table 4.1 presents the maximum deployment plans generated for each software component (i.e., container). Notice the maximum time requirement and the number of generated valid plan changes based on the available resources at the edge as well as edge application requirements specified at design time.

The edge safety application is configured to run and scale only at the available devices at the edge. To simplify the scenario, we evaluate the front-end component and show the adaption process in response to the component's changing workload during its runtime. The front-end is the first component that drones (i.e., edge devices) interact with the edge application by uploading their images continuously (i.e., every 1 sec).

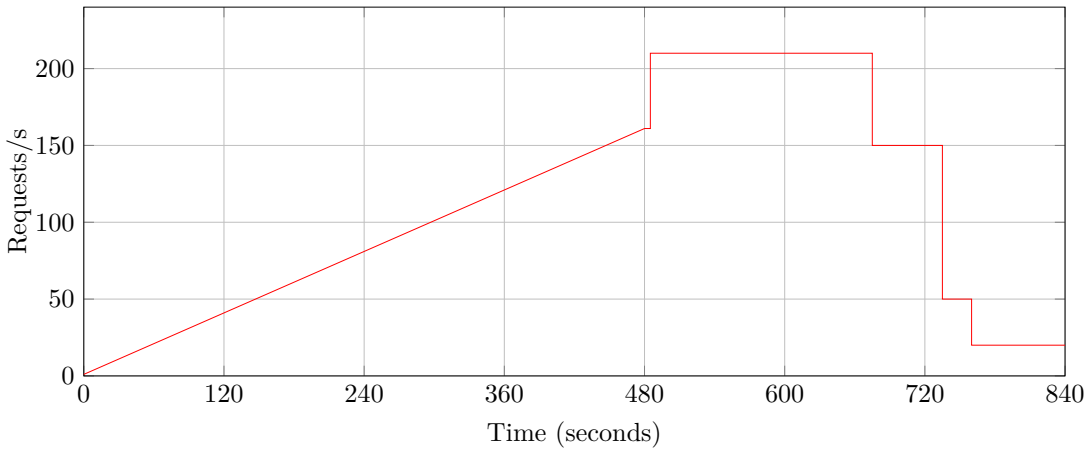


Figure 4.6: Workload used in the experiment.

Figure 4.6 illustrates the workload generated for the safety edge application and used in the experiment. First, the workload increases linearly every three seconds. Afterward,

we stress-test our approach by creating concurrent requests (i.e., up to 30 requests/s) and examine the front-end component behavior during its runtime. Such a workload is generated by new edge devices that use the service. For instance, referring to Figure 4.6, when the component receives up to 50 requests per second, the component may require to scale out to operate accurately since it may utilize the CPU more than 80%. Or, when the component receives less than 50 requests per second, it may need to scale-in to not overuse resources. Generally speaking, edge applications or their software components may experience various workloads over time (i.e., periodically, continuously, or unpredictable). The given workload is just an example used for testing purposes to show our approach's goal to automatically control edge application behavior in elasticity space. To enable elastic behavior for the front-end component, we define elastic requirements in Listing 4.2.

```
#ComponentLevel.EdgeApp01.Front_End
Co1: CONSTRAINT cpuUsage > 75
Co2: CONSTRAINT cpuUsage < 30 AND memUsage < 30
Co3: CONSTRAINT averageRes > 100
Mo1: MONITORING cost = cost.instant
Mo2: MONITORING cpuUsage = cpu.usage
Mo3: MONITORING memUsage = mem.Usage
Mo4: MONITORING averageRes = IO.response
St1: STRATEGY CASE Violated(Co1): ScaleOut
St2: STRATEGY CASE Violated(Co2): ScaleIn
St3: STRATEGY CASE Violated(Co3): ScaleOut
PR1: Priority(Co3) > Priority(Co2)
PR2: Priority(Co3) > Priority(Co1)
```

Listing 4.2: An example of elastic requirements: Front-end component.

Elastic requirements given in Listing 4.2 define the elastic behavior of the front-end component. Strategy (St3) states that if the average response latency is higher than 100 ms, the component should scale-out to ensure the service's quality. When Co1 or Co2 are violated, strategies St1 or St2 enforce specified actions to keep resource utilization in acceptable ranges. As can be noted, the specified metrics are monitored continuously for the front-end component. Furthermore, each constraint may have various priorities. For instance, no matter how much the CPU is utilized, the front-end component must scale if the provided service has high latency than a specified threshold. To that end, if both constraints are violated, the Co3 is enforced since it is prioritized over Co2. Similarly, Co3 is enforced first since it is prioritized over Co1.

The first graph of Figure 4.7 shows the CPU utilization by the front-end component under the given workload (see Figure 4.6). The second graph of Figure 4.7 shows the

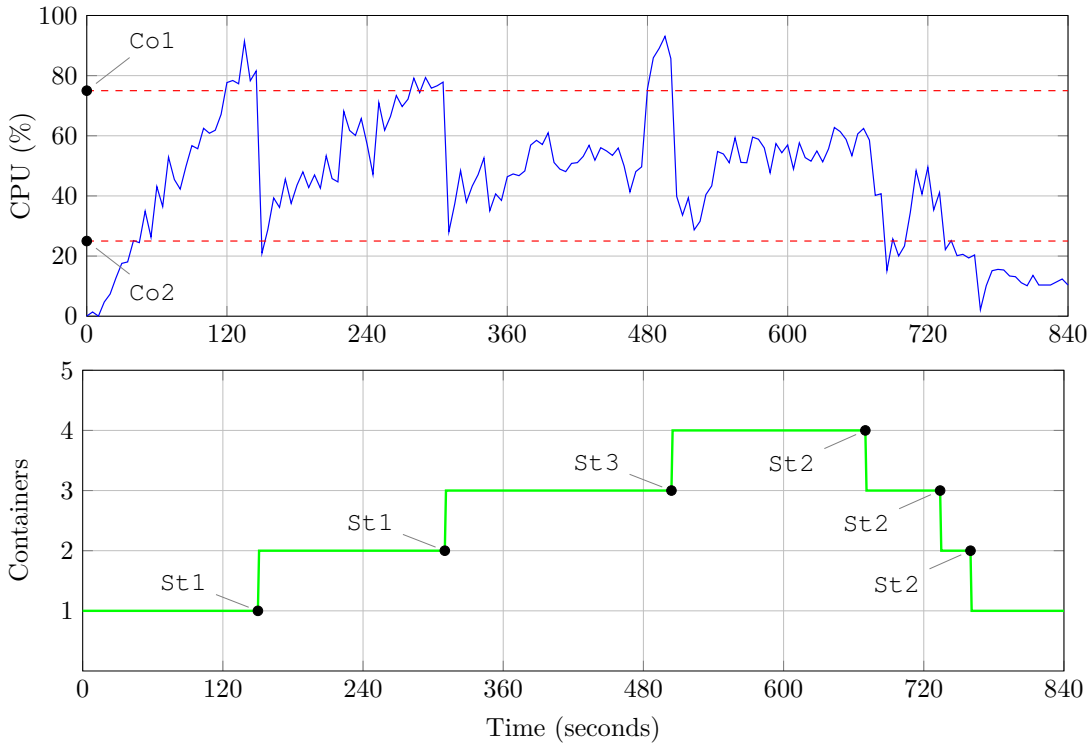


Figure 4.7: The CPU utilization and adaptation process.

front-end component adaption process in response to the workload. As noted, whenever elastic constraints (i.e., Co1 and Co2) are violated, the front-end component scales up or scales down. The adaption process occurs automatically in response to the current workload. The front-end component scales on multiple instances (i.e., containers) to provide the desired service quality. As can be noted, even with continually changing the workload of the front-end component, the CPU utilization remains between elastic boundaries. This ensures that the desired service quality is always guaranteed. The other important aspect is to overcome resource over-provisioning. As can be noted, when the front-end component's workload decreases, the container number is decreased as well (see the second graph of Figure 4.7). Besides, the front-end component's memory utilization remained within elastic boundaries and didn't violate elastic constraints. The CPU of an edge device may fluctuate up and down very quickly due to various workloads. This may cause undesired scaling operations for the same workload. Thus, specified metrics are monitored for five seconds to overcome the mentioned problem. The scaling operation is enforced if the mean value violates elastic constraints. Furthermore, Figure 4.8 shows the front-end component latency over time and the elastic constraint Co3 violation. However, in this situation, both Co1 and Co3 constraints are violated. As noted from the elastic requirement, the Co3 constraint is prioritized over the Co1. In this case, the strategy St3 will be enforced to keep the latency within the elastic boundary.

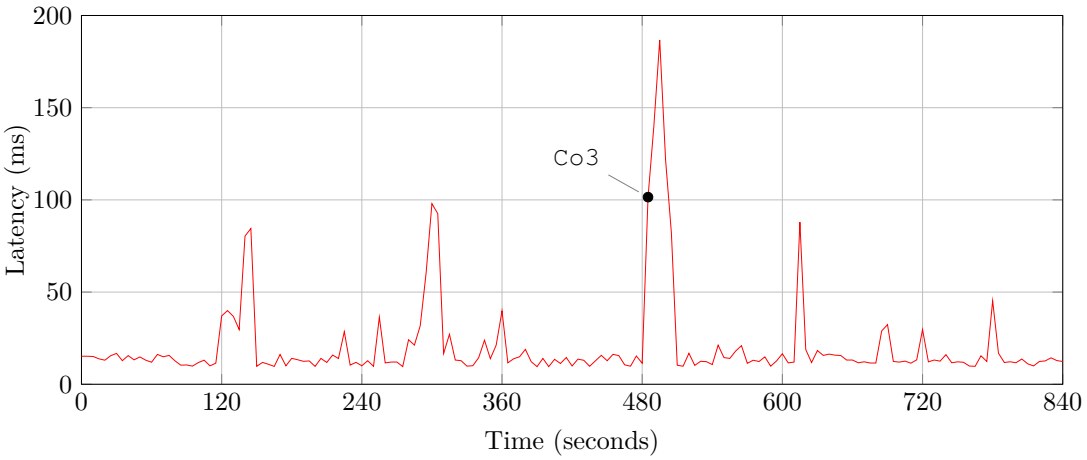


Figure 4.8: Front-end component latency and adaptation process.

A significant challenge remains the time required to start containerized components on low-powered edge devices. In our case, starting safety edge application components takes between 20 - 30 seconds. After the scaling operation is enforced, our approach checks and waits for whether a container is started or shut down. Thus, we avoid undesired situations such as enforcing multiple scaling operations for the same workload. Contrary to the starting operation, the shutdown process occurs in a few seconds for all containers. Nevertheless, edge application components can scale vertically and horizontally depending on the available resources at the edge. The application runtime platform (i.e., Docker Engine) manages this process and scales components within the list of eligible edge devices generated by the DP planner.

In Figure 4.9, we show the evolution of the front-end component in the three-dimensional space (cost, quality, and resources). The quality refers to the latency, the resources refer to the allocated CPU (i.e., edge devices), and the cost is estimated based on the resource allocated. In this case, the cost value is an assumption made to simulate the price paid for resource usage. As can be noted from Figure 4.9, when the service quality decreases (i.e., starting point with green dots), the front-end component scales by increasing the number of resources used as well as the cost is increased. Furthermore, the edge application scales down when the service is not used (i.e., red dot). To that end, such an approach guarantees to meet edge application resource demands at runtime. Other edge application components evolve in the elasticity space based on their load during the runtime. Similarly, the configurator device monitors elastic requirements specified at the edge application level. Thus, the configurator device considers the overall resource consumption of edge application components. For instance, the user may specify that a particular edge application cannot use more than 50% of available resources at the edge.

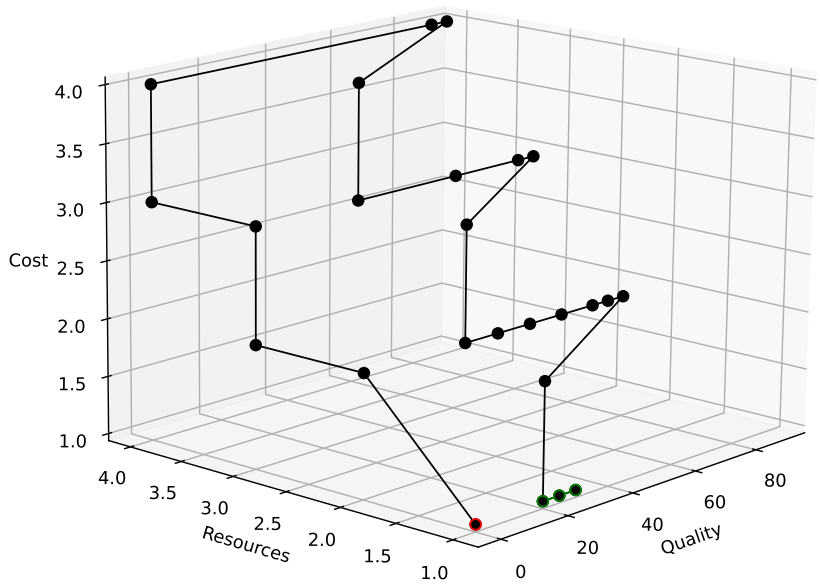


Figure 4.9: Evolution of front-end component in the elasticity space.

4.6.3 Discussion

We have demonstrated through a running example that automatic scaling of edge applications is easily achieved in the edge infrastructure with low-powered devices by using DECENT. Furthermore, we showed that our approach helps avoid highly undesirable situations, such as resource over-provisioning. This ensures that the available resources are used whenever edge applications need them. Nevertheless, the elasticity features play a key role in avoiding edge device failures due to resource over-utilization. For instance, a low-powered edge device can easily fail when an edge application or a software component fully utilizes device resources. Thus, specifying elastic requirements and the DECENT mechanism helps avoid overloading edge devices.

Several assumptions inherent in our approach must be further investigated. In the current prototype, elastic constraints do not conflict with each other. We focus on developing novel constraints and enforcement strategies related to these applications. Simplifying the development process of elastic specifications is among the future works we plan to do. Thus, we plan to integrate the language into an IDE such as c-Eclipse. The c-Eclipse framework provides a user-centric interface through which developers can describe their applications for deployment over edge and cloud. The language integrated into a development environment will make it easy for users to develop elastic specifications and specify correct values to avoid the wrong configuration. Nevertheless, the following tool will also help detect conflicting constraints that the user may select. Nonetheless, we acknowledge that the user may specify conflicting elastic constraints, and thus, we plan to investigate various techniques that would help identify and avoid such situations.

Within this thesis, our primary focus resided on enabling elasticity features at the edge infrastructure; thus, we consider only edge applications where all components are deployed on the edge (i.e., everything on the edge model). Accordingly, adding cloud or fog devices will expand overall available resources and allow executing application components (i.e., containers) in these environments when insufficient resources are at the edge. In future work, we will investigate performance aspects when moving edge application components in large-scale Edge-Cloud infrastructures and controlling their elasticity from the edge.

4.7 Summary

Satisfying dynamic and stringent requirements of edge applications has become challenging for resource-constrained edge networks. Even though edge applications can be modeled as multi-components, dynamic workloads may cause unexpected latencies higher than the expected response time between application components, IoT devices, and end-users. We proposed an efficient solution that simplifies the deployment process and enables elasticity controlling in edge applications deployed in Edge-Cloud infrastructure to overcome such challenges. The developer and user can characterize edge applications by specifying elasticity requirements captured and interpreted by DECENT. The DECENT runtime mechanism then performs complex elasticity controls at the edge of a network.

Edge networks can be different in size and setting; thus, the proposed system is configurable by the system designer. In this thesis, we consider edge networks as resource-constrained environments comprised of low-powered edge devices. The experiments conducted in a realistic testbed showed the feasibility of executing elastic features on low-powered edge devices and adapting edge application components at runtime at the edge. Furthermore, edge applications are executed in a runtime that considers the heterogeneity of edge resources. The proposed framework automatically re-configures edge applications to meet their specified elastic requirements.

CHAPTER 5

Dependable Resource Coordination on the Edge at Runtime

Software components within heterogeneous devices of Internet of Things (IoT) systems, use resources with various computational capabilities, including sensing or actuation endpoints. However, components do not live in isolation and must be able to coordinate with others to fulfill their goals. Satisfaction of requirements must persist in environments that are changing, unpredictable, and potentially unknown at system design time. Edge computers placed near IoT devices can be leveraged for this sort of control – providing resource management for end devices within their operational context. We propose a methodology and technical framework for engineering resource coordination at runtime, tailored for decentralized, pervasive systems of today. Our approach represents a paradigm shift in marrying distributed systems and formal aspects of software engineering. We adopt goal modeling to capture objectives within the system and use bounded model checking as the foundational technique to compute coordination plans which satisfy device goals. This occurs opportunistically at runtime without any knowledge about the operational status or presence of resources, but always in accordance to the edge's own goals. Our technical framework exhibits dependability guarantees regarding optimality and correctness of generated plans. We evaluate resource coordination performance and realizability on low-powered ARM-based edge devices.

The rest of the chapter is structured as follows. After a motivating example described in Section 2.3.3, Section 5.2 gives an overview of our approach within edge computing. Section 5.3 describes key modeling and methodological aspects, goal and modeling of which are expanded on Section 5.4. Subsequently, Section 5.5 illustrates bounded model checking for resource coordination. Section 5.6 provides an assessment of the realizability

of the proposed approach; related work is considered in Section 5.7, and Section 5.8 summarizes the chapter.

5.1 Introduction

Internet of Things (IoT) systems integrate heterogeneous devices, computing infrastructure, and cloud services with their ambient environments. New challenges and opportunities arise as rapidly growing cloud computing, mobile devices, sensors, and networks constitute larger ensembles of systems [DGC+16]. A neighbourhood, for example, may be saturated with hundreds of networked devices providing information to roaming humans, by combining information as they become available in the city environment. Dynamic resource management [GBMP13] is essential to achieving such pervasive behavior – it enables devices and services making up the Internet of Things (IoT) to perceive available resources, configure them and utilize them. Such resources may refer to computational, sensing or other types of domain-specific resources that software-intensive devices may take advantage of to achieve their goals.

IoT applications differ in type and complexity, with multiple system components deployed in diverse domains and environments which are often not known beforehand. Moreover, software-intensive components within devices making up the system, do not live in isolation and must be able to coordinate with others to fulfill application requirements [LC88]. Correct satisfaction of requirements must persist in environments that are changing, unpredictable, and potentially unknown at system design time.

Resources within IoT applications can have various computational capabilities, including sensing or actuation endpoints, storage or processing facilities. Often, those are architecturally abstracted as software services [MLM+06], referring to some functionality that different client IoT devices can reuse for different purposes. The concept of an IoT resource amounts to blurring the lines between software services and sensor values or actuation endpoints.

We are not concerned with mechanisms of accessing resources, their interfaces or policies here, but with the fact that different interdependent resources may be required to fulfill some objective of a software component residing in a device. Dependencies may be in the form of certain constraints – a resource to be operationalized may require the output of another, but its availability makes other resources additionally available. Dependencies may be specified in an implementation- and language-agnostic manner, and annotated over arbitrary resources that an application may use. We adopt an everything-as-a-service (XaaS) abstraction to uniformly represent physical things, hardware and software resources as microservices, irrespective of their specific nature [BSH+17, GTK+10, JCJL14].

Recent developments within distributed systems have led to the the architectural placement of a computing entity closer to the network *edge*, close to IoT end-devices, thus better satisfying system-wide goals such as high availability, performance, or privacy. Such edge entities may offer computation and control to local devices [RGXZ17]. Within

a neighborhood, for example, IoT devices may utilize resources of a local edge node benefiting from high connectivity to it as well as its awareness of other IoT devices in its scope. This allows an edge device to act as a mediator among devices, locally coordinating them in order to satisfy their resource needs. We build on the foundational edge concept, where edge computers are placed near IoT devices, within their local administrative domain or wireless network. We further advocate decentralization, as the edge is a *first-class entity* in our approach, responsible for IoT devices within its scope but bearing no dependencies for coordination to other edge nodes or the cloud. We recognize that *edge computing* means different things to different people; we identify an edge node as a low-powered computer part of an IoT deployment. The edge node is in the scope of connected devices whose software stacks are limited. We consider low-powered edge devices which are ARM-based, and resource constrained IoT devices such as microcontrollers populating the system as is the case in deployments of networked actuators and sensors in smart cities.

In this thesis, we propose a methodology and technical framework for engineering resource coordination for edge-enabled IoT. Our coordination approach targets decentralized multi-edge systems, where IoT devices advertise and request resources at their local edge node. If an IoT device requests an edge node for a resource objective which cannot be trivially obtained from resources readily available, some combination of resources must be derived. To solve this, coordination on the part of the edge node computes a plan, which the requesting device can use to fulfill its goal, in accordance to the edge's goals. Our concrete contributions are as follows.

- We provide a methodology where semantic annotations are specified at design time to arbitrary resources within an IoT system. Those record what a resource requires to be operational, and what effects its potential operationalization has on other resources or context values. Subsequently, objectives of entities within the IoT system are specified – from a requirements engineering perspective, our approach is *goal-driven* since we use edge and device goals to drive coordination of resources at runtime.
- When the IoT system is operational, a SAT/SMT solver situated on a low-powered edge device, leverages bounded model checking techniques at runtime to fulfill objectives of local IoT devices by coordinating available resources in its scope based on the active context.

We instrument coordination as a form of service composition [RS04], but tailored for the IoT. While building upon the significant state of the art of traditional service composition, our resource coordination technique differs for three key reasons:

1. We consider elementary IoT resources as microservices – instead of using a service description language [ABH+02, CDK+02, MVH+04], we adopt a lightweight

approach suitable for microservices inherent in modern IoT architectures and applications.

2. We allow quantifiers and integer linear arithmetic for specification due to the IoT domain, and

3. We target low-powered ARM-based edge computers for deployment.

Our approach represents a paradigm shift in marrying distributed systems and formal aspects of software engineering. Specifically, we adopt goal modeling to model objectives within IoT. We use bounded model checking as the foundational technique to compute coordination plans which satisfy device or edge goals. This occurs opportunistically at runtime, without any knowledge about the operational status of the system or which resources are present at the system's design time. The resource coordination facilities we provide are dependable because if there is a solution to a resource coordination problem for a device, the technique we utilize will provide a plan for it, and the plan will be optimal. This is in contrast to other approaches utilizing other coordination techniques such as based on AI [ASD18a, HN01, GNT04]. We acknowledge that the technique we adopt is computationally expensive, but offers dependability guarantees. To this end, our evaluation targets resource coordination performance and realizability on low-powered ARM-based edge devices.

5.2 Coordination at Runtime on the Edge

IoT applications can be of various types, software stacks, and complexities, with multiple system components deployed in diverse domains and contexts. Those, however, do not live in isolation and must be able to coordinate to fulfill application or end user requirements [LC88]. A software component hosted on some device for instance, may require a reading from a sensing endpoint in order to perform some computation and fulfill its objective. This problem is exacerbated within IoT deployments, as applications need to operate on diverse infrastructures and integrate heterogeneous components from various providers in a long-running system, with possibly conflicting goals between components.

5.2.1 IoT Resources, Services and Goals

IoT software components provide data, sensing and actuation, as well as computational resources to other software components, which can be abstractly represented with the concept of an IoT *resource* [JCJL14] Within an IoT system, components can have different software stacks but still interact – this is widely achieved by software services, the architectural abstraction permeating many systems today [Erl05]. A systems' development is then based on writing custom business logic which utilizes services. As IoT components are resource-constrained, services often take the form of loosely coupled microservices, communicating with lightweight methods. Examples of this are typically found as sensing or actuation endpoints – a temperature sensor responds with a temperature value

when invoked for instance, or a smart door unlocks a door when the security system requires it to. Such functionalities may be abstracted as resource microservices, that the software-enabled devices make available over their local scope such as a wireless network.

Resources that an IoT device may need from others (nearby), need to be appropriately composed and communicated to the device, in order for it to fulfill its objective. For example, computing a local weather forecast (i.e., an objective) in a smart agriculture setting may require temperature readings (i.e., resources) from available devices across a crop field. In the general sense, resources available within an IoT scope –some local context– need to be coordinated, with the IoT device's goal in mind. This coordination essentially amounts to *planning* as understood within self-adaptive systems: actively setting in motion configuration changes to satisfy certain objectives, in this case the goal of an IoT device which depends on other IoT devices' resources in its local scope. Satisfying an IoT component's goal however is challenging, as devices are deployed in changing and unpredictable (i.e., at design time) environments. Assumptions made at system design time about the availability or location of resources that a device needs may be violated. Thus, facilities providing control and coordination must be performed at runtime and based on the current environmental configuration, by ensuring that the IoT system can autonomously react to changes in different contexts in a dependable manner.

5.2.2 Coordinating Resources on the Edge

Centralizing computation of coordination – typically in the Cloud and evident in today's IoT-Cloud architectures is one solution but requires cloud control structures to be always available and within low latency. However, novel functional and non-functional requirements that have arisen in IoT systems dictate computation and control to be situated locally near devices. We advocate that since the edge computing entity is closer to end devices (and IoT application users), there is an opportunity for situating coordination there – something realized by empowering an edge computer to actively coordinate resources of IoT devices within its scope. We note that this fits the domain particularly well; IoT devices are found within a local scope, such as a local wireless network or a deployment within a limited geographical region (e.g., a city neighborhood). As such, placing an edge computing entity close to a set of locally-scoped devices providing coordination facilities is highly feasible.

Distributed systems mechanisms relevant to process coordination and control, such as service engineering and resource management must be adopted to identify, discover IoT resources. Methods developed within formal aspects of software engineering, such as requirements reasoning, model-driven planning and self-adaptive systems are then adopted in our approach to enable coordination of available resources at runtime. Models kept at runtime, facilitate coordination and the determination of how control actions can satisfy goals within the system. Regarding architectural deployment, the edge is a first-class entity in our approach, acting as a manifestation of a control agent responsible for receiving IoT device resource requests, observing contextual information and inducing appropriate actions to satisfy them.

5.2.3 Instrumenting Coordination

Figure 5.1 provides a birds-eye view of our approach to coordination at runtime on the edge. The capabilities offered by IoT devices' software stacks are abstracted as resources (i.e., microservices) and made available through the network, in accordance to the state-of-the-art. In our approach, those are specified at design time for each participating software-enabled device, together with possible goals that the device may seek to achieve (1). The resource configuration of the system, as well as the environment that the system may be found when operational is unknown at design time. At runtime, IoT devices are found within some local scope, and may interact with others to utilize their resources. However, device goals may have interdependent requirements on other resources, so coordination is required to fulfill a requesting device's functionality. An edge node, situated close to the IoT end-devices and managing the local IoT scope is responsible for coordinating available resources, at runtime (2). Participating devices then interact according to the generated coordination plans such that their goals are achieved (3).

Resource coordination occurs opportunistically at runtime, for which we propose a technique based on bounded model checking, offering guarantees of correctness and optimality of the generated plan which satisfies a requesting IoT device's goal. The resource coordination we propose extends traditional service composition [SQV+14] and brings it into the IoT context; (i) we adopt a lightweight approach suitable for resource-constrained IoT microservices, (ii) we allow quantifiers and integer linear arithmetic for specification, and (iii) we target low-powered ARM-based edge computers for deployment.

5.3 Domain Modelling and Methodology

In this section, we provide basic abstractions and methodological principles necessary to instrument coordination within an IoT domain we are situated in. For formalization purposes, we assume a global set of names or key-value pairs Π that appear throughout the system[1]. We begin by outlining key elements and assumptions of our approach, upon which we define a methodology that the system designer follows to instrument resource coordination at the edge.

IoT Resource. Architecturally, IoT devices are software components deployed in different environments, each containing resources. Generally, [BSH+17, GTK+10, JCJL14] we assume that an IoT system is architecturally composed of processes which are microservices. Such IoT microservices, when invoked yield resources; however, successful invocation entails meeting the requirements of a microservice, which may depend on others. We will refer to microservices and the resources that they yield interchangeably. This may occur for several sequences of resource invocations, thus motivating the need for coordination; knowledge of which resources are needed to operationalize a resource

[1] Without loss of generality, we take Π to be comprised of atomic propositions, essentially mapping identifiers and their values to true statements.

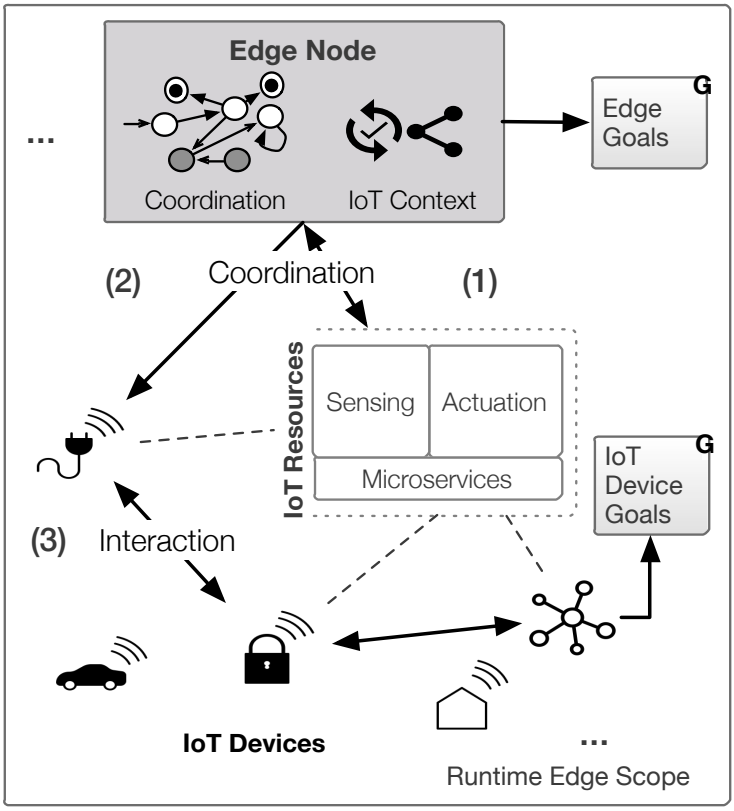


Figure 5.1: Runtime Resource Coordination on the Edge: Overview.

that an IoT device requires entails coordination. IoT resources within our approach are implementation- and language- agnostic; what we require is modeling of their pre-conditions (i.e., what they *require* to be activated), and their post-conditions (i.e. how their successful activation changes some context).

For our example, we assume that three exemplar resources are present in the IoT system of the city, highlighting different modelling aspects. A recycling truck has a recycling_truck resource, which empties waste cans in a neighborhood when it is present. A smart traffic_light ensures that municipal vehicles –such as the recycling truck– are met with green lights, and an irrigation actuator in a park is responsible for watering it when required. In depth treatment of resources will be described in Section 5.3.

Runtime Edge Context. In edge computing architectures, IoT end-devices interact with their environment, where the edge device is by definition located within the local domain of certain IoT devices – one can take that as the devices being in the logical scope of the local edge node. The status of various devices, resources as well as environmental information observed during system operation and that the edge node is aware of is referred to as the *runtime edge context*. Edge context is assumed to be local to some edge node (Figure 5.1). We identify as $\mathsf{C} \subseteq \Pi$ the runtime edge context, comprising of a set of

key-value pairs within an edge scope. A valuation of C refers to a specific moment in time – key-value pairs reflect the runtime physical or logical environment. While keys are unique identifiers, the domain of their values can range. Specifically, we allow booleans, a domain of a finite set, or arbitrary integers. Valuation may change because of monitored information (i.e., resulting in a change to a sensor value) or due to exogenous to the system stimuli. However, it may also change due to resources of IoT devices (i) if a resource is made available to others, this is reflected in C, and (ii) if it is operationalized, it may change values it its context. We assume appropriate instrumentation for correct accounting of the various values within the edge context.

For our example, neighborhoods and parks are edge scopes, in each of which an edge node is placed, accounting for the current context and IoT devices that may be nearby, and as we will observe later, coordinate their resources. We assume that waste_cans and traffic_light are two identifiers within the edge context of a neighborhood, values of which are populated by sensors in the waste cans and a traffic light IoT device, respectively (Formula 5.1). In the park, we assume that a sensor detects soil_moisture and another senses if the park is crowded. (Formula 5.2). Within a neighborhood context, we can observe different domains for values of its identifiers: while waste_cans can be true or false, a traffic_light can be either green or red. Differently, soil moisture in a park can be some integer value:

$$\mathsf{C}_{\mathsf{neighd}} = \Big\{\mathsf{waste\_cans} : \mathsf{bool}, \mathsf{traffic\_light} : \{\mathsf{green}|\mathsf{red}\}\Big\}; \tag{5.1}$$

$$\mathsf{C}_{\mathsf{park}} = \Big\{\mathsf{crowded} : \mathsf{bool}, \mathsf{soil\_moisture} : \mathsf{int}\Big\}. \tag{5.2}$$

IoT/Edge Goal: An objective which an IoT device or edge node seeks to achieve – satisfaction of which may depend on available resources at its local environment. Goals capture, at different levels of abstraction, the various objectives entities within the IoT system under consideration should achieve, or constraints of various context values within their control. A goal for an IoT or edge device is a logical formula over the set Π.

Back to our running example (shaded boxes within Figure 2.4), parks should be watered but this should not occur when they are crowded with people – this entails an objective of the edge node placed in the park. Similarly, a neighborhood edge node should ensure that if municipal vehicles are present, it should be ensured that green traffic lights allow them to pass. The overall city or district containing parks and neighborhoods, imposes constraints, such as water management. Such goals will be modeled precisely in Section 5.3.

Design Time Methodology. Our approach entails engineering resource coordination tailored to IoT systems, and it methodologically spans both design-time and runtime. Illustrated in Figure 5.2 by leveraging design time specifications, coordination is enabled at runtime, by the following steps:

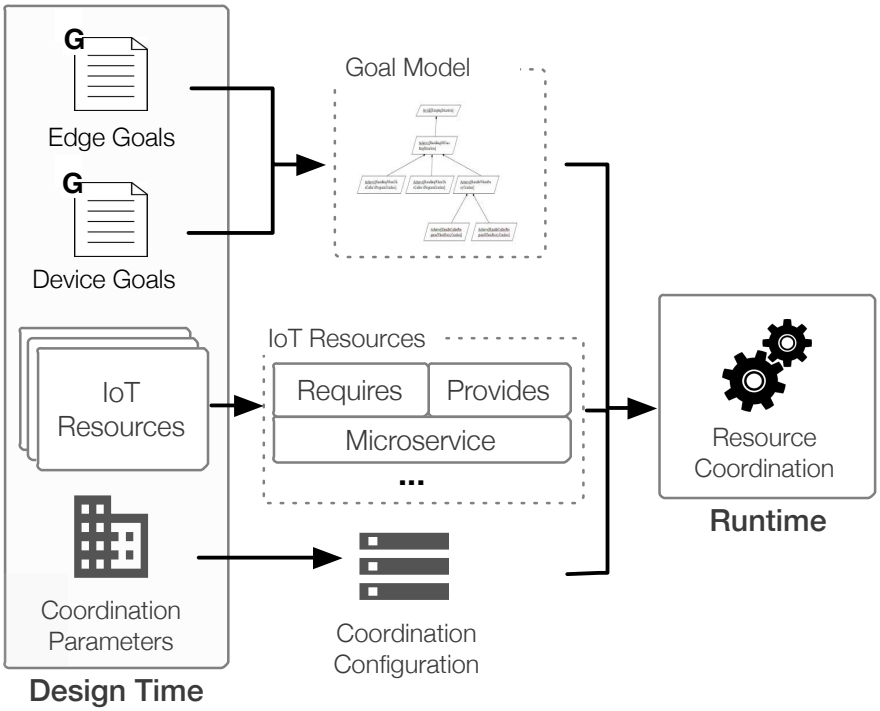


Figure 5.2: Engineering Coordination: Design time Methodology.

1. *Modelling IoT resources.* Language-agnostic, semantic annotations to arbitrary IoT resources of participating devices are specified. Such annotations record what a resource requires to be operational, and what effects its invocation has on other resources or context variables. This step will be described in Section 5.4.1.

2. *Modelling Device and Edge Goals.* Objectives of the various entities active within the system are captured in a goal model, which facilitates goal refinement, resolution of conflicts and goal interdependencies. This step will be described in Section 5.4.2.

3. *Specification of runtime coordination parameters.* The technique employed at runtime to satisfy resource requests from IoT devices, is deployed on a resource-constrained edge device. As such, depending on the particular deployment setting, coordination parameters regarding performance aspects are specified per the overall application requirements. The coordination technique is described in Section 5.5, while useful insights about parameterization will be illustrated in Section 5.6.

A further necessary step –out of scope of thesis – are application-specific as well as architectural deployment aspects. For a complete instrumentation of coordination, edge nodes must be deployed and be responsible for certain scopes (e.g. neighborhoods in a smart city). Communication and network management must be handled as well. Overall, we consider such aspects as orthogonal to our approach, and we assume they are in place.

5.4 Resources and Goals within IoT

Resources in an IoT context may be arbitrary, provided by heterogeneous software components deployed on devices from various vendors and architectural stacks. Within the overall IoT collective, IoT devices and edge nodes alike, may have objectives that they seek to achieve. In this section, we firstly describe how IoT resources can be generally represented, particularly with respect to what they require to be operationalized, and what their invocation entails for others. Secondly, we adopt requirements engineering methods to capture objectives throughout the system by goal modeling.

5.4.1 Modeling Resources within IoT

To model resources within an IoT system, we advocate the principle of *procedural abstraction*: capturing knowledge of IoT process internals is impractical in practice, but considering the requirements and effects of IoT processes is feasible. For example, one does not need to know how a sensor array calculates mean temperature based on a spatial dispersion of IoT sensors, but only that by invoking some software service, the current average temperature is obtained. To this end, as noted in Section 5.3, we assume that a software-intensive IoT system is architecturally composed of processes which are microservices. Such IoT microservices, once invoked, yield resources; however, successful invocation entails meeting the requirements of a microservice, which in turn may depend on others. This is where the use of pre-conditions and post-conditions is beneficial, which we informally refer to as what an IoT microservice *requires*, and what it *provides*. Pre- and post-conditions are first-order logical formulae as parameters, with propositions in set Π. Quantifiers (over finite sets) and integer linear arithmetic may be used for specification:

- *Requires* is a pre-condition directive that outlines what conditions should be true in a given context for a resource to become operational, essentially its requirements.
- *Provides* refers to a post-condition directive that outlines what conditions are true as a result of an operationalization of a resource.

More formally, a resource is a tuple $\lambda =< \mathsf{R}_\lambda, \mathsf{P}_\lambda >$ where R_λ and P_λ are first-order formulae of input and output parameters, respectively. Parameters themselves are sourced from the global set of propositions Π, and without loss of generality, are assumed to be key-value pairs. We slightly abuse notation and refer to $\mathsf{parameter}$ if $\mathsf{parameter} = \top$. Given the above, when a resource λ is invoked with input R_λ, λ returns output $\mathsf{p} \in \mathsf{P}_\lambda$. Requires and provides directives are specified at the system's design time, for every resource. For presentation purposes, we will write $[\mathsf{R}_\lambda]\ \lambda\ [\mathsf{P}_\lambda]$ and $\lambda =< \mathsf{R}_\lambda, \mathsf{P}_\lambda >$ interchangeably.

$$\big[\mathsf{municipal\_vehicle}\big]\ \mathsf{allow\_vehicle}\ \big[\mathsf{traffic\_light} = \mathsf{green}\big]. \tag{5.3}$$

Given the above, we can model the resources of our smart city example. Recall that traffic lights deployed facilitate municipal vehicles such as ambulances or recycling trucks, so that they are presented with green traffic lights. This traffic light functionality can be represented as a resource (i.e., that a traffic light IoT device offers), setting the traffic light to green when applicable. Formula 5.3 intuitively states that when a municipal_vehicle context value is true, the light turns green. The context value within $\mathsf{C}_{\mathsf{neighd}}$. is assumed to be set by e.g. a truck when it is within the local scope of the traffic light. The functionality of a recycling truck (i.e., as an IoT device) can be similarly represented in Formula 5.4.

$$\begin{bmatrix}\mathsf{traffic\_light} = \mathsf{green} \\ \wedge\ \mathsf{waste\_cans} = \mathsf{full}\end{bmatrix}\mathsf{recycling\_truck}[\mathsf{waste\_cans} = \mathsf{empty}]. \tag{5.4}$$

Quantitative values can also be captured in resource parameters – for this purpose, we support integer linear arithmetic. For example, the irrigation resource present in the park (i.e., due to an IoT device responsible for irrigation), should be activated when detected soil moisture is below a certain threshold and when the park is not crowded. The irrigation resource can then be modeled as in Formula 5.5. When some context value crowded is false and some other soil_moisture is less than 20, irrigation – if activated – will result in setting the latter to 20; those refer to $\mathsf{C}_{\mathsf{park}}$.

$$\left[\begin{array}{l}\neg\ \mathsf{crowded} \wedge \\ \mathsf{soil\_moisture} < 20\end{array}\right]\ \mathsf{irrigation} \qquad \left[\ \mathsf{soil\_moisture} = 20\ \right]. \tag{5.5}$$

In general, resource models in IoT [DBBM11] are widely established in literature and can be utilized to model resources as the formulae tuples $< R_\lambda, P_\lambda >$ we advocate, since the model is quite generic. Within parameters, propositions of $< R_\lambda, P_\lambda >$ formulae can include (i) location, describing the logical physical domain where a resource resides, (ii) administrative domain, describing a repository permitting authentication or authorization, (iii) type, characterizing a resource instance as a sensor, actuator or a logical entity, or (iv) capability, providing special abilities that a resource enjoys, based on some domain ontology. All of those, including quantitative cases, can be encoded as $< R_\lambda, P_\lambda >$ parameters and exposed in the namespace of an edge context using the method previously described.

5.4.2 Modelling Goals within IoT

Recall that both IoT devices as well as edge nodes may have goals; in our context, goals are objectives within the system that various entities seek to achieve. The role of the edge is to facilitate goal achievement for devices within its scope. Depending on the general state of the system, other edge nodes' goals may be affected in turn.

Goal-oriented modeling, widely used in requirements engineering [FFvLP98, vL09a, CSBW09, LMSM10], is a technique that can capture, clarify, and enable analysis of

system requirements. The structured form of a goal model allows refinement of system goals to subgoals, prioritization of requirements, as well as resolution of inconsistencies that may be due to conflicting stakeholder viewpoints. Our intuition to model goals in the IoT-edge context is that edge nodes in IoT systems are often arranged in a hierarchy, where the cloud is the global or root entity representing the whole system, and edge computers are found within that hierarchy with IoT devices as end nodes. Notice that this structure is reflected in our running example, where a district may contain multiple neighborhoods and parks each having a logical edge node. IoT devices are within the scope of an edge entity leading to a tree arrangement where IoT devices are leaves.

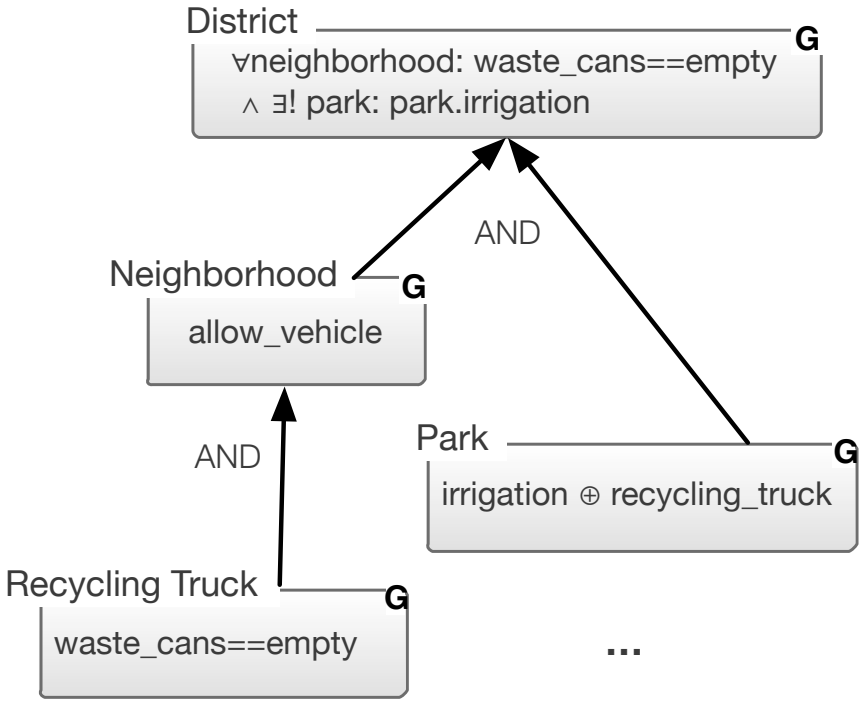


Figure 5.3: Goal model capturing objectives of edge and IoT devices.

In our approach, we adopt a form of discrete goal modeling to capture objectives of devices, edge nodes as well as their relationships[2]. As shown in the goal model of Figure 5.3 which encodes system concerns of the running example, each goal can be refined into subgoals through an *and-or* decomposition. At the leaf-level of the goal model reside IoT device goals – each leaf describes an objective of an IoT device. As with specification of pre- and post-conditions, a goal for an edge device is an arbitrary first-order logical formula E over the set of global set of names Π; for an IoT device, we denote its goal as G. Quantifiers (over finite sets) and integer linear arithmetic may be used for specification.

[2]Note that weights can be assigned to edge nodes if quantitative priorities are desired, leading to a *weighted goal model* [VL09b] and further constraints.

Goal modeling is used to (i) provide an indication of satisfaction of the various system objectives, (ii) provide a structured way of managing relationships within the edge-intensive system and (iii) to coordinate appropriately devices within edge scopes. The satisfaction status of each (sub-)goal in the goal model (e.g., Figure 5.3) can be determined by observable run time information or by active operations of IoT devices. The current context status on every edge node captures this. For example, the status of waste cans in a neighborhood can either be monitored (e.g., by sensors in the waste cans) or set to "empty" by a recycling truck. The goal model structure intuitively shows how other sub-goals are affected – observe that if a recycling truck empties the waste cans in a neighborhood, its respective goal will be affected. Values of subgoals are propagated upwards the goal model (i.e., the edge structure), affecting satisfaction of parent goals. Assuming a variable with values of a known finite set of neighborhoods in a district, the district goal states that for every neighborhood, the waste cans should be empty, and that there is exactly one park such that the irrigation is true. Similarly, a park goal captures the fact that irrigation should not be true at the same time where a recycling truck is present in its scope.

Edge goals govern how they coordinate resources for requesting IoT devices. Since the runtime edge context is unknown and coordination occurs at runtime, goal satisfaction happens opportunistically; edge nodes' and devices' goals may be satisfied depending on presence of other devices and runtime context values. This in turn, may affect other sub-goals of the system – for example, if the waste cans are emptied in a neighborhood and this happens for every neighborhood in a city, the district's goal may be satisfied (first clause in the conjunction in Fig. 5.3). On the other hand, goal relationships between edge nodes are not accounted for coordination as this would impair system performance and incur centralization – each edge in a decentralized manner imposes its own goals within its scope, but their resulting satisfaction is propagated to others as sub-goals. Specification of a goal model is left to the system designer, which may specify arbitrary goals for entities in the system.

5.5 Dependable Resource Matchmaking

As we observed, the edge as the coordinator within its active runtime context, receives a request from a device seeking to achieve some goal which depends on other IoT resources or context values. Coordination then amounts to figuring out how to combine available resources or context values to produce a plan, which is then be returned to the device. We call this process *resource matchmaking*, as it entails making a match between the requesting device and other devices, such that their resource combination can achieve a goal. Matchmaking as described is a complex problem as it amounts to NP-completeness; in this section we present the technique we utilize, which results in *dependability* guarantees; solutions are provided always correctly and optimally (if they exist).

To tackle resource matchmaking, after first formally defining the problem we demonstrate

how it can be mapped to a state-transition structure capturing evolution of resources in the system. Subsequently, we reduce the matchmaking problem to reachability within this state-transition structure. Finally, we provide a conjunctive normal form (CNF) encoding of the problem that is suitable as input to a solver, upon which bounded model checking [BCC+99] is used to solve it, yielding an optimal solution.

5.5.1 Resource Matchmaking Problem

Recall that a resource is a tuple $\lambda = < \mathsf{R}, \mathsf{P} >$ where R and P are first-order formulae of input and output parameters, respectively. We assume that when a resource λ is invoked with the input formula R, λ returns output P (i.e. resource microservices work correctly). To decide an invocation relationship from resource $\lambda_1(\mathsf{R}_1, \mathsf{P}_1)$ to $\lambda_2 = (\mathsf{R}_2, \mathsf{P}_2)$, it is necessary to compare outputs P_1 of λ_1 with inputs R_2 of λ_2. To establish a relationship, the requirements R_2 of λ_2 must be met. Generally, given a set of available resources and a request resource λ_{req}, we seek to find a resource λ such that $\mathsf{R}_{\lambda_{\mathsf{req}}} \subseteq \mathsf{P}_\lambda$. However, there might be the case that there is no single resource satisfying the requirement of the requesting resource. In that case, we seek to find a sequence $\lambda_1 \cdots \lambda_k$ of resources where in each step invocation of a resource λ_i occurs and the desired objective is eventually achieved. As there can be many such sequences, the optimal solution for the resource matchmaking problem is to find one with the minimum value for k.

5.5.2 Resource Evolution and Device Goal Reachability

To enable automated reasoning, we represent the evolution of resources in a state-transition system generally known as a (doubly) *Labelled Transition System* (LTS [CGP99]) which is a tuple $\mathcal{K} = (\mathcal{S}, \Pi, \Lambda, \mathcal{L}, \mathcal{A}, \mathcal{I}, \mathcal{G})$ where:

- Π is the global, finite set of atomic propositions,
- $\mathcal{S}$ is a set of states,
- $\mathcal{L} : \mathcal{S} \rightarrow 2^{\Pi}$ is a function that labels each state with the set of propositions Π that are true in that state.
- Λ is a set of transition labels capturing resources,
- $\mathcal{A} \subseteq \mathcal{S} \times \Lambda \times \mathcal{S}$ is a 3-adic accessibility relation. If $p, q \in \mathcal{S}$ and $\alpha \in \Lambda$, then $(p, \alpha, q) \in \mathcal{A}$ is written as $p \xrightarrow{\alpha} q$,
- $\mathcal{I} \in \mathcal{S}$ is an *initial* state and $\mathcal{G} \in \mathcal{S}$ is a device *goal* state.

States of $\mathcal{K}$ capture values (or parameter instantiations) while transitions record how those can change by moving from one state to its successors by operationalizing resources. Each state declaratively represents an instantiation of resources and context values (i.e., of Π) at some moment of time. The accessibility relation $\mathcal{A}$ between states shows how

parameters instantiations and context values change moving from a state s to another s'; it corresponds to the transitions of $\mathcal{K}$. Intuitively, starting from an initial state of the system representing an initial configuration, application of resources λ_i generates states according to their $\mathsf{R_i}$ and $\mathsf{P_i}$ and context values.

Given an incoming request for G from a device, the initial state $\mathcal{I} \in \mathcal{S}$ captures context values at the edge node at the time of the resource request (i.e., at runtime). The goal state $\mathcal{G}$ captures some configuration where G holds. Solving the matchmaking problem as presented, amounts to *reachability* [EMB11] of $\mathcal{G}$ within dLTS $\mathcal{K}$, illustrated in the following.

5.5.3 Resource Matchmaking with Bounded Model Checking

As we observed, given a request for a goal G from a device, the desired outcome for the matchmaking problem is to find a sequence of resource applications which starting from an initial state, bring the system to a state where G is fulfilled. In the following, we show how this reachability problem [EMB11] can be solved with bounded model checking [BCC+99] through an encoding to a CNF formula.

To formalize the reachability problem, it is first necessary to introduce the following definitions. Given that $s_i \in \mathcal{S}, 0 \leq i \leq n$ and $\alpha_i \in \Lambda$, a finite *computation* is defined as a finite composition of transitions:

$$s_0 \xrightarrow{\alpha_1 \cdot \ldots \cdot \alpha_n} s_n =_{def} s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \; s_{n-1} \xrightarrow{\alpha_n} s_n.$$

The concatenation $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_n$ of labels (representing resource invocations) is called a *trace* originating from s_0. The sequence of states $s_1 \cdot \ldots \cdot s_{n-1}$ is called the *sequence of traversed states*. State s_0 is the *originating state* of the sequence and state s_n the *end state*. Reachability entails the existence of a computation $\mathcal{I} \cdot s_1 \cdot \ldots \cdot s_{n-1} \cdot \mathcal{G}$. Each state s_i along the computation captures available values (or resource parameter instantiations) in time instant i. The desired outcome is the respective trace; if the requesting IoT device invokes the resources indicated by the trace in series, it can reach $\mathcal{G}$, where its goal G is fulfilled.

Recall that instantiated parameters and context values as propositions that live on states $\mathcal{S}$ are drawn from set Π, while labels (corresponding to resources) from set Λ; a relation $\mathcal{A}$ has the form $\mathcal{S} \times \Lambda \times \mathcal{S}$. The fundamental intuition to obtaining the trace, is establishing the relation $\mathcal{A}$ that represents accessibility from a state s to its subsequent state – let s' be this subsequent state. To do this, we exploit the fact that a resource application operates on R_λ, in a way that yields P_λ in the next state s'. Let $\mathcal{T}$ be a helper function yielding *true* if a label $\lambda \in \Lambda$ and P_λ can be combined leading to R_λ. We represent as s_Π the propositions describing state $s \in \mathcal{S}$ as a conjunction. E is the formulation of the goal of the edge node where coordination takes place. Establishing $\mathcal{A}$ starting from the initial state $\mathcal{I}$, traversing states of the computation and eventually reaching the device goal state $\mathcal{G}$ amounts to the following formula encoding the computation:

$$(\mathcal{I}_\Pi \wedge \mathsf{E}) \bigwedge_{0 \leq i < k} \mathcal{T}(\mathsf{s}_{\Pi_i \wedge \mathsf{E}}, \Lambda_{i+1}, \mathsf{s}'_{\Pi_{i+1} \wedge \mathsf{E}}) \bigwedge \mathcal{G}_\Pi \wedge \mathsf{E}. \tag{5.6}$$

Formula 5.6 starts with a conjunction of a set of propositions describing the initial state conjuncted with the edge's goals (recall that E itself is a first-order logical formula). Subsequently, it encodes the existence of a computation whose transitions are labeled according to resources Λ. Each state $s \in \mathcal{S}$ of the computation is a conjunction between the edge goal formula E and propositions describing the state. Finally, a goal state ($\mathcal{G}_\Pi$) is reached while maintaining satisfaction of the edge goal E. The edge goal acts as a further constraint on every computation state – it must be always fulfilled as resources are invoked. The device's goal is the final state reached, while the edge's goal as well as the resource invocations govern how it is reached.

Note that the index k represents the length of the trace. Formula 5.6 is true if and only if there exists a computation of length k from state $\mathcal{I}$ to goal state $\mathcal{G}$ of the dLTS $\mathcal{K}$. Notice that the formula is a conjunction of a finite collection of literals, thus in CNF form. Following the definition of Formula 5.6, a SAT/SMT solver can be used to check its satisfiability [BCC+99], for incremental values of n. The smallest n where the formula is satisfied, represents the optimal solution. The respective trace represents the solution sequence of resource invocations.

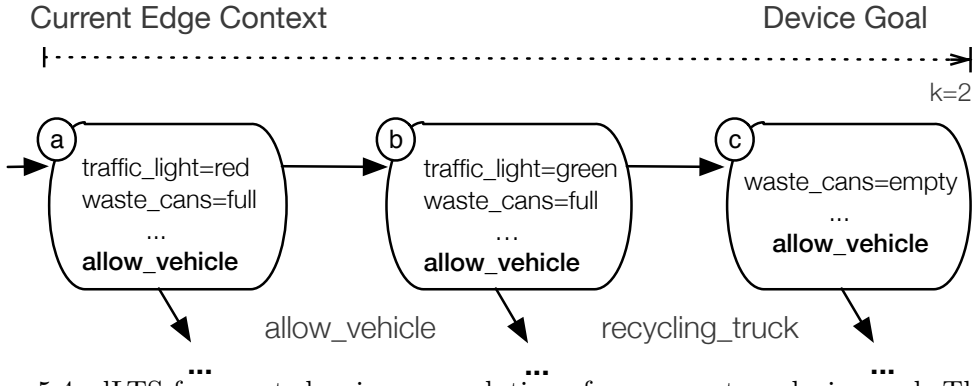


Figure 5.4: dLTS fragment showing an evolution of resources to a device goal. The edge goal is in bold, as a constraint through the states of the computation.

For our resource coordination purposes, we essentially ask for an assignment that satisfies the constraints of each resource, leading to the fulfillment of the device goal. The values of the transitions consist the coordination plan, consisting of the resource invocations that the requesting device must perform to satisfy its goal. Formally, the plan returned is the concatenation $\alpha_1 \cdot \alpha_2 \cdot \ldots \cdot \alpha_n$ of labels (representing resource invocations) amounting to the trace originating from $\mathcal{I}$ and leading to a goal state $\mathcal{G}$ where the device's goal G is satisfied. Certainly, if there exists no satisfiable solution, a plan cannot be computed. If a plan exists however at a minimal length k, there are guarantees about optimality – there

is no plan at length less than k that satisfies the goal. Given a coordination problem, computation of a plan in practice can be achieved by employing a SAT/SMT solver, from which a satisfiable assignment of Formula 5.6 is requested.

Figure 5.4, shows the dLTS corresponding to our example; state (a) captures the current edge context, where a recycling truck arrives. Recall that the goal of the truck is to empty the waste cans, which it communicates to the local neighborhood edge node. The plan computed at the edge node, shows that an invocation of allow_vehicle can enable recycling_truck, which will lead to the satisfaction of the truck's goal. Note how allow_vehicle is present on every computation state as it is a constraint imposed by the edge node.

5.6 Evaluation

For evaluating the proposed approach, we developed tool support and a proof-of-concept implementation based on the CVC4 SMT solver [BCD+11b]. Noting the absence of approaches utilizing SMT solving on the edge, we deployed the prototype on low-powered ARM-based devices representative of edge nodes situated typically close to IoT devices in wide area settings such as smart cities. The technique we advocate for resource matchmaking is based on bounded model checking, a highly computationally expensive operation which is usually performed at design time. However, we bring it to system runtime. To this end, our evaluation goals target realization and feasibility of our approach for coordinating resources at runtime for the edge-enabled IoT. Concretely, we aim to:

- Investigate feasibility over concrete deployment on low-powered, ARM-based edge devices;
- Assess performance of SMT-based matchmaking over hard matchmaking problem instances.

Investigating the feasibility of the approach is crucial for showing the applicability of the proposed framework in real-life scenarios. Therefore, we investigate solving problems with different complexities, which results in different formula sizes, and coordination times required to achieve user or device goals at the edge. Naturally, as problem size increases, so does the formula and the coordination time. We present our evaluation setup on Section 5.6.1, and experimental results obtained in Section 5.6.2. We conclude with a discussion in Section 5.6.3.

5.6.1 Experiment Setup: Synthesized Resources

Our experiment setup entails (i) generating a suitable dataset and (ii) deploying the prototypical framework on low-powered devices which serve as edge nodes. To obtain a

suitable matchmaking dataset for our experiments, we automatically generate problem instances, each containing (i) a set of IoT resource specifications, (ii) some IoT context assumed to be active when the procedure is invoked, and (iii) some resource goal that a device is assumed to have requested. We synthesize matchmaking problem instances, varying the number of resources available, resource pre-conditions, and number of operators among them, given a global set of names Π, where $|\Pi| = 100$. Specifically, our experimental dataset comprises of the specification of a set of randomized problem instances in turn each comprising of:

- A set of resources X assumed to be available within an edge scope. Each is modeled per Section 5.4, as $< R_\lambda, P_\lambda >$. The cardinality of X ranges from 10 to 50, yielding different problem instances.
- For each problem instance I, cardinality of R_{λ_i} sets for every resource $i \in I$ ranges from 5 to 15 of parameters, which are randomly combined with a number of operators: $10 \leq |R_\lambda| \leq 15$. Operators are inserted randomly within R_{λ_i}. Resource post-conditions are a conjunction of five parameters: $|P_\lambda| = 5$. $R_\lambda, P_\lambda \subseteq \Pi$.
- A context description, referring to the set of context values when the edge node initiates the coordination process. We assume a conjunction of $|\mathsf{C}| = 20$ such context values, where $C \subseteq \Pi$.
- A random device goal G, which is a conjunction of five elements of Π.

From the synthesized problem instances, we select ones that are satisfiable, to ensure coverability of the whole process of computing coordination plans presented in Section 5.5, and to reduce noise in the results. Throughout the process, we use boolean operators only, to simplify the automated resource configuration generation, since finding satisfiable instances on random SMT propositions amounts to a random search. Moreover, to ensure uniformity we consider instances where the optimal plan is found at a bound of 5 (i.e., $k = 5$ ref. Section 5.5). Subsequently, we deploy the reasoning machinery on an edge device.

Our prototypical implementation employs the procedure described in Section 5.5 and is deployed on a low-powered ARMv8 R-Pi3 device featuring a 1.2GHz CPU and 1GB RAM, serving as the edge node. Given a resource configuration, the edge node's functionality – implemented in Python and C– consists essentially of the following steps: (i) the appropriate bounded model checking formula representation (Formula 5.6) is encoded depending on the problem instance, (ii) the CVC4 solver is invoked upon it, and (iii) the plan is computed from the satisfiability assignment of the solver. Functionality is exposed through lightweight REST, with which participating devices in the edge scope update context values and request coordination plans. The procedure described is invoked for every requesting device, resulting in the computation of a coordination plan. We ignore network overhead. As we observed in Section 5.5.3, the coordination plan consists of the resource invocations that the requesting device must perform within the edge scope to

satisfy its goal. Subsequently, we evaluate how such instances perform in practice by simulating requests from devices to the edge node. Device requests are drawn from the problem instance dataset of the previous step.

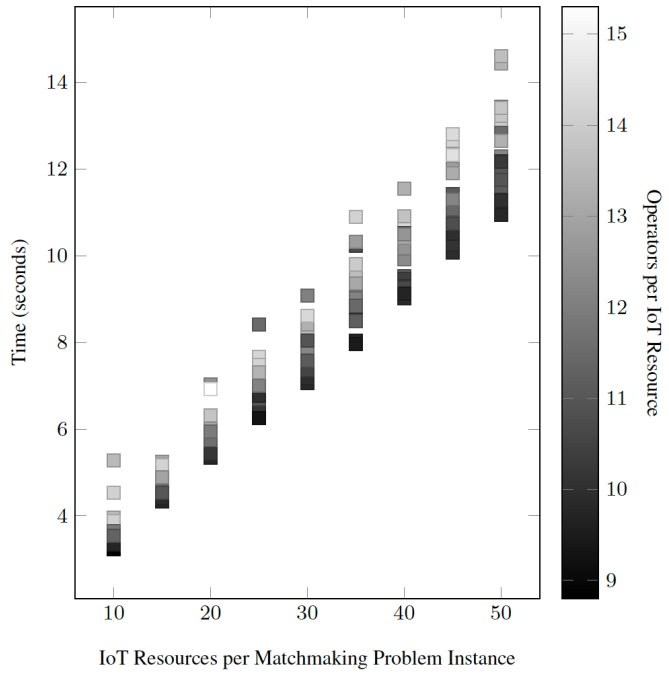
5.6.2 Experiment Results: Resource Coordination

To obtain experiment results based on the synthesized dataset, we simulate requests from devices, to investigate performance of the various problem instances, and we account for the time taken to coordinate the resources in every problem instance. We ignore network overhead, and we report on the total computation performance of the coordination plans, from a request to the plan response by the coordinating edge node.

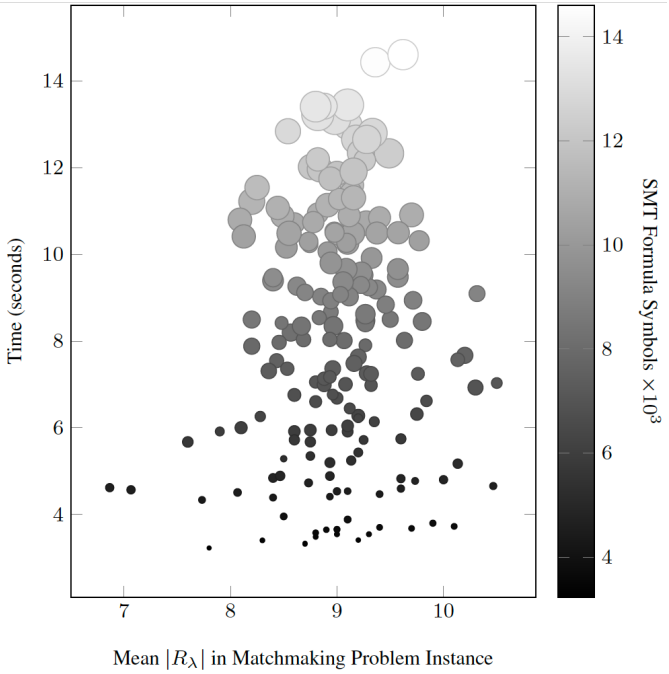
In Figure 5.5a, the number of IoT resources (X) over time is illustrated – each data point is a single problem instance (i.e. a resources-context-goal configuration). Additionally, one can observe the number of operators used in the requires dependancies of (every) resource in the configuration (shading in data points). Evidently, the more IoT resources there are in a problem instance the more time it takes to coordinate. Due to the boolean satisfiability solving [NLBH$^+$04] that underlies the matchmaking process, SAT/SMT problem instances with different number of operators perform differently, although the number of resources is kept constant (i.e., within a vertical line in Figure 5.5a). For example, one can observe that in the vertical line denoting 25 IoT resources, a problem instance utilizing 14 operators per (every) resource is harder than a problem instance with 10 operators. Hardness of SAT/SMT satisfiability [ABLM08] is beyond the scope of this thesis. Due to the synthesized nature of our evaluation dataset, we did not consider edge goals as those would be defined per application. However, those would not affect results significantly due to the small expected size that their encoding would add as an overhead.

Figure 5.5b captures mean $|R_\lambda|$ size per matchmaking problem instance across time. Each data point is a single problem instance – the same resources-context-goal configurations of the previous Figure 5.5a. The size of the resulting SAT/SMT formulae (as per Formula 5.6 of Section 5.5) corresponding to the coordination problem encoding is represented by the point size. Number of symbols within formulae range from $4k$ to $14k$. Naturally, as formula size increases, so does the coordination time. We can observe that again, certain problem instances with small average cardinality $|R_\lambda|$ lead to hard instances, and vice versa. However, the formula size is a strong indicator of coordination time.

Based on the above results, a system designer can obtain insights depending on her particular problem setting. Within a typical design process, a designer requires knowledge of the performance of the system, due to the definition of some Service Level Agreement (SLA). Our evaluation results show, that by selecting a number of IoT resources (e.g., 20 resources on Figure 5.5a) and an average number of operators per resource dependencies (e.g., 10), a matchmaking performance at least 5 seconds is to be expected. Then, formula size and average $|R_\lambda|$ can also give an indication of expected time.



(a) Resource set cardinality (|X| of matchmaking problem instances over coordination time. Shading in points indicates the average number of operators used in each satisfiable problem instance.



(b) Mean resource dependencies in matchmaking problem instances over coordination time. Points size indicates the size of the SMT encoding of the respective bounded model checking problem.

Figure 5.5: Matchmaking problem instances; coordination time, R operators and SMT symbols on an ARMv8 edge device.

5.6.3 Discussion

We have demonstrated that by using our coordination framework, coordination of IoT resources in an edge setting in a dependable manner can be performed. Furthermore, we showed that our technique based on SMT-based bounded model checking at the edge is performant and feasible for realistic problem sizes, even on low-powered ARM-based devices. We especially note that our resource coordination technique guarantees correct and optimal results due to its formal foundations.

We showed performance based on synthesized IoT resource problem instances. Our results are actionable since a system designer can estimate performance of coordination based on the problem space which is induced in her particular setting. This, combined with testing resource configurations prior to deployment can drive design decisions during system development. Essentially, given a number of IoT resources, propositions and operators per resource (Figures 5.5a - 5.5b), one can estimate a time that coordination computation can be achieved. The type of operators used within specification as well as the formula structure obviously affects satisfiability. We plan to investigate what type and mix of operators occurs in practice in IoT resources specification, and construct guidelines and metrics relevant to the resource encoding arising from our coordination technique.

Several assumptions inherent in our approach must be further investigated. We considered a coordination plan of 5, suitable for the business logic of resource-constrained devices participating within an IoT system; however there may exist settings where a higher or lower plan length is desired. Moreover, operationalization of our approach with optimality in mind, entails finding plans first on plan length 1, then if none is found on plan length 2, etc. To optimize this step-wise search, the SAT/SMT problem encoded can be incrementally introduced to a solver which makes use of past unfoldings to possibly find satisfiable solutions faster. We identify this as future work.

Moreover, temporal aspects of both the specification as well as the overall process must be investigated. Firstly. the planning time plus its execution (i.e. the device invoking the resources described in the plan) must be faster than the rate of change of the environment – as such, longer plans may not be advisable, and k should be largely kept small [TPGN18]. Secondly, temporal aspects regarding resource invocations are not captured in the model. Algebraic operations upon countable parameters would also be useful, as invocation of a resource microservice of a battery-powered actuator might consume energy. In our example for instance, irrigation in the park may take time. We identify integrating temporal aspects both regarding the model as well as planning and execution as a significant avenue of future work.

Operationalization of our framework could also benefit from domain-specific adjustments and heuristics. For example, previous plans may be stored (e.g., memoized) to avoid computing them again. The depth of the solution search may be adjusted depending on current edge computational load or other factors. On the problem level, grouping IoT resources in an ontology can allow the underlying solver to disregard irrelevant or

unsatisfiable solutions faster, thus rendering our approach capable to consider a higher number and more complex IoT resources. Finally, we note the absence of approaches utilizing SAT/SMT solving at the edge, and underline the opportunities that this brings for dependable edge-enabled IoT settings.

5.7 Related Work

We presented a methodology and technical framework to engineering resource coordination for the edge-enabled IoT – touching upon several research areas. Consequently, we classify related work into three categories. First, we discuss key approaches in the conception of resources as services within IoT. Then, we review related techniques on service composition, as they apply to IoT. Lastly, we discuss related engineering approaches from the domain of self-adaptive systems, framing our approach within the overall software engineering domain.

5.7.1 Resources as Services within IoT

The prevalence of internet-connected devices featuring various actuation and sensing capabilities provides new means for development of composite software systems. So far, cloud computing has been seen as a key component for the development, deployment, and coordination of IoT collectives.

In recent years, the paradigm of Service-Oriented Architecture (SOA) [GTK+10, LCH14] has received considerable attention in the field of IoT. Spiess et al. [SKG+09] proposed an architecture for effective integration of IoT in enterprise services, where they are used to implement business processes. Since the services are offered in a device level with frequent changes, a traditional business process language like BPEL is not built to support such dynamics. As result, an extended version of BPEL is provided to model business processes at design time which supports dynamic changes of services during process execution. Furthermore, to satisfy application needs or in response to unforeseen context changes, it is possible to remotely deploy new services during runtime. Meyer et al. [MRM13] investigate how an "IoT device" component and its native services can be expressed as a resource in an IoT-aware process model. Cheng et al. [CZZC16] propose a situation-aware IoT coordination platform based on the event-driven service-oriented architecture (SOA) paradigm. The proposed system architecture effectively utilizes SOA and EDA paradigms – SOA is used to resolve interoperability issues among heterogeneous services and physical entities while EDA is used to address the problem of the cross-business-domain. Furthermore, Zhang et al. [ZDC14] present an event-driven service-oriented architecture for IoT services. Sarkar et al. [SSP+15] proposed a layered and distributed architecture for IoT, which overcomes most of the obstacles in the process of large-scale expansion of IoT.

In pervasive environments, selecting appropriate resources and services that satisfy user's requirements is a challenge. Due to their dynamic nature, efficient resource discovery

is essential in order to achieve wide user acceptance. A significant number of works within service discovery has been focused on context based approaches. Butt et al. [BPGO13] provide a service selection technique to offer the appropriate service to a user application depending on the available context information. Rasch et al. [RLS+11] propose a proactive service discovery approach for pervasive environments, described by a formal context model which effectively captures the dynamics of context and the relationship between services and context. Wang et al.[WC16] propose an architecture for service discovery in smart cities, focusing on taking a set of devices around a citizen and proactively producing a list of services that surround the user using his preferences. Jin et al.[JCJL14] describe device, resource, and service as the three core concepts in a model which specifies relationships among them. Moreover, four quality of service attributes are defined, which reflect features of physical services. Yang and Li[YL14] propose an efficient strategy from the perspective of sensory and data selections and aggregation, with genetic algorithms as the global optimization method. Since we adopt the concept of everything-as-a-service (XaaS) abstraction to uniformly represent physical things, hardware and software resources as microservice such discussed works are relevant to our proposed approach.

Well-known approaches adopt semantic web technologies and matching techniques for effective service discovery. Yue-an Zhu [ZM10] design service discovery in pervasive computing using the description language OWL-S, matching services according to their category, Input/Output parameters, and QoS. Mokhtar et al. [MPG+08] support efficient, semantic, context- and QoS-aware service discovery [BMKGI06] on top of existing service discovery protocols (SDPs) [ZMN05] – this operates at a higher, semantic abstraction level, and is thus independent of the specific underlying SOA technology employed. In addition, a language for semantic service description covers both functional and non-functional service characteristics as well as a set of conformance relations and prescribes the way for applying them in order to perform service matching. Approaches related to the semantic web technologies used for service discovery are also relevant to our proposed approach. Since semantic service description covers both functional and non-functional they can be included also to the methodology that we propose for specifuying resources.

5.7.2 Service Composition and IoT

As service-oriented architecture (SOA) becomes widely used, providing the right services that satisfy a user's goal is becoming a big challenge in IoT environments. Due to dynamicity, heterogeneity and function constraints, IoT services differ from traditional services. In addition, IoT services are related more to the physical world by sensing state by inducing operations that will cause a state change. A great number of approaches have been proposed to deal with such service composition; we employ a SAT-based technique similar to Kil et al. [KN13], where the semantic aspect is considered, helping the composition engine to identify more correct, complete and optimal candidates as a solution. However, we extend it to SMT, we use linear integer arithmetic and first-order formulae, and deploy on resource-constrained edge devices instrumented at runtime.

Mayer et al. [MTS09] present a consistency-based service composition approach that serves as a unified platform. The proposed framework is based on a declarative constraint language to express user requirements, process constraints, and service profiles on a conceptual level and also on the instance level. Pistore et al. [PRT05] propose a solution for automated composition at the process level using OWL-S. A composition takes into account that executing a web service requires interactions that may involve different sequential, conditional, and iterative steps. A process level description of the composite service is generated by using each individual description of services. However, the proposed solution does not consider selecting the services that take part in the composition.

An architectural approach to enable the automated formation and adaptation of Emergent Configurations (ECs) in the IoT have been proposed by [ASD18b]. An EC is formed by a set of things, with their services, functionalities and applications, to realize a user goal. ECs are adapted in response to (un)foreseen context changes e.g., changes in available things or due to changing or evolving user goals. Hussein et al. [HLR17] propose a model-driven approach to ease the development of adaptive IoT systems. A design model is specified based on the system requirements as well as the system functionality and adaptations. Furthermore, it is used to generate the system implementation, transformed to an IoT platform-specific model. This model is used for generating code and a deployment to a hardware platform. After adaption is triggered, the system changes its state based on the designed model. Urbieta et al. [UGBM$^+$17] propose an adaptive service composition framework. The framework is based on an abstract service model representing services and user tasks in terms of their signature, specification, and conversation. Xinming and Yan [LS14] propose a service mining scheme based on semantic for IoT to provide users with interesting composite services. Service composition is achieved by combining and recommending to users according to the calculation of service similarity – however, not including service composition QoS.

Ciortea et al. [CBZF16] propose a decentralized approach to IoT mashup composition that considers flexibility and responsiveness of resulting applications. Goal-Driven software agents are equipped with precompiled plans which cooperate with one another through socio-technical networks (STNs) to compose IoT mashups at runtime in pursuit of their goals. Various IoT devices are modeled as agents based on their capabilities. Agents are goal-driven and rely on precompiled plans that specify how to achieve their goals. Whenever the goal cannot be fulfilled by a single agent, agents cooperate with one another through STNs to compose mashups which achieve the goals. In contrast, we synthesize plans at runtime, and utilizing available resources opportunistically from the runtime edge context. Mayer et al. [MVKM16] proposed service composition system that enables the goal-driven configuration of smart environments for end-users by combining semantic metadata and reasoning with a visual modeling tool – instead of using predefined service mashups, creation of them in a dynamic manner fulfills the desired user goal. This dynamicity is similar to our coordination approach. Such flexibility is achieved through using embedded semantic API descriptions. Hence, service mashups can adapt to dynamic environments and are fault-tolerant.

5.7.3 Self-adaptive systems and IoT

Self-adaptive software becomes an inseparable part of systems characterized by uncertain environments, evolving requirements, and unexpected failures. In order to meet strict functional and non-functional requirements in applications within diverse areas, Calinescu et al. [CWG+18] propose a methodology and instantiation of dynamic safety cases which allows adjusting the system during execution while providing the intended functionality and its requirements. Marrella et al. [MMS17] propose a model and prototype Process Management System featuring a set of techniques providing support for automated adaptation of knowledge-intensive processes at runtime. Such provided techniques are able to automatically adapt and recover process instances when an exception occurs and without the intervention of domain experts at runtime. Chen et al. [CLZ+15] propose a runtime model-based approach to IoT application development. The initial step toward the proposed approach is considering that sensor devices are abstracted as runtime models that are automatically connected with the corresponding systems. Based on the application scenario, a customized model is constructed and the synchronization between the model and sensor device is achieved through model transformation. As result, the application logic is mapped and executed on the customized model after some definition are given such as group of meta-models, mapping rules, and model-level programs.

In the context of self-management architectures, a significant number of generic approaches have been proposed. Kramer et al. [KM07] proposed a three-layer reference model to support automatic (re)configuration of self-managed systems, consisting of a component control layer, a change management layer and a goal management layer. The component layer is responsible to provide change management which re-configures the software components while the change management generates plans to achieve system goals. An overall model relies on a set of plans which aims to achieve the desired system goals. Whenever new goals are introduced to the system, the change management layer is responsible to generate new plans for achieving desired goals. Thus, we follow this methodology essentially targeting low-powered ARM-based edge computers for deployment. In addition, the resource coordination facilities we provide are dependable to each other. Weyns et al. [WIHM18] propose architecture-based adaptation approach to solve the concrete problem of automating the management of IoT. The software system utilizes a feedback loop that employs models@runtime and statistical techniques to reason about the system and induce adaptation to ensure the required goals.

Software systems are deployed in dynamic environments that change over time and often have to adapt to the changing conditions in order to meet system goals. Well-known approaches have been developed for runtime monitoring for different kinds of systems. Seiger et al. [SHHA17] present an approach for enabling self-adaptive workflows based on the MAPE-K (Monitor, Analyze, Plan and Execute on a Knowledge Base) control loop for self-adaptive workflows in cyber-physical systems. The proposed approach within MAPE-K loop monitors and analyses the real-world effects through sensor and context data which is used to check for faulty errors in the physical world. If an inconsistency between the sensed physical world and the assumed cyber world can be detected, a

compensation strategy is chosen and the adapted process is executed. Cailliau et al. [CvL17] proposes obstacle-driven runtime adaptation techniques for increased satisfaction of probabilistic system goals. The propose approach is based on the MAPE cycle and relies on a model where goals and obstacles are refined and specified in a probabilistic manner. However, in contrast to the proposed approach we guarantee the optimality and correctness of generated coordination plans. We identify quantitative extensions to our goal modeling as future work. The resource coordination facilities we provide are dependable because if there is a solution to a resource coordination problem for a device, the technique we utilize will provide a plan for it, and the plan will be optimal. This is in contrast to other approaches utilizing other coordination techniques such as based on AI [ASD18a, HN01, GNT04].

5.8 Summary

Software components within pervasive IoT systems, make use of resources which can be various computational capabilities, including sensing or actuation endpoints. Components do not live in isolation and must be able to coordinate with others to fulfill their goals, while edge computers placed near end-devices can be leveraged for control – providing resource management for devices within their active context. To this end, we proposed a methodology and technical framework for engineering resource coordination at runtime, tailored for the decentralized, pervasive systems of today. Our approach represents a paradigm shift in marrying distributed systems and formal aspects of software engineering. We adopted goal modeling to capture objectives within the IoT and used bounded model checking as the foundational technique to compute coordination plans which satisfy device goals. This occurs opportunistically at runtime, without any knowledge about the operational status or presence of resources in the system at the system's design time, but always in accordance to the edge's own goals. Our technical framework exhibits dependability guarantees regarding optimality and correctness of generated coordination plans, and is realizable on edge nodes deployed on low-powered ARM-based edge devices.

CHAPTER 6

Conclusion and Future Work

This chapter concludes the thesis. In Section 6.1, we give a summary of the covered topics and the presented contributions. In Section 6.2, we revisit the research questions provided in the introduction. Finally, in Section 6.3, we discuss the limitations of our work and provide a brief outlook for future research directions.

6.1 Summary

Within this thesis, our focus resided on providing feasible techniques for resource management at the edge. Such techniques aim to assist and manage runtime aspects of edge applications running on a resource-constrained edge network. More precisely, we develop a novel edge-based mechanism (i.e., configurator) that provides several resource management features such as (1) resource discovery in a decentralized manner, (2) controlling elasticity of edge applications at the edge, and (3) resource coordination at the edge.

In Chapter 2, we first hypothesized that the cloud-based IoT platform serves as an environment where developers or domain experts design their edge applications and define resource demands (i.e., application resource requirements, elasticity, etc.) and other dependencies (i.e., resource management or domain-specific functionalities). Furthermore, the cloud-based IoT platform allows developers to develop particular resource management functionalities. These functionalities can be deployed manually or automatically when an edge application with a specific functionality dependency is deployed in an edge network. These functionalities aim to assist edge applications from different domains to run correctly on an edge network. Nevertheless, developing the cloud-based IoT platform remains out of the main focus of this thesis - the introduced challenges within this platform require further investigation and future work to address all open research challenges. Second, we introduced a decentralized edge-based system for (1) automatically discovering heterogeneous resources in an edge network to make them available to the runtime, (2) deploying and controlling elasticity in edge applications running in an edge

network, and (3) enabling resource coordination at runtime. Through three resource management perspectives and their operational mechanisms, we enable executing and managing edge applications on heterogeneous edge infrastructures and beyond.

In Chapter 3, we presented a prototype that enables the automatic discovery of heterogeneous resources and makes them available to the runtime. The complexity of discovering resources automatically in edge networks necessitated investigating the most suitable communication model, resource modeling, and decentralized metadata management. For the former, we proposed a solution on how to organize edge devices within an edge network. Our edge-based system introduces the configurator and other cluster coordinators to overcome challenges presented with the uncertainty of an edge network. With the scaling of an edge network, new clusters are formed as well as new coordinators are created. Both the configurator and other coordinators are dynamically placed in the most suitable edge devices enabling the system to scale easily with the scaling of the edge network. At this stage, the edge device with the configurator role is responsible for assisting cluster coordinators in exchanging resource metadata information among them. The resulting mechanism efficiently discovers heterogeneous resources in a resource-constrained edge network.

In Chapter 4, we extend the configurator mechanism with novel functionalities such that it enables deploying, managing, and monitoring edge applications at the edge and beyond. More precisely, we propose a framework and its runtime mechanism for controlling elasticity requirements in edge applications. On the deployment request for a particular edge application, our runtime mechanism gets the resource requirements for each component (i.e., including elasticity requirements) and deployment and scaling policy for the overall application defined by the application developer or domain expert at design time. Based on these requirements, edge application components are deployed such that all requirements are fulfilled. After the successful deployment, each edge device that runs an edge application component enables locally the elasticity interpreter. The elasticity interpreter captures, interprets, and continuously monitors the elastic requirements for the application component that runs locally. Because the environment we are taking into account is highly dynamic and cannot be static, edge application components may face various workloads. Thus, whether a violation occurs, the responsible elasticity interpreter notifies the configurator to enforce the strategy specified in the elastic requirements. The enforcement strategies aim to improve or keep constant the overall edge application *quality* (i.e., response time, etc.), optimize the operating *cost*, and avoid situations such as *resource* over-provisioning or under-provisioning on resource-constrained infrastructures. The resulting mechanism efficiently adapts to varying load patterns by adjusting the amount of provisioned resources to exactly match their current need and minimizing hosting costs.

In Chapter 3 and Chapter 4, we showed three resource management techniques (i.e., resource discovery, resource allocation, and resource migration) to enable the operation of edge applications in a resource-constrained edge network. In Chapter 5, we propose a technical framework for engineering resource coordination at the edge. More precisely,

executing SAT/SMT solver on a low-powered edge device enables coordinating available resources to achieve users or edge applications goals (i.e., if possible). The resulting mechanism guarantees regarding optimality and correctness of generated coordination plans. Furthermore, the proposed framework is realizable on edge nodes deployed on low-powered ARM-based edge devices.

6.2 Research questions

In Section 1.2, we presented three research questions that have driven the research presented in this thesis. To conclude our thesis, we revisit research questions, summarize our contribution, and discuss the limitations of our work.

(**RQ**1) *What are appropriate system architectures that enable resource management in resource-constrained edge networks?*

We have addressed this question in Chapter 2 and partially in Chapter 5. We can make the following observations from the analysis in Chapter 2 of existing and emerging edge computing scenarios from different domains. We first advocate that edge devices should not exist in isolation and must collaborate with other edge devices to enable more complex tasks to be executed at the edge. Thereby, interaction with other edge devices allows for extending the scope of available resources. Accordingly, multiple heterogeneous edge devices will create computing clusters, similar to how cloud data center servers reside together in racks. Such an environment with numerous connected clusters forms an edge network within the spatial boundaries. Such edge networks are characterized as dynamic, uncertain, and resource-constrained environments. At the same time, they provide a seamless opportunity for deploying and executing more complex tasks (i.e., edge applications) and edge-based systems. Because edge networks are dynamic environments that change over time, we therefore argued and extensively elaborated on the necessity for designing and developing modern edge-based systems in a decentralized manner and with self-adaptive capabilities. Furthermore, because edge networks can scale, we argued that interaction between edge devices can be achieved through peer-to-peer communication protocols (i.e., DHT) followed by mechanisms to organize edge devices into clusters. Thus, Chapter 2 introduced a decentralized edge-based system that provides a collection of resource management mechanisms to enable deploying, executing, and maintaining the desired functionality of edge applications running in an edge network.

Edge computing has introduced new opportunities for designing and developing modern systems in a decentralized manner. Geographically distributed edge networks together with fog and cloud infrastructures will form the so-called Edge-Cloud computing continuum. Efforts to utilize available resources at the edge of the network and beyond have resulted in application-specific solutions ranging from centralized architectures to decentralized systems. We foresee that edge networks

will exist within different domains (i.e., healthcare, smart home, etc.); thus, the edge-based systems must support a wide range of edge application requirements. Our contribution has shown that a modular edge-based system is one way to add functionalities to support the execution of different edge applications and fulfill resource needs. Because edge applications may have various resource needs (i.e., goals), we argued and extensively elaborated a novel framework that enables resource coordination to obtain a resource that cannot be trivially acquired from available resources, as shown in Chapter 5. It is worth noting that the coordination process occurs in a decentralized manner and is executed on low-powered edge devices.

(**RQ**2) *What is an appropriate way to automatically discover and utilize heterogeneous resources in resource-constrained edge networks?*

Based on the analysis in Chapter 2 of existing and emerging edge computing scenarios, we can make a general observation that edge-based systems need to establish three design goals: (a) decentralization of the system components to enable the execution of resource management features on resource-constrained edge networks, (b) expandability of the system with novel resource management features such that to fulfill resource demands for edge applications (c) self-adapting to dynamic changes such that guarantees to provide reliable and low-latency services in proximity to end-users. By considering (a) and (c) goals mentioned above, we introduced a novel decentralized resource discovery mechanism in Chapter 3. Efficiently utilizing available resources distributed among multiple resource-constrained edge devices required a technique capable of discovering resource information. First, we referred to heterogeneous resources as computational (i.e., edge devices, etc.) and sensory (i.e., IoT sensors, actuators, etc.). Second, we hypothesize that sensory resources are attached to edge devices or networked IoT resources in the vicinity of an edge device. And third, we hypothesize that manufacturers provide information about the functionality and properties of each resource in JSON files stored in corresponding edge devices. Additionally, the system designer manually sets resource information such as IP or access type (i.e., private or public), or such information can be automatically assigned when a resource is operational. We referred to such resource information as resource metadata.

We have shown that exchanging resource metadata between edge devices in an automatic manner enables sharing resources across the whole system. Because each edge device replicates information about available resources in an edge network, users or edge applications can request any edge device to discover resources by performing complex queries locally. On the contrary, performing a resource discovery algorithm (i.e., by querying the entire network based on multiple keywords) for each resource is a network and computationally demanding process. We therefore argued that using the replication process for discovering resources is feasible and applicable even on resource-constrained edge networks.

Through examples, we have shown that an edge network could be a dynamic environment; as a result, we proposed that edge devices in proximity or within an area are connected in a peer-to-peer way using a DHT-based communication protocol. Even though peer-to-peer communication does not treat edge devices heterogeneity, we advocate that enabling resource discovery in an automatic and decentralized manner overcomes such an issue. Because edge devices can join continuously, the automatic resource exchange process becomes a highly network-demanding process and the scalability issue a critical challenge. Furthermore, because edge devices can be resource-constrained and dynamic, decentralization and self-adaptation become key requirements for reliable resource discovery. Thus, this first calls for a feasible approach to organizing edge devices and then introducing decentralization with self-adaptive capabilities in an edge network. We proposed a solution that introduced two roles (i.e., the configurator and cluster coordinators) dynamically assigned to edge devices. The proposed approach showed that scaling and automatic resource discovery are possible. Dividing an edge network into clusters, organizing newly joined edge devices into clusters, and decentralizing responsibilities by introducing new roles avoided the necessity to have N x N communication between edge devices. Our evaluation in a realistic testbed proves that the proposed solution is feasible on resource-constrained and dynamic edge networks.

(**RQ**3) *How can elasticity features be executed on low-powered computational resources in edge networks?*

We have addressed this question in Chapter 4 by extending the previous approach presented in Chapter 3 with a lightweight decentralized mechanism for controlling the elasticity of edge applications in edge networks. In Chapter 3, our evaluation proved that the proposed solution increases the number of eligible deployments on edge applications with IoT resource dependencies. We proposed that the edge device with the configurator role must provide mechanisms for deploying and managing edge applications in an edge network. Because edge networks are dynamic and edge applications are composed of microservices whose load may change over time, we advocate the necessity to enable elasticity features to guarantee QoS throughout their entire life cycle execution. Our proposed framework combines three different resource management techniques to efficiently utilize the computational resources in an edge network and maintain the desired edge application functionality throughout its entire execution. More precisely, our approach captures and interprets edge application requirements (i.e., including elastic requirements, deployment and scale policy) specified by the developer or domain expert. Because edge applications are composed of microservices, we allow elasticity requirements to be specified in different granularity. Our elasticity framework operates in a decentralized manner. The configurator provides the elasticity enactment engine to control and enforce various actions on the application runtime platform (i.e., docker-engine). Each edge device that executes application components captures, interprets, and monitors

corresponding elastic requirements and notifies the elastic enactment engine in the event of a violation. Our evaluation in a realistic testbed proves that the proposed solution is feasible on resource-constrained edge networks. Furthermore, an edge application operation may depend on specific IoT resources; thus, this challenges scaling operations in an edge network. Therefore, we have demonstrated that our approach by combining three resource management techniques (i.e., resource discovery, resource allocation, and resource migration) enabled us to correctly scale edge applications at runtime.

6.3 Limitations and Future Work

This section briefly discusses the limitations and future work of the proposed cloud-based IoT platform and edge-based framework. Furthermore, we discuss the boundaries for each proposed methodology to overcome the challenges discussed in this thesis.

6.3.1 Towards Decentralized Edge-Based Systems

Our decentralized technical framework is suitable for various use cases in different domains where the number of edge devices is not expected to be massive. The proposed approach allows system designers to create and configure their own edge network based on their needs. Afterward, the proposed framework enables edge devices to be organized within the edge network through the introduced techniques. The system and network can easily scale while resources are automatically discovered. Furthermore, at the request of the system designer, various edge applications can be deployed, while the elasticity mechanism maintains the desired functionality of edge applications running on the edge network.

Several aspects of the cloud-based IoT platform require further investigation since the proposed approach is a conceptual framework rather than a technical approach. Accordingly, this led us to hypothesize several aspects, such as the process when the system designer or developer requests to deploy an edge application or extend the functionalities of the configurator at the edge. Moreover, we assume that an edge application and its requirements are given by a developer or a domain expert at design time. Nevertheless, we address several crucial challenges for enabling resource management in a decentralized manner at the edge. Despite this, there are still open challenges that must be addressed to fully operationalize the configurator mechanism in an edge network. For instance, our thesis does not consider privacy and security issues that remain a critical aspect of edge-based systems. We hypothesized that edge devices within a newly formed edge network are trusted. This assumption should be replaced with proper privacy and security techniques to protect users' data processed in an edge network. One possible solution is to authenticate edge devices when they join an edge network as presented, for instance, in [PON+18]. Since each edge device is an entrance door to an edge network, this enforces that each edge device must provide such security mechanisms. Therefore, the above-mentioned features are not trivial tasks, and they require careful analysis and

proper mechanisms to prevent attacks or unwanted traffic in an edge network [RLM18]. Furthermore, protecting data by guaranteeing confidentiality and integrity [MYAZ15] is another critical challenge that must be addressed and further investigated in future work.

Additionally, integrating blockchain technology in the configurator, respectively, in the orchestrator component will enable edge networks to collaborate with other edge networks in the vicinity. The premise is simple: the edge owner installs a specific blockchain feature (e.g., blockchain function) on its own edge network, which automatically connects to the fog blockchain platform. The owner lists shareable resources and payment terms and gets paid when other users in the platform utilize them. For instance, blockchain platforms (e.g., distributed storage[1,2] or distributed computation power[3]) deployed on fog infrastructure enable edge network owners to earn money by renting their unused storage and idle CPU cycles to those in need.

One another promising research direction is to develop novel decentralized resource management techniques that target resource-constrained edge networks. Currently, migrating and executing several well-known cloud-based resource management techniques or AI algorithms to enable edge intelligence at the edge remains still a challenging task due to the dynamicity, uncertainty, and resource-constrained edge networks. Thus, developing novel and lightweight algorithms that target resource-constrained edge networks is crucial for the future of edge-based systems.

6.3.2 Resource Discovery

In Chapter 3, respectively, in Section 3.5.1, we have shown several limitations of our proposed approach. In this thesis, we do not treat communications aspects between IoT resources and edge devices. We assume that these resources are attached or connected to edge devices. Furthermore, the main limitation of our approach is that we developed our prototype in Java which requires using the Java Virtual Machine as a runtime platform. Running the JVM in resource-constrained devices is a resource-intensive process. Nevertheless, our proposed technique aims to show the feasibility of discovering resources.

After the systematic review of P2P protocols, we decided to use the well-known DHT-based protocol called Kademlia protocol to enable communication between edge devices. Thus, our proposed approach is built on top of Kademlia. However, we have observed that our approach has several limitations when determining coordinators on clusters of different sizes. Our evaluation shows that the percentage of responsive edge devices during the process to determine coordinator on a cluster with more than 40 devices decreases around to 92%. More precisely, on a cluster with 30 edge devices, we can observe that all edge devices respond on time to the process of determining a cluster coordinator. We acknowledge that we may have various edge scenarios where the number

[1]Storj (Distributed Storage), https://storj.io
[2]Filecoin (Distributed Storage), https://filecoin.io
[3]iExec (Distributed Computation), https://iex.ec/

of edge devices within a cluster needs to be more extensive. Thus, further optimizations are required as well as several assumptions described in Chapter 3 must be further explored. Nevertheless, the evaluation on a realistic testbed with 60 edge devices (i.e., RPi3 and virtual instances) showed that the overall overhead introduced by our resource discovery mechanism is very low and feasible in resource-constrained edge networks. In future work, we plan to investigate techniques that will enable edge devices to discover nearby devices that allow them to join an edge network in an automatic manner. In addition to that, we plan to investigate additional components for monitoring the status and availability of discovered resources at the edge.

6.3.3 On Controlling Elasticity of Edge Applications

In Chapter 4, we have shown several limitations of our approach, especially during the design time and deployment stage of edge applications. We first assume that the developer or a domain expert has basic knowledge of how to develop elastic specifications and specify correct values. In the current prototype version, we assume that elastic constraints do not conflict with each other. Nevertheless, the above-mentioned aspect represents a limitation of our approach. We acknowledge that developers or domain experts may specify conflicting elastic requirements, leading to a wrong configuration and causing undesired and frequently scaling operations.

In the current version of our prototype, we specify elastic requirements in a JSON file. Such a file for each edge application is accessible by any edge device by querying the specific DHT. However, we must provide proper techniques such that the SYBL elastic requirements can be quickly injected/integrated into various description languages, e.g., injecting into TOSCA files, docker-compose files (i.e., YAML), or XML descriptions. Supporting various description languages is particularly important since edge applications are designated to run on different computing infrastructures available within the Edge-Cloud continuum. In future work, we intend to simplify the development process of elastic requirements by integrating the language into an IDE such as c-Eclipse. One way to integrate the language is through the c-Eclipse framework, which provides a user-centric interface for developers to describe their applications for deployment over edge and cloud. The language integrated into a development environment will make it easy for users to develop elastic specifications and specify correct values to avoid the wrong configuration. Nevertheless, the following tool will also help to automatically detect conflicting constraints that the user may select.

We consider only an edge network with a single cluster in our evaluation. Our solution supports controlling the elasticity of edge applications running on an edge network with multiple clusters. However, since the configurator is the only node that enforces scaling operations, performance issues may arise when the edge network continuously scales. The performance can become even more critical when multiple edge applications are deployed and numerous elastic violations occur concurrently. This, in turn, may generate some latencies until the configurator takes action. Thus, further decentralization of the elastic framework is necessary to allow the system's scalability and improve the

overall configurator response time. Nevertheless, in future work, we plan to extend the functionality of the cluster coordinators, which will support the configurator to enforce various actions when several elastic violations simultaneously occur.

6.3.4 Resource coordination

We believe that enabling coordination at the edge, as presented in Chapter 5, paves the way for situating control logic close to end-devices, leading to increased decentralization in edge-based systems. We plan to investigate dealing with conflicting goals, where edge nodes or devices have requirements that overlap but do not agree [NKF94] and which may require negotiation to resolve. Although goals in our approach were assumed to be functional, if non-functional requirements are considered, tradeoffs may need to be made concerning, e.g., cost, performance, energy, etc. We identify integrating temporal aspects regarding the model and planning and execution performance as a significant avenue of future work. Furthermore, temporal aspects of both specifications, as well as the overall coordination process must be further investigated. Resources may be countable, reflecting some limited availability such as a cost – this requires extending the encoding. Moreover, resource invocation timings can be captured in the model, as countable parameters would be helpful for quantitative requirements specification related to, e.g., device SLAs. Operational aspects that need to be investigated regarding the coordination process must be faster than the rate of change in the environment. Finally, considering the perspective of a complete realization of the dependable runtime coordination approach presented, distributed systems aspects need to be treated.

Furthermore, our approach proposed within this thesis focuses only on enabling resource coordination at the edge. Nevertheless, we investigated a different method and analyzed different perspectives to allow resource coordination by considering a wide range of computational resources within the Edge-Cloud infrastructure. Several resources should temporarily cooperate to accomplish the requested goal to achieve user or edge application goals. We refer to this process as dynamically forming a Goal-driven IoT System (GDS), which utilizes available IoT resources and enables cooperation between resources to achieve various goals requested at runtime. Each IoT resource may have several functionalities; for instance, the intelligent light may have multiple functionalities like turning on, turning off, or maintaining light levels. Accordingly, the number and types of the IoT resources that constitute a GDS (i.e., goal) can differ from one environment to another. Therefore, a GDS may have different performance requirements based on the requested goals. Thus, we focus on the deployment aspect, such as where a GDS should be formed in Edge-Cloud infrastructure. The resulting mechanism efficiently generates optimal plans where a GDS should be formed. We analyzed two scenarios in a simulation environment to show the feasibility and applicability of the proposed approach. Thus, this contribution focuses on enabling dynamic deploying and adapting GDS in response to changes in their constituents or deployment topologies. We present the contribution originally in [AMS+20].

List of Figures

List of Tables

List of Algorithms

Bibliography

[AADP15] Hamid Reza Arkian, Reza Ebrahimi Atani, Abolfazl Diyanat, and Atefe Pourkhalili. A cluster-based vehicular cloud architecture with learning-based resource management. *The Journal of Supercomputing*, 71(4):1401–1426, 2015.

[ABH+02] Anupriya Ankolekar, Mark Burstein, Jerry R Hobbs, Ora Lassila, David Martin, Drew McDermott, Sheila A McIlraith, Srini Narayanan, Massimo Paolucci, Terry Payne, et al. Daml-s: Web service description for the semantic web. In *International Semantic Web Conference*, pages 348–363. Springer, 2002.

[ABL15] Jean-Paul Arcangeli, Raja Boujbel, and Sébastien Leriche. Automatic deployment of distributed software systems: Definitions and state of the art. *Journal of Systems and Software*, 103:198–218, 2015.

[ABLM08] Carlos Ansótegui, María Luisa Bonet, Jordi Levy, and Felip Manya. Measuring the hardness of sat instances. In *AAAI*, volume 8, pages 222–228, 2008.

[AD18] Cosmin Avasalcai and Schahram Dustdar. Latency-aware decentralized resource management for iot applications. In *Proceedings of the 8th International Conference on the Internet of Things*, page 30. ACM, 2018.

[AD19] Cosmin Avasalcai and Schahram Dustdar. Latency-aware distributed resource provisioning for deploying iot applications at the edge of the network. In *Future of Information and Communication Conference*, pages 377–391. Springer, 2019.

[ADS18] Majid Ashouri, Paul Davidsson, and Romina Spalazzese. Cloud, edge, or both? towards decision support for designing iot applications. In *2018 Fifth International Conference on Internet of Things: Systems, Management and Security*, pages 155–162. IEEE, 2018.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A Survey. *Computer Networks*, 54(15):2787–2805, 2010.

[AMD20] Cosmin Avasalcai, Ilir Murturi, and Schahram Dustdar. Edge and fog: A survey, use cases, and future challenges. *Fog Computing: Theory and Practice*, pages 43–65, 2020.

[AMS+20] Fahed Alkhabbas, Ilir Murturi, Romina Spalazzese, Paul Davidsson, and Schahram Dustdar. A goal-driven approach for deploying self-adaptive iot systems. In *2020 IEEE International Conference on Software Architecture (ICSA)*, pages 146–156. IEEE, 2020.

[AP18] Arif Ahmed and Guillaume Pierre. Docker container deployment in fog computing infrastructures. In *2018 IEEE International Conference on Edge Computing (EDGE)*, pages 1–8. IEEE, 2018.

[ASD18a] Fahed Alkhabbas, Romina Spalazzese, and Paul Davidsson. Eco-iot: an architectural approach for realizing emergent configurations in the internet of things. In *European Conference on Software Architecture*, pages 86–102. Springer, 2018.

[ASD18b] Fahed Alkhabbas, Romina Spalazzese, and Paul Davidsson. Eco-iot: An architectural approach for realizing emergent configurations in the internet of things. In *European Conference on Software Architecture*, pages 86–102. Springer, 2018.

[ATC+21] Mohammad S Aslanpour, Adel N Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. Serverless edge computing: vision and challenges. In *2021 Australasian Computer Science Week Multiconference*, pages 1–10, 2021.

[ATD19] Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. Decentralized resource auctioning for latency-sensitive edge computing. In *2019 IEEE International Conference on Edge Computing (EDGE)*, pages 72–76. IEEE, 2019.

[AV15] Priyanka Athwani and Deo Prakash Vidyarthi. Resource discovery in mobile cloud computing: A clustering based approach. In *2015 IEEE UP Section Conference on Electrical Computer and Electronics (UPCON)*, pages 1–6. IEEE, 2015.

[BCC+99] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Masahiro Fujita, and Yunshan Zhu. Symbolic model checking using sat procedures instead of bdds. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 317–320. ACM, 1999.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. 2003.

[BCD+11a] Clark Barrett, Christopher L Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. Cvc4. In *International Conference on Computer Aided Verification*, pages 171–177. Springer, 2011.

[BCD+11b] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovi'c, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In Ganesh Gopalakrishnan and Shaz Qadeer, editors, *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV '11)*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, July 2011. Snowbird, Utah.

[BDW13] Wolfgang Beer, Bernhard Dorninger, and Mario Winterer. Flexible and reliable software architecture for industrial user interfaces. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–6. IEEE, 2013.

[BF17] Antonio Brogi and Stefano Forti. Qos-aware deployment of iot applications through the fog. *IEEE Internet of Things Journal*, 4(5):1185–1192, 2017.

[BFG19] Antonio Brogi, Stefano Forti, and Marco Gaglianese. Measuring the fog, gently. In *International Conference on Service-Oriented Computing*, pages 523–538. Springer, 2019.

[BFGL19] Antonio Brogi, Stefano Forti, Carlos Guerrero, and Isaac Lera. How to Place Your Apps in the Fog-State of the Art and Open Challenges. *arXiv preprint arXiv:1901.05717*, 2019.

[BM18] Xhevahir Bajrami and Ilir Murturi. An efficient approach to monitoring environmental conditions using a wireless sensor network and nodemcu. *e & i Elektrotechnik und Informationstechnik*, 135(3):294–301, 2018.

[BMKGI06] Sonia Ben Mokhtar, Anupam Kaul, Nikolaos Georgantas, and Valérie Issarny. Efficient semantic service discovery in pervasive computing environments. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, pages 240–259. Springer-Verlag New York, Inc., 2006.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing*, pages 13–16. ACM, 2012.

[BPGO13] Talal Ashraf Butt, Iain Phillips, Lin Guan, and George Oikonomou. Adaptive and context-aware service discovery for the internet of things. In *Internet of things, smart spaces, and next generation networking*, pages 36–47. Springer, 2013.

[BS07] Changhun Bae and Wayne E Stark. A tradeoff between energy and bandwidth efficiency in wireless networks. In *MILCOM 2007-IEEE Military Communications Conference*, pages 1–7. IEEE, 2007.

[BSH+17] Athman Bouguettaya, Munindar Singh, Michael Huhns, Quan Z Sheng, Hai Dong, Qi Yu, Azadeh Ghari Neiat, Sajib Mistry, Boualem Benatallah, Brahim Medjahed, et al. A service computing manifesto: the next 10 years. *Communications of the ACM*, 60(4):64–72, 2017.

[BT18] Clark Barrett and Cesare Tinelli. Satisfiability modulo theories. In *Handbook of Model Checking*, pages 305–343. Springer, 2018.

[CBZF16] Andrei Ciortea, Olivier Boissier, Antoine Zimmermann, and Adina Magda Florea. Responsive decentralized composition of service mashups for the internet of things. In *Proceedings of the 6th International Conference on the Internet of Things*, pages 53–61. ACM, 2016.

[CDK+02] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the web services web: an introduction to soap, wsdl, and uddi. *IEEE Internet computing*, 6(2):86–93, 2002.

[CGP99] Edmund M Clarke, Orna Grumberg, and Doron A Peled. *Model Checking*. MIT press, 1999.

[CK18] Djabir Abdeldjalil Chekired and Lyes Khoukhi. Multi-tier fog architecture: A new delay-tolerant network for iot data processing. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[CLZ+15] Xing Chen, Aipeng Li, Xue'e Zeng, Wenzhong Guo, and Gang Huang. Runtime model based approach to iot application development. *Frontiers of Computer Science*, 9(4):540–553, 2015.

[CMTD13] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. Sybl: An extensible language for controlling elasticity in cloud applications. In *2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, pages 112–119. IEEE, 2013.

[Cou] CouchDB. https://www.couchdb.com/. [Online accessed: January 2020].

[CSBW09] B. Cheng, P. Sawyer, N. Bencomo, and J. Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MoDELS'09)*, pages 468–483, 2009.

[CvL17] Antoine Cailliau and Axel van Lamsweerde. Runtime monitoring and resolution of probabilistic obstacles to system goals. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2017 IEEE/ACM 12th International Symposium on*, pages 1–11. IEEE, 2017.

[CWG+18] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering*, 44(11):1039–1069, 2018.

[CZZC16] Bo Cheng, Da Zhu, Shuai Zhao, and Junliang Chen. Situation-aware iot service coordination using the event-driven soa paradigm. *IEEE Transactions on Network and Service Management*, 13(2):349–361, 2016.

[DAM19] Schahram Dustdar, Cosmin Avasalcai, and Ilir Murturi. Edge and fog computing: Vision and research challenges. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 96–9609. IEEE, 2019.

[DBBM11] Suparna De, Payam Barnaghi, Martin Bauer, and Stefan Meissner. Service modelling for the internet of things. In *2011 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 949–955. IEEE, 2011.

[DGC+16] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of Things*, pages 61–75. Elsevier, 2016.

[DGST11] Schahram Dustdar, Yike Guo, Benjamin Satzger, and Hong-Linh Truong. Principles of elastic processes. *IEEE Internet Computing*, 15(5):66–71, 2011.

[DM20] Schahram Dustdar and Ilir Murturi. Towards distributed edge-based systems. In *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI)*, pages 1–9. IEEE, 2020.

[DM21] Schahram Dustdar and Ilir Murturi. *Towards IoT Processes on the Edge*, pages 167–178. Springer International Publishing, Cham, 2021.

[DMB18] Vincenzo De Maio and Ivona Brandic. First hop mobile offloading of dag computations. In *Proceedings of the 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 83–92. IEEE Press, 2018.

[DS21] V Divya and Leena R Sri. Fault tolerant resource allocation in fog environment using game theory-based reinforcement learning. *CONCURRENCY AND COMPUTATION-PRACTICE & EXPERIENCE*, 2021.

[EMB11] Niklas Een, Alan Mishchenko, and Robert Brayton. Efficient implementation of property directed reachability. In *Proceedings of the International Conference on Formal Methods in Computer-Aided Design*, pages 125–134. FMCAD Inc, 2011.

[End00] Herbert B Enderton. A mathematical introduction to logic. undergraduate texts in mathematics, 2000.

[EPM13] Thomas Erl, Ricardo Puttini, and Zaigham Mahmood. *Cloud computing: concepts, technology, & architecture*. Pearson Education, 2013.

[Erl05] Thomas Erl. Service-oriented architecture (soa): concepts, technology, and design, 2005.

[F+16] Frank HP Fitzek et al. On network coded distributed storage: How to repair in a fog of unreliable peers. In *2016 International Symposium on Wireless Communication Systems (ISWCS)*, pages 188–193. IEEE, 2016.

[FACP18] Jonathan Fürst, Mauricio Fadel Argerich, Bin Cheng, and Apostolos Papageorgiou. Elastic services for edge computing. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 358–362. IEEE, 2018.

[FAS17] Qiang Fan, Nirwan Ansari, and Xiang Sun. Energy driven avatar migration in green cloudlet networks. *IEEE Communications Letters*, 21(7):1601–1604, 2017.

[FFvLP98] M. S. Feather, S. Fickas, A. van Lamsweerde, and C. Ponsard. Reconciling system requirements and runtime behavior. In *Proceedings of the 9th international workshop on Software specification and design*, page 50. IEEE Computer Society, 1998.

[FGB20] Stefano Forti, Marco Gaglianese, and Antonio Brogi. Lightweight self-organising distributed monitoring of fog infrastructures. *Future Generation Computer Systems*, 2020.

[FGB21] Stefano Forti, Marco Gaglianese, and Antonio Brogi. Lightweight self-organising distributed monitoring of fog infrastructures. *Future Generation Computer Systems*, 114:605–618, 2021.

[FGRT16] Christine Fricker, Fabrice Guillemin, Philippe Robert, and Guilherme Thompson. Analysis of an offloading scheme for data centers in the framework of fog computing. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, 1(4):1–18, 2016.

[FJFCSG+19] Rafael Fayos-Jordan, Santiago Felici-Castell, Jaume Segura-Garcia, Adolfo Pastor-Aparicio, and Jesus Lopez-Ballester. Elastic computing in the

fog on internet of things to improve the performance of low cost nodes. *Electronics*, 8(12):1489, 2019.

[FSH17] Shuo Feng, Peyman Setoodeh, and Simon Haykin. Smart home: Cognitive interactive people-centric internet of things. *IEEE Communications Magazine*, 55(2):34–39, 2017.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[GD18] Marjan Gusev and Schahram Dustdar. Going back to the roots—the evolution of edge computing, an iot perspective. *IEEE Internet Computing*, 22(2):5–15, 2018.

[GIMA10] Daniel Giusto, Antonio Iera, Giacomo Morabito, and Luigi Atzori. *The internet of things: 20th Tyrrhenian Workshop on Digital Communications*. Springer Science & Business Media, 2010.

[GNT04] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: theory and practice*. Elsevier, 2004.

[GRT09] Emanuele Goldoni, Giuseppe Rossi, and Alberto Torelli. Assolo, a new method for available bandwidth estimation. In *2009 Fourth International Conference on Internet Monitoring and Protection*, pages 130–136. IEEE, 2009.

[GTK+10] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *IEEE transactions on Services Computing*, 3(3):223–235, 2010.

[GZG+15] Lin Gu, Deze Zeng, Song Guo, Ahmed Barnawi, and Yong Xiang. Cost efficient resource management in fog computing supported medical cyber-physical system. *IEEE Transactions on Emerging Topics in Computing*, 5(1):108–119, 2015.

[HHI19] Morghan Hartmann, Umair Sajid Hashmi, and Ali Imran. Edge computing in smart health care systems: Review, challenges, and research directions. *Transactions on Emerging Telecommunications Technologies*, page e3710, 2019.

[HKR13] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in cloud computing: What it is, and what it is not. In *Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13)*, pages 23–27, 2013.

[HLR17] Mahmoud Hussein, Shuai Li, and Ansgar Radermacher. Model-driven development of adaptive iot systems. In *4st International Workshop on Interplay of Model-Driven and Component-Based Software Engineering (ModComp) 2017 Workshop Pre-proceedings*, page 20, 2017.

[HN01] Jörg Hoffmann and Bernhard Nebel. The ff planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14:253–302, 2001.

[HV19] Cheol-Ho Hong and Blesson Varghese. Resource management in fog/edge computing: a survey on architectures, infrastructure, and algorithms. *ACM Computing Surveys (CSUR)*, 52(5):1–37, 2019.

[HXWC15] Mohammed A Hassan, Mengbai Xiao, Qi Wei, and Songqing Chen. Help your mobile applications with fog computing. In *2015 12th Annual IEEE International Conference on Sensing, Communication, and Networking-Workshops (SECON Workshops)*, pages 1–6. IEEE, 2015.

[IBM] IBM. Nmon. https://developer.ibm.com/technologies/systems/articles/aunmon_analyser/. [Online accessed: January 2020].

[IFB+18] Michaela Iorga, Larry Feldman, Robert Barton, Michael J Martin, Nedim S Goren, and Charif Mahmoudi. Fog computing conceptual model. Technical report, 2018.

[IM09] Stratis Ioannidis and Peter Marbach. Absence of evidence as evidence of absence: A simple mechanism for scalable p2p search. In *IEEE INFOCOM 2009*, pages 576–584. IEEE, 2009.

[JCJL14] Xiongnan Jin, Sejin Chun, Jooik Jung, and Kyong-Ho Lee. Iot service selection based on physical service model and absolute dominance relationship. In *Service-Oriented Computing and Applications (SOCA), 2014 IEEE 7th International Conference on*, pages 65–72. IEEE, 2014.

[JMB06] Gian Paolo Jesi, Alberto Montresor, and Ozalp Babaoglu. Proximity-aware superpeer overlay topologies. In *IEEE International Workshop on Self-Managed Networks, Systems, and Services*, pages 43–57. Springer, 2006.

[JT17] Rakesh Jain and Samir Tata. Cloud to edge: distributed deployment of process-aware iot applications. In *2017 IEEE International Conference on Edge Computing (EDGE)*, pages 182–189. IEEE, 2017.

[Kar19] Vasileios Karagiannis. Compute node communication in the fog: Survey and research challenges. In *Proceedings of the Workshop on Fog Computing and the IoT*, pages 36–40. ACM, 2019.

[KL14] Jaeho Kim and Jang-Won Lee. Openiot: An open service framework for the internet of things. In *2014 IEEE world forum on internet of things (WF-IoT)*, pages 89–93. IEEE, 2014.

[KM07] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *2007 Future of Software Engineering*, pages 259–268. IEEE Computer Society, 2007.

[KN13] Hyunyoung Kil and Wonhong Nam. Semantic web service composition via model checking techniques. *International Journal of Web and Grid Services*, 9(4):339–350, 2013.

[KSL+19] Vasileios Karagiannis, Stefan Schulte, Joao Leitao, et al. Enabling fog computing using self-organizing compute nodes. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2019.

[KTD06] Mohammed Ali Kaafar, Thierry Turletti, and Walid Dabbous. A locating-first approach for scalable overlay multicast. In *200614th IEEE International Workshop on Quality of Service*, pages 2–11. IEEE, 2006.

[LB20] Imen Ben Lahmar and Khouloud Boukadi. Resource allocation in fog computing: A systematic mapping study. In *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 86–93. IEEE, 2020.

[LC88] Victor R Lesser and Daniel D Corkill. Functionally accurate, cooperative distributed systems. In *Readings in Distributed Artificial Intelligence*, pages 295–310. Elsevier, 1988.

[LCH14] Jenq-Shiou Leu, Chi-Feng Chen, and Kun-Che Hsu. Improving heterogeneous soa-based iot message stability by shortest processing time scheduling. *IEEE Transactions on Services Computing*, 7(4):575–585, 2014.

[LDMB17] Ivan Lujic, Vincenzo De Maio, and Ivona Brandic. Efficient edge storage management based on near real-time forecasts. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 21–30. IEEE, 2017.

[LMSM10] S. Liaskos, S. McIlraith, S. Sohrabi, and J. Mylopoulos. Integrating preferences into goal models for requirements engineering. In *Proceedings of the 18th IEEE International Requirements Engineering Conference*, pages 135–144, 2010.

[LNST14] Wei Liu, Takayuki Nishio, Ryoichi Shinkuma, and Tatsuro Takahashi. Adaptive resource discovery in mobile cloud computing. *Computer Communications*, 50:119–129, 2014.

[LOD18] He Li, Kaoru Ota, and Mianxiong Dong. Learning iot in edge: Deep learning for the internet of things with edge computing. *IEEE Network*, 32(1):96–101, 2018.

[LPSD17] Adrien Lebre, Jonathan Pastor, Anthony Simonet, and Frédéric Desprez. Revising openstack to operate fog/edge computing infrastructures. In *2017 IEEE international conference on cloud engineering (IC2E)*, pages 138–148. IEEE, 2017.

[LS14] Xinming Li and Yan Sun. A service mining scheme based on semantic for internet of things. *Chinese Journal of Electronics*, 23(2):236–242, 2014.

[LT19] Ivan Lujic and Hong-Linh Truong. Architecturing elastic edge storage services for data-driven decision making. In *European Conference on Software Architecture*, pages 97–105. Springer, 2019.

[LTJ+19] Nianyu Li, Christos Tsigkanos, Zhi Jin, Schahram Dustdar, Zhenjiang Hu, and Carlo Ghezzi. Poet: Privacy on the edge with bidirectional data transformations. In *2019 IEEE International Conference on Pervasive Computing and Communications, PerCom 2019, Kyoto, Japan, March 11-15, 2019*. IEEE, 2019.

[MATD19] Ilir Murturi, Cosmin Avasalcai, Christos Tsigkanos, and Schahram Dustdar. Edge-to-edge resource discovery using metadata replication. In *2019 IEEE 3rd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–6. IEEE, 2019.

[MB17] Pavel Mach and Zdenek Becvar. Mobile edge computing: A survey on architecture and computation offloading. *IEEE Communications Surveys & Tutorials*, 19(3):1628–1656, 2017.

[MBP13] Stephen Makonin, Lyn Bartram, and Fred Popowich. A smarter smart home: Case studies of ambient intelligence. *IEEE Pervasive Computing*, 12(1):58–66, 2013.

[MCB+21] Adriana Mijuskovic, Alessandro Chiumento, Rob Bemthuis, Adina Aldea, and Paul Havinga. Resource management techniques for cloud/fog and edge computing: An evaluation framework and classification. *Sensors*, 21(5):1832, 2021.

[MCPR14] Davide Mascitti, Marco Conti, Andrea Passarella, and Laura Ricci. Service provisioning through opportunistic computing in mobile clouds. *Procedia Computer Science*, 40:143–150, 2014.

[MD21a] Ilir Murturi and Schahram Dustdar. Decent: A decentralized configurator for controlling elasticity in dynamic edge networks. *ACM Transactions on Internet Technology (TOIT)*, 2021.

[MD21b] Ilir Murturi and Schahram Dustdar. A decentralized approach for resource discovery using metadata replication in edge networks. *IEEE Transactions on Services Computing*, 2021.

[MJK+21] Ilir Murturi, Chao Jia, Bernhard Kerbl, Michael Wimmer, Schahram Dustdar, and Christos Tsigkanos. On provisioning procedural geometry workloads on edge architectures. In *17th International Conference on Web Information Systems and Technologies (WEBIST)*, pages 1–6, 2021.

[MKB18] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of everything*, pages 103–130. Springer, 2018.

[MLM+06] C Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, Rebekah Metz, and Booz Allen Hamilton. Reference model for service oriented architecture 1.0. *OASIS standard*, 12:18, 2006.

[MM02] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[MMS17] Andrea Marrella, Massimo Mecella, and Sebastian Sardina. Intelligent process adaptation in the smartpm system. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):25, 2017.

[MPG+08] Sonia Ben Mokhtar, Davy Preuveneers, Nikolaos Georgantas, Valérie Issarny, and Yolande Berbers. Easy: Efficient semantic service discovery in pervasive computing environments with qos and context support. *Journal of Systems and Software*, 81(5):785–808, 2008.

[MRB20] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. Application management in fog computing environments: A taxonomy, review and future directions. *ACM Computing Surveys (CSUR)*, 53(4):1–43, 2020.

[MRM13] Sonja Meyer, Andreas Ruppen, and Carsten Magerkurth. Internet of things-aware process modeling: integrating iot devices as business process resources. In *International conference on advanced information systems engineering*, pages 84–98. Springer, 2013.

[MS14] Sunilkumar S Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (iaas) in cloud computing: A survey. *Journal of network and computer applications*, 41:424–440, 2014.

[MSBD18] Mohammad Mozaffari, Walid Saad, Mehdi Bennis, and Mérouane Debbah. Communications and control for wireless drone-based antenna array. *IEEE Transactions on Communications*, 67(1):820–834, 2018.

[MTS09] Wolfgang Mayer, Rajesh Thiagarajan, and Markus Stumptner. Service composition as generative constraint satisfaction. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 888–895. IEEE, 2009.

[MVH+04] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[MVKM16] Simon Mayer, Ruben Verborgh, Matthias Kovatsch, and Friedemann Mattern. Smart configuration of smart environments. *IEEE Trans. Automation Science and Engineering*, 13(3):1247–1255, 2016.

[MWV00] Navneet Malpani, Jennifer L Welch, and Nitin Vaidya. Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 96–103, 2000.

[MYAZ15] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341. IEEE, 2015.

[NKF94] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *IEEE Transactions on software engineering*, 20(10):760–773, 1994.

[NLBH+04] Eugene Nudelman, Kevin Leyton-Brown, Holger H Hoos, Alex Devkar, and Yoav Shoham. Understanding random sat: Beyond the clauses-to-variables ratio. In *International Conference on Principles and Practice of Constraint Programming*, pages 438–452. Springer, 2004.

[NMP+20] Stefan Nastic, Andrea Morichetta, Thomas Pusztai, Schahram Dustdar, Xiaoning Ding, Deepak Vij, and Ying Xiong. Sloc: Service level objectives for next generation cloud computing. *IEEE Internet Computing*, 24(3):39–50, 2020.

[NPP+20] Nguyen Dinh Nguyen, Linh-An Phan, Dae-Heon Park, Sehan Kim, and Taehong Kim. Elasticfog: elastic resource provisioning in container-based fog computing. *IEEE Access*, 8:183879–183890, 2020.

[PAG+19] Pasquale Pace, Gianluca Aloi, Raffaele Gravina, Giuseppe Caliciuri, Giancarlo Fortino, and Antonio Liotta. An edge-based architecture to support efficient applications for healthcare industry 4.0. *IEEE Transactions on Industrial Informatics*, 15(1):481–489, 2019.

[PKBK20] Sharmila Patil-Karpe, SH Brahmananda, and Shrunoti Karpe. Review of resource allocation in fog computing. In *Smart Intelligent Computing and Applications*, pages 327–334. Springer, 2020.

[PON+18] Deepak Puthal, Mohammad S Obaidat, Priyadarsi Nanda, Mukesh Prasad, Saraju P Mohanty, and Albert Y Zomaya. Secure and sustainable load balancing of edge data centers in fog computing. *IEEE Communications Magazine*, 56(5):60–65, 2018.

[PP12] Federica Paganelli and David Parlanti. A dht-based discovery service for the internet of things. *Journal of Computer Networks and Communications*, 2012, 2012.

[PRT05] Marco Pistore, Pierluigi Roberti, and Paolo Traverso. Process-level composition of executable web services:" on-the-fly" versus" once-for-all" composition. In *European Semantic Web Conference*, pages 62–77. Springer, 2005.

[QLW+16] Qi Qi, Jianxin Liao, Jingyu Wang, Qi Li, and Yufei Cao. Dynamic resource orchestration for multi-task application in heterogeneous mobile cloud computing. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 221–226. IEEE, 2016.

[Ray16] Partha Pratim Ray. A survey of iot cloud platforms. *Future Computing and Informatics Journal*, 1(1-2):35–46, 2016.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[RGXZ17] Ju Ren, Hui Guo, Chugui Xu, and Yaoxue Zhang. Serving at the edge: A scalable iot architecture based on transparent computing. *IEEE Network*, 31(5):96–105, 2017.

[RLM18] Rodrigo Roman, Javier Lopez, and Masahiro Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[RLS+11] Katharina Rasch, Fei Li, Sanjin Sehic, Rassul Ayani, and Schahram Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14(4):295–319, 2011.

[RRL+14] M Reza Rahimi, Jian Ren, Chi Harold Liu, Athanasios V Vasilakos, and Nalini Venkatasubramanian. Mobile Cloud Computing: A Survey, State of Art and Future Directions. *Mobile Networks and Applications*, 19(2):133–143, 2014.

[RS04] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *International Workshop on Semantic Web Services and Web Process Composition*, pages 43–54. Springer, 2004.

[RSB+17] Federico Rizzo, Giovanni Luca Spoto, Paolo Brizzi, Dario Bonino, Giuseppe Di Bella, and Pino Castrogiovanni. Beekup: A distributed and safe p2p storage framework for ioe applications. In *2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN)*, pages 44–51. IEEE, 2017.

[RSNK16] Tiago Gama Rodrigues, Katsuya Suto, Hiroki Nishiyama, and Nei Kato. Hybrid method for minimizing service delay in edge cloud computing through vm migration and transmission power control. *IEEE Transactions on Computers*, 66(5):810–819, 2016.

[SCD15] Heng Shi, Nan Chen, and Ralph Deters. Combining mobile and fog computing: Using coap to link mobile device clouds with fog computing. In *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pages 564–571. IEEE, 2015.

[SCZ+16] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE internet of things journal*, 3(5):637–646, 2016.

[SCZY17] Yuvraj Sahni, Jiannong Cao, Shigeng Zhang, and Lei Yang. Edge mesh: A new paradigm to enable distributed intelligence in internet of things. *IEEE access*, 5:16441–16458, 2017.

[SD16] Weisong Shi and Schahram Dustdar. The promise of edge computing. *Computer*, 49(5):78–81, 2016.

[SHHA17] Ronny Seiger, Steffen Huber, Peter Heisig, and Uwe Aßmann. Toward a framework for self-adaptive workflows in cyber-physical systems. *Software & Systems Modeling*, pages 1–18, 2017.

[sig] Hyperic Sigar. "https://github.com/hyperic/sigar/wiki/overview". [Online accessed: January 2020].

[SKG+09] Patrik Spiess, Stamatis Karnouskos, Dominique Guinard, Domnic Savio, Oliver Baecker, Luciana Moreira Sá De Souza, and Vlad Trifa. Soa-based integration of the internet of things in enterprise services. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 968–975. IEEE, 2009.

[SKR+18] Olena Skarlat, Vasileios Karagiannis, Thomas Rausch, Kevin Bachmann, and Stefan Schulte. A framework for optimization, service placement, and runtime operation in the fog. In *2018 IEEE/ACM 11th International*

Conference on Utility and Cloud Computing (UCC), pages 164–173. IEEE, 2018.

[SM12] Audie Sumaray and S Kami Makki. A comparison of data serialization formats for optimal efficiency on a mobile platform. In *Proceedings of the 6th international conference on ubiquitous information management and communication*, page 48. ACM, 2012.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[SNSD17] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards qos-aware fog service placement. In *2017 IEEE 1st international conference on Fog and Edge Computing (ICFEC)*, pages 89–96. IEEE, 2017.

[SQV+14] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218–238, 2014.

[SS21] Olena Skarlat and Stefan Schulte. Fogframe: a framework for iot application execution in the fog. *PeerJ Computer Science*, 7:e588, 2021.

[SSL20] Maria Shehade and Theopisti Stylianou-Lambert. Virtual reality in museums: Exploring the experiences of museum professionals. *Applied Sciences*, 10(11):4031, 2020.

[SSP+15] Chayan Sarkar, Akshay Uttama Nambi SN, R Venkatesha Prasad, Abdur Rahim, Ricardo Neisse, and Gianmarco Baldini. Diat: A scalable distributed architecture for iot. *IEEE Internet of Things journal*, 2(3):230–239, 2015.

[SUS14] Yoshitaka Shibata, Noriki Uchida, and Norio Shiratori. Analysis of and proposal for a disaster information network from experience of the great east japan earthquake. *IEEE Communications Magazine*, 52(3):44–50, 2014.

[SWVDT18] José Santos, Tim Wauters, Bruno Volckaert, and Filip De Turck. Towards dynamic fog resource provisioning for smart city applications. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 290–294. IEEE, 2018.

[TBT17] Genc Tato, Marin Bertier, and Cédric Tedeschi. Designing overlay networks for decentralized clouds. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 391–396. IEEE, 2017.

[TBT18] Genc Tato, Marin Bertier, and Cédric Tedeschi. Koala: Towards lazy and locality-aware overlays for decentralized clouds. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2018.

[THL16] Ling Tang, Shibo He, and Qianmu Li. Double-sided bidding mechanism for resource sharing in mobile cloud. *IEEE Transactions on vehicular technology*, 66(2):1798–1809, 2016.

[TMD19] Christos Tsigkanos, Ilir Murturi, and Schahram Dustdar. Dependable resource coordination on the edge at runtime. *Proceedings of the IEEE*, 107(8):1520–1536, 2019.

[TNL+19] Christos Tsigkanos, Laura Nenzi, Michele Loreti, Martin Garriga, Schahram Dustdar, and Carlo Ghezzi. Inferring analyzable models from trajectories of spatially-distributed internet of things. In *Proceedings of the 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 100–106. IEEE Press, 2019.

[TNT18] Klervie Toczé and Simin Nadjm-Tehrani. A taxonomy for management and optimization of multiple resources in edge computing. *Wireless Communications and Mobile Computing*, 2018, 2018.

[TPGN18] Christos Tsigkanos, Liliana Pasquale, Carlo Ghezzi, and Bashar Nuseibeh. On the interplay between cyber and physical spaces for adaptive security. *IEEE Trans. Dependable Sec. Comput.*, 15(3):466–480, 2018.

[TTT+18] Fan-Hsun Tseng, Ming-Shiun Tsai, Chia-Wei Tseng, Yao-Tsung Yang, Chien-Chang Liu, and Li-Der Chou. A lightweight autoscaling mechanism for fog computing in industrial applications. *IEEE Transactions on Industrial Informatics*, 14(10):4529–4537, 2018.

[UGBM+17] Aitor Urbieta, Alejandra González-Beltrán, S Ben Mokhtar, M Anwar Hossain, and Licia Capra. Adaptive and context-aware service composition for iot-based smart cities. *Future Generation Computer Systems*, 76:262–274, 2017.

[VFD+16] Massimo Villari, Maria Fazio, Schahram Dustdar, Omer Rana, and Rajiv Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3(6):76–83, 2016.

[vL09a] A. van Lamsweerde. *Requirements engineering - from system goals to UML models to software specification*. Wiley, 2009.

[VL09b] Axel Van Lamsweerde. *Requirements engineering: From system goals to UML models to software*, volume 10. Chichester, UK: John Wiley & Sons, 2009.

[WC16] Edward Wang and Richard Chow. What can i do here? iot service discovery in smart cities. In *2016 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–6. IEEE, 2016.

[WIHM18] Danny Weyns, M Usman Iftikhar, Danny Hughes, and Nelson Matthys. Applying architecture-based adaptation to automate the management of internet-of-things. In *European Conference on Software Architecture*, pages 49–67. Springer, 2018.

[YHZ+17] Shanhe Yi, Zijiang Hao, Qingyang Zhang, Quan Zhang, Weisong Shi, and Qun Li. Lavea: Latency-aware video analytics on edge computing platform. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, page 15. ACM, 2017.

[YL14] Zhen Yang and Deshi Li. Iot information service composition driven by user requirement. In *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*, pages 1509–1513. IEEE, 2014.

[ZCX+13] Yawei Zhao, Fei Cai, Junjie Xie, Lailong Luo, Xiaoqiang Teng, Honghui Chen, and Weijie Kong. A new dht supporting multi-attribute queries for grid information services. In *2013 IEEE 10th International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing*, pages 1675–1680. IEEE, 2013.

[ZDC14] Yang Zhang, Li Duan, and Jun Liang Chen. Event-driven soa for iot services. In *Services Computing (SCC), 2014 IEEE International Conference on*, pages 629–636. IEEE, 2014.

[ZFJB18] Alessandro Zanni, Stefan Forsstrom, Ulf Jennehag, and Paolo Bellavista. Elastic provisioning of internet of things services using fog computing: An experience report. In *2018 6th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (MobileCloud)*, pages 17–22. IEEE, 2018.

[ZGG+16] Deze Zeng, Lin Gu, Song Guo, Zixue Cheng, and Shui Yu. Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system. *IEEE Transactions on Computers*, 65(12):3702–3712, 2016.

[ZKJ+01] Ben Yanbin Zhao, John Kubiatowicz, Anthony D Joseph, et al. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. 2001.

[ZM10] Yue-an Zhu and Xiao-hua Meng. A framework for service discovery in pervasive computing. In *Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on*, pages 1–4. IEEE, 2010.

[ZMN05] Fen Zhu, Matt W Mutka, and Lionel M Ni. Service discovery in pervasive computing environments. *IEEE Pervasive computing*, (4):81–90, 2005.