UNIVERSITY OF OULU

FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING

Henna Kokkonen

EDGE INTELLIGENCE SIMULATOR: A PLATFORM FOR SIMULATING INTELLIGENT EDGE ORCHESTRATION SOLUTIONS

Master's Thesis
Degree Programme in Computer Science and Engineering
October 2023

Kokkonen H. (2023) Edge Intelligence Simulator: A Platform for Simulating Intelligent Edge Orchestration Solutions. University of Oulu, Degree Programme in Computer Science and Engineering, 115 p.

ABSTRACT

To support the stringent requirements of the future intelligent and interactive applications, intelligence needs to become an essential part of the resource management in the edge environment. Developing intelligent orchestration solutions is a challenging and arduous task, where the evaluation and comparison of the proposed solution is a focal point. Simulation is commonly used to evaluate and compare proposed solutions. However, there does not currently exist openly available simulators that would have a specific focus on supporting the research on intelligent edge orchestration methods.

This thesis presents a simulation platform called Edge Intelligence Simulator (EISim), the purpose of which is to facilitate the research on intelligent edge orchestration solutions. In its current form, the platform supports simulating deep reinforcement learning based solutions and different orchestration control topologies in scenarios related to task offloading and resource pricing on edge. The platform also includes additional tools for creating simulation environments, running simulations for agent training and evaluation, and plotting results.

This thesis gives a comprehensive overview of the state of the art in edge and fog simulation, orchestration, offloading, and resource pricing, which provides a basis for the design of EISim. The methods and tools that form the foundation of the current EISim implementation are also presented, along with a detailed description of the EISim architecture, default implementations, use, and additional tools. Finally, EISim with its default implementations is validated and evaluated through a large-scale simulation study with 24 simulation scenarios.

The results of the simulation study verify the end-to-end performance of EISim and show its capability to produce sensible results. The results also illustrate how EISim can help the researcher in controlling and monitoring the training of intelligent agents, as well as in evaluating solutions against different control topologies.

Keywords: simulation, edge computing, artificial intelligence, offloading, resource pricing, reinforcement learning

Kokkonen H. (2023) Reunaälysimulaattori: Alusta älykkäiden reunalaskennan orkestrointiratkaisujen simulointiin. Oulun yliopisto, Tietotekniikan tutkinto-ohjelma, 115 s.

TIIVISTELMÄ

Älykkäiden ratkaisujen täytyy tulla olennaiseksi osaksi reunaympäristön resurssien hallinnointia, jotta tulevaisuuden vuorovaikutteisten ja älykkäiden sovellusten suoritusta voidaan tukea tasolla, joka täyttää sovellusten tiukat suoritusvaatimukset. Älykkäiden orkestrointiratkaisujen kehitys on vaativa ja työläs prosessi, jonka keskiöön kuuluu olennaisesti menetelmien testaaminen ja vertailu muita menetelmiä vasten. Simulointia käytetään tyypillisesti menetelmien arviointiin ja vertailuun, mutta tällä hetkellä ei ole avoimesti saatavilla simulaattoreita, jotka eritoten keskittyisivät tukemaan älykkäiden reunaorkestrointiratkaisujen kehitystä.

Tässä opinnäytetyössä esitellään simulaatioalusta nimeltään Edge Intelligence Simulator (EISim; Reunaälysimulaattori), jonka tarkoitus on helpottaa älykkäiden reunaorkestrointiratkaisujen tutkimusta. Nykymuodossaan se tukee vahvistusoppimispohjaisten ratkaisujen sekä erityyppisten orkestroinnin kontrollitopologioiden simulointia skenaarioissa, jotka liittyvät laskennan siirtoon ja resurssien hinnoitteluun reunaympäristössä. Alustan mukana tulee myös lisätyökaluja, joita voi käyttää simulaatioympäristöjen luomiseen, simulaatioiden ajamiseen agenttien koulutusta ja arviointia varten, sekä simulaatiotulosten visualisoimiseen.

Tämä opinnäytetyö sisältää kattavan katsauksen reunaympäristön simuloinnin, reunaorkestroinnin, laskennan siirron ja resurssien hinnoittelun nykytilaan kirjallisuudessa, mikä tarjoaa kunnollisen lähtökohdan EISimin toteutukselle. Opinnäytetyö esittelee menetelmät ja työkalut, joihin EISimin tämänhetkinen toteutus perustuu, sekä antaa yksityiskohtaisen kuvauksen EISimin arkkitehtuurista, oletustoteutuksista, käytöstä ja lisätyökaluista. EISimin validointia ja arviointia varten esitellään laaja simulaatiotutkimus, jossa EISimin oletustoteutuksia simuloidaan 24 simulaatioskenaariossa.

Simulaatiotutkimuksen tulokset todentavat EISimin kokonaisvaltaisen toimintakyvyn, sekä osoittavat EISimin kyvyn tuottaa järkeviä tuloksia. Tulokset myös havainnollistavat, miten EISim voi auttaa tutkijoita älykkäiden agenttien koulutuksessa ja ratkaisujen arvioinnissa eri kontrollitopologioita vasten.

Avainsanat: simulaatio, reunalaskenta, tekoäly, laskennan siirto, resurssien hinnoittelu, vahvistusoppiminen

TABLE OF CONTENTS

FOREWORD

This thesis has been supported by the ECSEL JU FRACTAL (grant 877056), receiving funding from the EU Horizon 2020 programme and Business Finland. I would like to express my gratitude to the FRACTAL project for supporting this thesis, as well as thank my supervisors Dr. Lauri Lovén and Doc. Susanna Pirttikangas for excellent guidance.

Oulu, October 11th, 2023

Henna Kokkonen

LIST OF ABBREVIATIONS AND SYMBOLS

AI	Artificial Intelligence
AP	Access Point
API	Application Programming Interface
BS	Base Station
CRB	Computing Resource Block
CSP	Cloud Service Provider
DDPG	Deep Deterministic Policy Gradient
DES	Discrete Event Simulation
DNN	Deep Neural Network
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
EISim	Edge Intelligence Simulator
ESP	Edge Service Provider
FIFO	First In First Out
GUI	Graphical User Interface
IID	Independent and Identically Distributed
IoT	Internet of Things
LAN	Local Area Network
MAN	Metropolitan Area Network
MARL	Multi-Agent Reinforcement Learning
MAS	Multi-Agent System
MDP	Markov Decision Process
MEC	Multi-access Edge Computing
MI	Million Instruction
MIP	Mixed Integer Programming
MIPS	Million Instructions Per Second
ML	Machine Learning
NUM	Network Utility Maximization
OS	Operating System
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RL	Reinforcement Learning
RRB	Radio Resource Block
RSU	Road Side Unit
SLA	Service Level Agreement
SLO	Service Level Objective
TD	Temporal Difference
TWST	Tunable Weight Spanning Tree
VEC	Vehicular Edge Computing
VM	Virtual Machine
VNF	Virtual Network Function
WAN	Wide Area Network

$\mathcal{A}$	action space
a	action
B	mini-batch size
B_e	battery level of an edge device
$\mathcal{C}$	set of edge servers in a cluster
C	confidence level
c	computational demand of a task
$\mathcal{D}$	experience replay
D	task execution delay
D_{max}	maximum tolerable delay of a task
d	degree of an AP node
d_{in}	input data size of a task
d_{out}	output data size of a task
E	energy consumption
f	computational capacity of one CPU core
$\mathcal{I}$	index set for AP nodes
J	actor objective function
K	number of edge server clusters
L	critic loss function
l	task queue length on an edge server
l^{avg}	average task queue length in a cluster
$\bar{m}$	mean of an evaluation metric
$\mathcal{N}$	noise process
N	number of edge servers
n_{add}	number of links to add to TWST
n_{AP}	number of APs
n_e	number of episodes
n^c	number of cores in CPU
$\boldsymbol{P}$	permutation array
P_{idle}	power consumption when CPU is idle
P_{max}	power consumption when CPU is at 100%
P_t	transmission power of an edge device
P_r	receiver power of an edge device
$\bar{\boldsymbol{p}}$	average location of AP nodes
$\boldsymbol{p}$	location of an AP node
p	price
p_{pref}	preferred price of an edge device
Q^{MI}	total number of MIs summed over a set of tasks
Q_{π}	Q-function
R	reward function
r	reward
r^d	downlink transmission rate
r^u	uplink transmission rate
$\mathcal{S}$	state space
s	state
s_m	sample standard deviation of an evaluation metric
T	transition probability function

$t^*_{n_e-1}$	upper critical value for Student's t-distribution with $n_e - 1$ degrees of freedom
V_π	value function
w_d	importance weight for task execution delay
w_e	importance weight for energy consumption
w_p	importance weight for monetary cost
$\boldsymbol{x}$	vector of binary offloading decision variables
y	TD target for critic
i	AP node index
j	edge server index
k	cluster index
t	time slot index
α	weight parameter for adding links to TWST
β	weight parameter for adding links to TWST
γ	discount factor
ζ_e	energy cost coefficient
η	weight parameter for adding links to TWST
$\boldsymbol{\theta}^\mu$	actor parameter vector
$\boldsymbol{\theta}^{\mu'}$	target actor parameter vector
$\boldsymbol{\theta}^Q$	critic parameter vector
$\boldsymbol{\theta}^{Q'}$	target critic parameter vector
ι	time slot length
κ	weight parameter for TWST
λ	arrival rate of tasks
μ	deterministic policy function
μ'	exploration policy function
ν	weight parameter for edge server placement
π	stochastic policy function
τ	parameter for updating target networks

1. INTRODUCTION

1.1. Background

The future entails a myriad of novel, interactive and intelligent applications in areas such as autonomous vehicles, smart city, healthcare and Industry 4.0 (see, e.g., [1, 2, 3]). These applications have high, ever-growing requirements in terms of security, reliability and performance. For example, applications can process highly sensitive personal data which must not fall into the hands of unauthorized parties, intelligent applications should ensure a certain level of accuracy in their forecasts, and many applications have critical, ultra low latency requirements, the violation of which can lead to dire repercussions.

Currently, the development of interactive and intelligent applications is heavily dependent on cloud, the abundant computing and storage resources of which are a necessity for the computationally intensive Artificial Intelligence (AI) methods. However, cloud-native processing requires transmitting data and results between the end users and the cloud, which naturally increases the latency and raises privacy concerns. Further, the amount of data transmitted to the cloud can be enormous, which overburdens the network and is far from an energy efficient solution. Hence, several computing paradigms, such as edge and fog computing, Multi-access Edge Computing (MEC) and cloudlet-based computing [4], have emerged to bring the computing and storage resources from the cloud *to the edge*, closer to the end users. Even though these paradigms have differences in their architectural considerations and driving forces, they all have the same essence: placing and using computational resources between the end user and the distant cloud in order to reduce latency and energy consumption, as well as increase security and privacy by keeping the application data local.

Bringing the intelligent applications onto the edge resources between the end users and the cloud is not a simple task. Traditional AI is inherently centralized and resource consuming, while the edge is inherently distributed and limited in resources compared to cloud. Further, the edge nodes are highly heterogeneous in terms of their operating principles and capabilities, while the edge environment as a whole is characterized by intermittent connectivity, distributed and non-IID (Independent and Identically Distributed) data, as well as geographically distributed, opportunistic computing resources [5]. Research on developing and adapting AI methods to the edge environment has been coined as *AI on Edge* [6, 7], which is an active research area with an ample amount of research [7, 8, 9, 10, 11]. However, in order to fully realize the envisioned interactive and intelligent applications, the *orchestration* of the edge resources must be intelligent as well.

The research on edge orchestration has been very dispersed as the studies usually focus on the orchestration of only one aspect of the whole environment, such as networks, containers or services [5, 12, 13, 14, 15]. Further, the orchestration paradigms of these aspects, such as container orchestration, typically follow the essence of traditional cloud orchestration, which uses centralized, best-effort and reactive orchestration techniques [5]. However, the fulfillment of the stringent requirements of the future applications requires that the orchestration enables adaptive, context-aware and autonomous behavior on the edge. This requires distributed, proactive and optimized orchestration solutions that are built upon distributed intelligence. The research on developing and applying distributed AI methods for performing orchestration functions has been coined as *AI for Edge* [6, 7], and it has not received as much attention as *AI on Edge* in the context of *edge intelligence* research [5].

The lack of a holistic view on edge orchestration has been a significant deficit in terms of developing novel, distributed orchestration solutions. Only recently there has been efforts to piece together different aspects on orchestration in order to create a more holistic view. Kokkonen et al. [5] present an early vision for the future of edge orchestration. The vision relies on a more encompassing view on resources in the *computing continuum* that spans from the end devices all the way to the distant cloud. The architecture of the computing continuum is envisioned as a Multi-Agent System (MAS) consisting of autonomous, intelligent, and self-interested agents. Agents manage resources in the computing continuum and aim to fulfill externally set objectives on cost, quality and resource usage. Each agent has local autonomy when it comes to deciding on when and how to conduct actions related to orchestration functions.

The autonomous agents form a hierarchy where higher levels control lower levels by setting their objectives and constraints. Each administrative domain (e.g., application providers, infrastructure and platform providers) is represented by a stakeholder agent at the highest level, under which other resource agents inside the same domain are organized as clusters and further sub-clusters. Such a hierarchy enforces local decision making and follows the idea of loose coupling [16, 17] with a minimal amount of centralized control.

The vision states that through *AI for Edge*, that is, the development of intelligent solutions for distributed, multi-domain and multi-tenant edge orchestration, the edge environment will eventually evolve into a coherent, autonomous computing continuum. The continuum will be able to orchestrate its limited resources in a globally optimized manner while being aware of and ready to adapt to the dynamic environment. Naturally, realizing such a vision of orchestration built upon MAS paradigm, distributed AI, local autonomy and loose coupling is an extremely challenging endeavor.

1.2. Contribution

The realization of the aforementioned vision brings forth multiple open research questions. Some of the most important questions concern the optimal level of autonomy in the system and the adaptation and application of AI methods in the challenging computing continuum environment. Any proposed solution requires ways to test and evaluate the method and compare it with other potential solutions.

Simulation is a commonly adopted way for evaluating proposed methods, as it provides a controllable and cost-effective testing environment. However, there does not currently exist openly available simulation platforms that would particularly support research on intelligent edge orchestration methods.

This thesis presents a simulation platform called *Edge Intelligence Simulator* (EISim), the purpose of which is to facilitate research on intelligent edge orchestration solutions particularly in the context of the aforementioned vision. EISim is built on top of an existing fog simulator called PureEdgeSim [18], extending it towards supporting the easier testing and evaluation of intelligent orchestration methods.

Intelligent orchestration methods in this thesis particularly refer to (Deep) Reinforcement Learning ((D)RL) based solutions. This is because reinforcement based learning is seen as one key ingredient for intelligent computing continuum orchestration in the vision [5], mainly due to its ability to learn from experience decision-making policies that can adapt to complex systems and achieve long-term optimization.

In its current form, EISim supports simulating scenarios related to task offloading and resource pricing. Task offloading is supported, because it is the core function of PureEdgeSim. Resource pricing, in turn, has a crucial role due to the multi-domain nature of the computing continuum, which means that there are many different stakeholders, such as end device owners, infrastructure owners and application providers. The stakeholders offering their resources for the computational demand of other stakeholders naturally want to profit from this exchange.

The focus of EISim is particularly on evaluating and comparing the performance of orchestration solutions against different orchestration control topologies. This is because the control topology of the orchestration solution closely relates to the level of autonomy in the system, and the optimal level of autonomy is a central research question in the aforementioned vision. Hence, to facilitate the research on the optimal level of autonomy, EISim offers three default task orchestration algorithm implementations that correspond to the three main control topologies, namely *decentralized*, *hybrid* and *centralized*. It is important to note that in its current form, EISim focuses on edge-based processing, meaning that tasks are either processed locally by the edge devices that generate them, or offloaded to the edge servers. Default implementations with wider focus are left for future work.

Decentralized control topology regards every edge device and server in the system as a self-interested, autonomous agent. Each device agent aims to maximize its own utility which takes into account task execution latency, energy consumption and monetary cost of the execution on the edge platform. Each edge server agent aims to optimize its resource usage and maximize its profit, with the high level objective of maximizing the profit of the Edge Service Provider (ESP). In this control topology, every server decides about the price for task execution on its resources, and devices decide whether to offload their tasks and to which server.

In the hybrid control topology, edge servers are clustered into groups with assigned cluster heads. A cluster head agent decides the price for task execution in the cluster, and each device agent decides to which cluster it offloads the tasks. The cluster head agent also decides how the incoming tasks are allocated on the cluster nodes.

In the centralized control topology, there is one central edge server agent that decides the price for the task execution on the platform, as well as the allocation of all offloaded tasks. Device agents only decide whether they offload or not.

The decentralized control topology has the most autonomous agents, as both servers and devices are able to decide about the pricing and offloading targets independently. The hybrid control topology introduces a level of control where cluster heads decide independently for the price and task allocation inside their clusters. This reduces the autonomy of other edge servers inside the clusters, as well as that of the devices as they can no longer decide about the exact location for the execution of their tasks, although they still have partial autonomy with regard to deciding the destination cluster. The centralized control topology has the least amount of autonomy, as all edge servers hand over the decision-making power to the central orchestrator, and devices lose almost completely their say in where their tasks are executed.

For investigating resource pricing methods, EISim allows researchers to plug in their own pricing algorithms for servers. The implemented default algorithm is Deep Deterministic Policy Gradient (DDPG) [19], which is a state-of-the-art DRL algorithm for continuous action spaces.

To further facilitate the research, EISim offers tools for environment setup, agent training and result plotting. The environment setup tools consist of three tools that can be used to automatically generate 1) Metropolitan Area Network (MAN) (including Access Point (AP) placement, topology creation and edge server placement), 2) edge server clusters for the hybrid control topology, and 3) edge server specification files that are given as input to EISim. Agent training tools are a set of scripts that can be used to run hyperparameter tuning, model training and model evaluation simulations for the pricing agents. Finally, result plotting tools generate plots for analyzing the results of the simulations for each phase (hyperparameter tuning, training and evaluation).

Overall, EISim adds into PureEdgeSim the capabilities to simulate intelligent, DRL-based solutions and resource pricing, as well as pre-implemented algorithms for the three main orchestration control topologies. EISim also offers a more versatile application model, improves the extensibility of PureEdgeSim, and enables the reproducibility of the simulation results. Further, the additional tools of EISim are an exclusive feature, meaning that they do not exist in any form in PureEdgeSim.

EISim is validated and evaluated through a large-scale MEC simulation case study that verifies the end-to-end performance of EISim in 24 simulation scenarios. The fidelity of EISim is analyzed through the sensibility of the simulation results.

The rest of the thesis is organized as follows. Chapter 2 defines the key terms used in this thesis, and provides an overview of the state of the art in edge and fog simulation, orchestration, task offloading and resource pricing. Chapter 3 describes the methods and tools used in the current EISim implementation. Chapter 4 describes the implementation details of EISim and presents the evaluation scenarios used to validate EISim. Chapter 5 presents and analyzes the evaluation results. Chapter 6 discusses the significance, limitations and future development directions of EISim. Finally, Chapter 7 concludes the thesis.

2. LITERATURE REVIEW

2.1. Definitions

The meaning of many terms especially in the edge computing related literature can be very ambiguous because some terms may be used interchangeably or differentiated in meaning depending on the context. Further, there can be different notions of the meaning of some general terms such as autonomy. The following sections define how certain key terms are understood in the context of this thesis.

2.1.1. General Terminology

An agent is a sense-decide-act feedback loop. An agent uses sensors to perceive the state of its environment, after which a decision-making process is deployed to find an action that maximizes some externally set performance measure [20]. An agent then acts upon the environment through actuators.

Autonomy is defined as in [5]: "Autonomy refers to an agent's ability to make decisions without any influence of external authority such as users, administrators or other agents. While such decisions can be made by the agent on the behalf of an external authority, that authority itself cannot influence a fully autonomous agent making the decision."

Fog computing refers to "a highly virtualized platform that provides compute, storage, and networking services between end devices and traditional Cloud Computing Data Centers, typically, but not exclusively located at the edge of network", as first defined by Cisco in 2012 [21]. The term has since received many alternative definitions [22], but the driving force behind the definitions for fog computing is typically the Internet of Things (IoT). In this thesis, fog computing is understood as a hierarchical view on the computing, storage and networking resources between the end devices and cloud, that is, it includes the immediate edge (lowest level in the hierarchy) between end devices and cloud but also includes the resources deeper into the network (higher levels) from the edge towards the cloud.

Edge computing does not have a standardized definition, it can be understood differently based on the context. For example, edge computing may be regarded as an all-encompassing term that includes paradigms such as fog computing, MEC and cloudlet-based computing [23, 24], or it may be seen to be interchangeable with the term fog computing [25], or it may be regarded as its own paradigm with its own focus alongside fog computing, MEC, etc. [26]. In this thesis, edge computing is understood to encompass any paradigm that specifically uses computing, storage and networking resources on the lowest level (the edge) between end devices and cloud. The location of this edge may vary based on the paradigm and use case. For example, in IoT, smart phones can belong to the edge as they can act as the IoT gateways between controllers / sensors and cloud. On the other hand, in MEC, smart phones are mobile devices that belong to the end device level, and the edge is located on the Radio Access Network (RAN) edge, the resources of which serve the needs of mobile applications.

Computing continuum refers to the visionary view of the seamless integration of the distributed resources available on all computing tiers (device, edge / fog, cloud) in a way that the demanding and varying workflows of the future applications can be efficiently supported [27, 28]. Hence, the term computing continuum can also be regarded to encompass any distributed computing approach that deals with heterogeneous and opportunistic resources [5].

A resource is defined in the same encompassing way as in [5]. A resource refers to any type of communication, computation, or data related entity in the computing continuum that may reside on any of a number of abstraction layers. These resource layers include the following:

- fundamental resources (materials, energy, information, time, frequency, space)
- cyber-physical resources (sensors, actuators, connected devices)
- hardware resources (e.g., physical network interfaces, processing units and storage units)
- Operating System (OS) resources (e.g., OS network interfaces and OS services)
- middleware resources (e.g., software-defined networking, middleware Application Programming Interfaces (APIs) and container frameworks)
- application resources (e.g. Virtual Network Functions (VNFs), application APIs, services, and application data)
- workflow resources (data flows, tasks, and data sets available for individual application workflows)

Orchestration can be holistically defined as “the management of resources in the computing continuum, from fundamental resources to workflow resources” [5] by adopting the above encompassing definition for a resource.

Offloading is one orchestration function, where software code with related input data and necessary system settings is transported to another node for processing [29]. The transportation is typically triggered by heavy workload or lack of resources on the sending node. After the offloaded code has been executed, the results are sent back to the offloading requester. Offloading can happen either horizontally (offloading to neighbouring nodes on the same level) or vertically (offloading to edge / fog or cloud).

Simulation is defined as “the imitation of the operation of a real-world process or system over time” [30]. It uses a model that represents the behavior and characteristic of the system to generate an artificial history of the system. The artificial history is then used to describe and analyze the behavior of the system. The main purpose of simulation is to help in understanding, designing and improving real systems.

2.1.2. Control Topologies

Control topology refers to the degree of centralization in orchestration solutions. It can be understood from two perspectives.

The first perspective considers the degree of centralization in the interaction or coupling between decision-making agents [17, 31]. In this regard, control is centralized if there is a leader-follower type of hierarchy where a strong central agent coordinates the lower level agents and has a global view on the system. This naturally leads to high decision making delays and exponential growth of the information needed to control the system. On the other hand, control is regarded to be decentralized, if each agent makes decisions with complete local autonomy without interacting with other agents. Such decentralized control could be realized, if, for example, the agents are geographically far apart. Finally, control is regarded to be distributed, if the decision makers at least exchange information with their nearest neighbours. In both decentralized and distributed control, there is no hierarchy of agents. In this perspective, hybrid control can be regarded to be a combination of the central and distributed control, where agents form a hierarchy with a weak central agent and nearly autonomous lower-level agents. Hybrid control covers all other control types, as the strength of the coupling between the agents determines the degree of centralization.

The second perspective considers the orchestration control, that is, what is the degree of centralization in decision making [32, 33]. In this perspective, the control is centralized if there is one central orchestrator agent in the system that has the decision-making power over the resources in the system. On the other hand, control is called decentralized if there are groups of nodes with assigned local leaders that have the decision-making power over the resources in the group. Local leaders can coordinate decision making and communicate with each other through peer-to-peer protocols. Finally, control is called distributed if there is no central or leader nodes. In this case, each node controls its own resources and can coordinate its decision making and communicate with other nodes in any way. It is also good to note that there might be discrepancies in the literature with regard to how the terms distributed and decentralized are used. In some cases, distributed can be used to refer to the control topology where there are groups of nodes with local leaders, whereas decentralized is used to refer to the control topology with no leaders.

In this thesis, control topologies are divided into three main categories of centralized, hybrid and decentralized. The definitions for these combine elements from the two perspectives introduced above, but the emphasis in the categorization is on the level of autonomy the control topology endows to the local system entities.

The control topology of an orchestration solution is centralized if the system has a master-worker type of control hierarchy, where one central agent either 1) makes all the decisions on behalf of the system entities, or 2) strongly coordinates the decision making in the system. The first aspect considers the type of system models that only have one agent that collects sensing information from other system entities to make all the decisions related to the system resources. In such models, other system entities are not agents as per the definition in Section 2.1.1. Hence, there is a complete lack of local autonomy. The second aspect, in turn, considers system models where the other system entities are also agents, but their decisions are heavily controlled by a central agent. The heavy control can be realized, for example, in the form of an iterative algorithm, where the central agent and the lower-level agents exchange and update their decisions until the central agent determines that an equilibrium solution has been reached. In such a case, other agents lack local autonomy as their final decisions are dictated by the central agent.

In the decentralized control topology, the agents in the system do not form any type of control hierarchy. In other words, the system model considers the system entities as fully autonomous agents that have the final decision-making power over their local resources. This definition does not limit the interaction patterns between the agents, meaning that the agents can interact with other agents to coordinate the decisions in any way. For example, the agents could deploy negotiation protocols to reach consensus on optimal resource allocation.

The hybrid control topology covers any type of middle form between the centralized and decentralized extremes. This means that the system model involves some type of control hierarchy between the agents. In other words, there are some elements of centralized control in the system, but there are also system entities that can make their decisions in a nearly or fully autonomous manner. For example, hybrid control topology is realized in a system with several central controllers. These controllers decide for the system entities in their group, but the controllers themselves make their decisions with local autonomy, possibly coordinating with other controllers through negotiation protocols. As another example, hybrid control topology is also realized in a system with one central agent that operates on a slower time-scale than the lower-level agents and influences the decision making of the lower-level agents by setting their goals and constraints. In this case, the lower-level agents are nearly autonomous, because the higher-level agent does not dictate the decisions of the lower-level agents, but rather intends to direct their decision making towards a global optimum.

2.2. The Simulation of Edge and Fog Environments

2.2.1. Motivation for Simulation

Any method developed for the orchestration of the edge or fog environment requires evaluation. Ideally, solutions could be tested and evaluated in a real-life environment. This would require either the use of commercial edge or fog computing platforms or building a testbed. However, commercial edge or fog platforms are scarce. Further, commercial platforms do not usually grant a full access to their infrastructure, nor support research-originated heavy workflows that may disturb the other users of the platform. Building an edge or fog testbed, in turn, is an expensive option which results in a test environment that cannot be easily scaled. Hence, the best alternative for performance evaluation is simulation, which allows to evaluate the performance of the developed method in a repeatable and controllable manner.

Even in cases where a real-life environment is available for testing, it is usually worthwhile to also use simulation for the development and evaluation of the method. Simulation can bring significant cost and time savings, as the simulation environment can be effortlessly modified into a completely new configuration, and simulations can be run significantly faster than the wall-clock time. Further, because a real environment is not as controllable as a simulation environment, it is a lot more difficult to investigate in real environments why some phenomenon occurs in the system. In a simulated environment, a microscopic examination of the system can be carried out to determine the cause of the phenomenon [30]. Simulation models can also provide a better understanding of the system variables and their relationships, which provides insight into the importance and effect of different variables on the system performance.

2.2.2. Building Simulation Models

General Workflow

The general steps of a simulation study are shown in Figure 1. This workflow is presented in accordance with the work of Banks et al. [34].

Any simulation project starts with properly stating and formulating the problem underlying the study, which includes representing the problem in a way that others can understand it. The next step is to set the objectives of the project. These objectives indicate the questions to which the simulation project aims to answer. Along with setting the objectives, the project plan should be set, which includes, for example, describing the scenarios that are to be investigated, deciding the timetable of the project, and specifying the hardware and software requirements.

After the project foundation has been set, the next step is to start the simulation model building. This process starts off with the building of a conceptual model that is an abstraction of the real system. A conceptual model is defined as "a non-software specific description of the simulation model that is to be developed, describing the objectives, inputs, outputs, content, assumptions and simplifications of the model" [35]. The content of the simulation model is specified in terms of what system components are modelled and to what level of detail, as well as how different components are interconnected.

Data collection should be done in parallel with the simulation model building. This refers to collecting data of the behavior and characteristics of the real system. In general, three types of data should be collected [35]. Some preliminary or contextual data is needed in starting the model building process as it helps to develop understanding of the system. The conceptual model itself sets requirements for the type of data that should be collected in order to realize the conceptual model. Finally, data from the real system is needed to validate the conceptual model.

The next step after conceptual modelling is model translation. This refers to building an operational model that is a computer-based realization of the conceptual model. An operational model can also be called a computer model.

Model verification is the process of ensuring that a conceptual model has been transformed into an operational model with sufficient accuracy. Model validation, in turn, is the process of ensuring that the simulation model represents the real system with sufficient accuracy. 'Sufficiency' is defined with respect to the purpose of the model. Verification has a lot narrower scope than validation, as it focuses on the implementation of the conceptual model as a computer program. Hence, it can be seen as a subset of the wider problem of model validation [35]. Model validation and verification are fundamental steps in a simulation project, as they determine the level of confidence that can be placed on the simulation results.

In the experimental design step, the length of one simulation run, the number of simulation runs, and the method of simulation initialization are decided for each simulation scenario. How these factors are selected is important for obtaining accurate results from the simulation model [35]. Experimental design typically also addresses how different simulation scenarios are compared and how new simulation scenarios are selected.

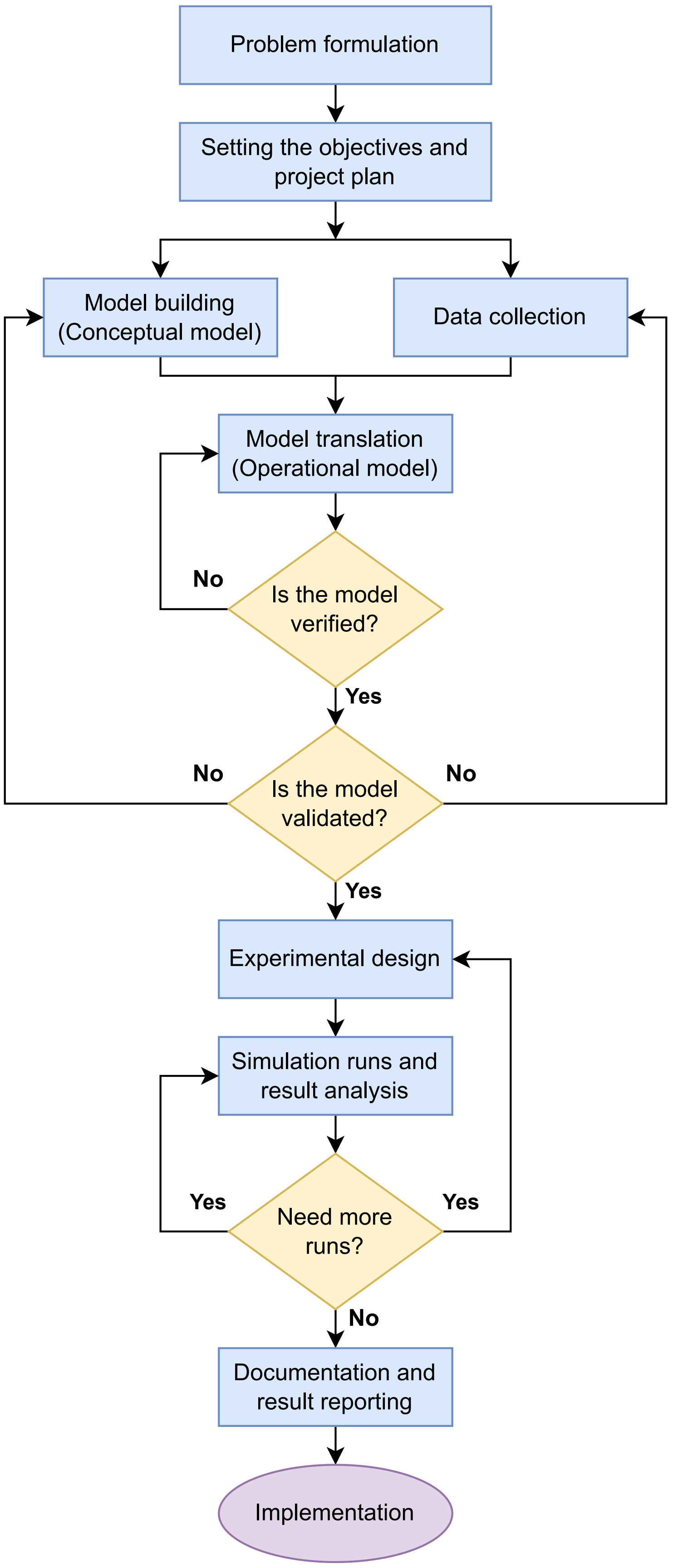

Problem formulation
Setting the objectives and project plan
Model building (Conceptual model)
Data collection
Model translation (Operational model)
No
Is the model verified?
Yes
No
Is the model validated?
No
Yes
Experimental design
Simulation runs and result analysis
Yes
Need more runs?
Yes
No
Documentation and result reporting
Implementation

Figure 1. The general workflow of a simulation project. Adapted from [34].

After experimental design, simulation scenarios are run and results analyzed. Based on the result analysis, it may be necessary to do more simulation runs to get more accurate results. It may also be necessary to redesign the simulation experiments by, for example, adding more simulation scenarios or changing the initialization method.

Finally, documentation is another crucial part of a simulation project. What needs to be documented and how depends largely on the project, but, in general, there should be three types of documentation [35]: project documentation, model documentation and user documentation. Project documentation typically describes, for example, the project specification, the model verification and validation performed, as well as the experimental scenarios and their results. Model documentation typically includes comments on the code, a description of the model structure, and a specification of the model inputs and outputs and their formats. User documentation is meant for those who want to use the simulation model. It should include a guide to running the simulation model, as well as similar descriptions of the project and the model that are included in the project and model documentation.

A simulation project results in some type of implementation in a real world. Implementation can be understood to refer to something tangible, such as implementing the changes proposed by the simulation model in the real system, or handing the simulation model over to others to use. However, it can also be understood from a more abstract perspective [35]. In this perspective, implementation is seen as learning. In other words, one result of a simulation project is the improved understanding of the real system, which is gained from the whole process of developing and using the simulation model.

It is important to note that even though this section presents a simulation project as a straightforward process of steps, this process is actually highly iterative. For example, it is advisable to start with a very simple conceptual model that is gradually improved. The need for modifications to this model can arise at any point of the project, such as during model translation, validation or experimentation. As another example, model verification should be tightly connected to the model translation, meaning that the operational model should be coded and verified in small parts. Further, model validation should be done throughout the project.

Validating Simulation Models

Model validation is vital for qualifying the utility of a simulation model. It aims to determine how accurately the simulation model represents the real system and whether the level of accuracy is sufficient for the purpose of the simulation model. Figure 1 is a little misleading when it comes to the validation of a simulation model as it presents validation as one step of a simulation project workflow. Validation, however, should be intimately connected with the whole life-cycle of a simulation project. Figure 2 shows the process of model validation as a part of a simulation project life-cycle. The validation process is presented in accordance with the work of Robinson [35].

Conceptual model validation determines whether the content, assumptions and simplifications of the conceptual model are sufficiently accurate for the purpose of the simulation model. There are no formal methods for validating a conceptual model [35]. Hence, conceptual model validation typically relies on the knowledge and the assessments of system experts.

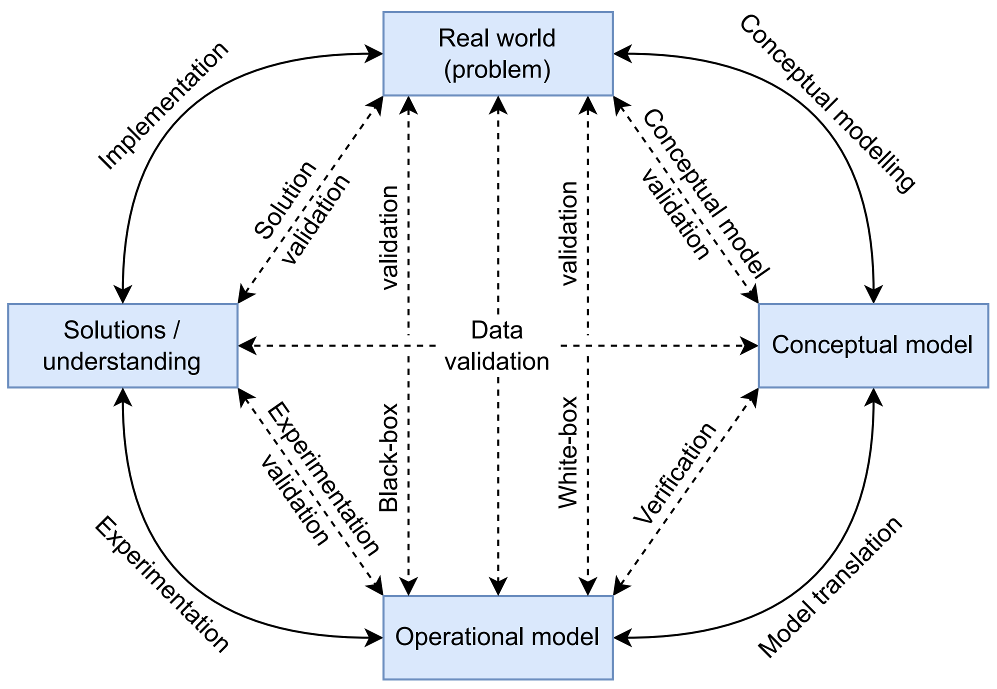


Figure 2. The process of validating a simulation model. Adapted from [35].

White-box validation determines whether the components of the operational model represent the corresponding real world components with sufficient accuracy. This is similar to verification, as both are micro-level checks of the model. However, white-box validation compares the operational model to the real system, whereas verification compares the operational model to the conceptual model. Verification can be done by the modeller alone. White-box validation, on the other hand, requires the involvement of system experts.

Black-box validation determines whether the overall simulation model represents the real system with sufficient accuracy. It is a macro-level check of the model, which is done by comparing the output of the simulation model with the output of the real system under same conditions. Black-box validation could also be done against another simulation model of the same or a similar real system.

Data validation determines whether the contextual data and the data collected for the conceptual model realization and the model validation are sufficiently accurate. This includes assessing the reliability of data sources and inspecting the data for any inconsistencies and errors.

Experimentation validation determines whether the experimental procedures provide results that are sufficiently accurate. This includes removing possible initialization bias, determining appropriate simulation run-length and number of runs, and using suitable methods for finding new simulation scenarios.

Finally, solution validation can only be done after the simulation project has reached the implementation state. It compares the final model of some proposed solution to the implemented solution and determines whether the model of the proposed solution is sufficiently accurate. In practice, solution validation is rarely done because it can take several years to implement findings from a simulation study [35].

Model validation is an arduous and time-consuming process. Often there is not enough resources to conduct thorough model validation. Further, it is impossible to prove that a simulation model is absolutely correct [35]. Hence, validation processes do not generally try to prove that the model is correct, they try to prove that it is *incorrect*. In this sense, the more tests are done in which it cannot be proved that the simulation model is incorrect, the more confidence can be placed on the model. After all, it is the level of confidence that can be placed on the model that determines whether the simulation model can be used for decision making.

When it comes to validating simulation models of edge and fog environments, a significant problem is the lack of actual real-life deployments [36]. In other words, no comparable data from real deployments is readily available. Hence, the validation of edge and fog simulation models is typically done by investigating the models in scenarios that correspond to real-life use cases. The results of these scenarios are compared against the expectations and intuitions of those who have detailed understanding of the system. This is also the approach adopted in this thesis. Further, as more testing over a variety of use cases increases the confidence on the model and is crucial for validation, fog and edge simulators typically branch off from existing, well-known cloud or edge / fog simulators (see Section 2.2.3). This is also why EISim is built on top of an existing simulator.

Types of Simulation Models

In general, two types of simulation models can be recognized in the edge and fog computing literature, namely *static* and *dynamic* models.

Static models are highly abstract, analytical models that focus on describing the system using mathematical equations [37]. These models can focus on modelling the system at some fixed time instant or over a time period, but they do not model the change in the system state over time. In other words, they focus on analyzing the system under static conditions, where typically the number and location of the nodes does not change and many important characteristics of the system, such as the arrival rate of tasks, the transmission time per packet, task size, the processing time per task, or the resource utilization level of the servers, are treated as constant means that are input into the model (see e.g., [38, 39, 40, 41]).

Due to their analytical nature, static models rely strongly on simplifying assumptions that sometimes can be very invalid [37]. For example, it may be assumed that the distribution of Virtual Machines (VMs) is uniform over the computing nodes [38] or that all the tasks can be processed within a time frame [41]. However, static models also have some benefits. They are fast to execute and provide precise output in a sense that the response to the same input is always the same. They can be beneficial in providing a high-level, overall view on the performance of the system under different architectures, but they are not suitable for analyzing online decision-making algorithms, where a decision in one instant of time can have long-term impacts on the system performance.

Dynamic models are focused on modelling changes in the system state over time. Such models are more suitable for analyzing online decision-making algorithms, and they are also what is commonly known as ‘simulators’ [37]. There are two approaches to modelling the progression of time in dynamic models, namely time-driven and event-driven. In time-driven simulation, the system state changes at equal time intervals during a simulation run. This is also known as continuous simulation because it is used to simulate systems where the system state changes continuously over time [35]. Event-driven simulation, in turn, is known as Discrete Event Simulation (DES). In DES, the system state changes only at those discrete points in time at which events occur [30]. In edge and fog environments, such events could be, for example, the arrival of a task, the start of the task execution, and the end of the task execution.

Purely time-driven simulation can be very inefficient, as there can be several time steps during which no change happens in the system. Determining an appropriate time interval is also a challenge, because smaller intervals slow down the simulation, whereas longer intervals compromise the accuracy [35]. In edge and fog computing, event-driven simulation is the prevalent approach. For example, all the simulators in Section 2.2.3 are based on DES. However, time-driven and event-driven approaches are not mutually exclusive, they can be combined. For example, DES-based edge and fog simulators can include an event that happens at fixed intervals and updates the continuous state variables, such as the energy consumption and the location of the devices (e.g., in PureEdgeSim [18]).

Dynamic models generally include randomness. For example, the arrival times and lengths of the tasks and the mobility paths of the devices are often generated randomly. Hence, the same inputs to the model generate different outputs, which means that the simulation must be run several times under the same conditions and the results aggregated in order to analyze the system.

2.2.3. Edge and Fog Simulators

A plethora of edge and fog simulators have been developed in the research community [42, 43, 44, 45]. Because of the complexity of the edge and fog environments, it is very difficult to develop a simulator that could simulate every factor affecting the performance realistically. Hence, simulators rely on abstractions that simplify the environment. To which degree different levels in the environment (e.g., networking, OS, middleware or application level details and protocols) are abstracted out depends on the intended use of the simulator.

The following gives a short comparison of different simulators that were considered to serve as the basis of EISim. The simulators were selected based on two factors: 1) the simulator is an edge or fog simulator, that is, it simulates processing on edge or fog nodes, and 2) the software of the simulator is open source.

iFogSim is one of the most referenced simulators in the literature [46, 42]. It builds upon a popular cloud simulator CloudSim [47], and is designed to simulate application placement and scheduling in a fog environment, particularly in IoT use cases with sense-process-actuate or stream processing application models. One application is modelled as a directed acyclic graph, where the nodes are application components and the edges correspond to data dependencies. The placement policy determines how application components are placed on fog nodes, whereas the scheduling policy determines how the resources of a fog node are divided among all the application components that have been placed on the node. During simulation, IoT devices generate tuples that are processed by the application components placed on the fog nodes.

iFogSim has several deficiencies. It can only simulate centralized application placement policies, as the placement policy is run by a broker that has knowledge of the whole fog environment and the resource availability of each fog node. Further, it does not support simulating runtime load-balancing algorithms, does not have any type of mobility model for the devices, nor simulates the possible failures of links and nodes. It only supports tree topologies, meaning that communication between nodes on the same level is not possible. The network model is also very simplistic, because it assigns a fixed latency on each link and does not consider the effect of network load on transmission delays.

Extensions to iFogSim that aim to address some of the deficiencies have been developed. MyiFogSim [48] adds in mobility support through migration of VMs and containers between cloudlets. It adds two customizable functions to iFogSim: migration policy and migration strategy. Migration policy determines when a user's VM should be migrated considering the user's speed, direction and geographical location. Migration strategy determines where the user's VM is migrated and how the migration is performed.

MobFogSim [49] is a refined version of MyiFogSim. It allows more generalized VM and container mobility simulations. The major difference with regard to MyiFogSim is the addition of more realistic user mobility patterns: MobFogSim supports customized user mobility patterns that can be given as input data. Besides adding in mobility and migration support, neither MyiFogSim nor MobFogSim addresses any other deficiencies in iFogSim.

iFogSim2 [50] redefines many core components of iFogSim, such as fog devices, application modules, sensors and actuators, and adds in support for mobility, application migration, dynamic cluster formation, and microservice orchestration. It allows implementing customized migration policies, distributed cluster formation algorithms, as well as microservice placement and scheduling methods. It supports a more realistic mobility model, where customized mobility patterns can be read from input files. Further, it allows implementing decentralized runtime load-balancing algorithms, as the node that generates a service request uses the implemented scheduling policy to decide where the request is routed. For finding which microservices are placed on which nodes, iFogSim2 offers a service discovery functionality, which stores the dynamic microservice to node mapping. However, iFogSim2 does not provide a failure model nor a more realistic network model.

YAFS [51] simulates IoT applications in a fog environment. It adopts the application model from iFogSim, describing the dependencies between application modules as messages. The network topology is graph based, and YAFS has support for multiple graph formats and can import topologies from CAIDA [52] and BRITE [53] tools. YAFS offers three types of customizable policies: selection, placement and population. Whenever a node generates a message, the selection policy decides the node to which the message is sent for processing, as well as the route of the message through the network. The placement policy decides the placement of the application modules on the nodes. The population policy decides the placement of workload generators on the nodes, the type of messages they generate, and the temporal distribution based on which the messages are generated. All these policies are dynamic and application-specific, that is, they can be invoked any time during the simulation and each application has its own selection, placement and population policy. YAFS supports modelling the failures of links and nodes, even though the possible recovery of a failed node is not considered. However, YAFS does not offer any default implementations for mobility models.

EdgeCloudSim [54] is another well-known simulator built on top of CloudSim. It allows implementing a centralized edge orchestrator module that decides about the scheduling of user tasks on edge servers or cloud. The user tasks are generated by end devices according to a Poisson process. EdgeCloudSim offers some benefits over iFogSim. It offers a mobility model for the end devices, and its network model is more realistic as it takes into account the effect of network load on transmission delays. However, there are many deficiencies. EdgeCloudSim offers only one simple default mobility model. Further, EdgeCloudSim does not model the energy consumption of the end devices and cloud datacenters, nor offers a failure model. EdgeCloudSim also supports only a three-level edge computing environment, where all edge servers are a single hop away from the end devices in a wireless environment, and a global cloud is directly above the edge server tier.

PFogSim [55] has been built on top of EdgeCloudSim, extending its features as follows: support for simulating multi-layered fog environments; mobility of fog nodes can be simulated in addition to the end devices; dynamic networking support that allows network routes to be updated in response to changes in the availability and location of fog nodes; multi-hop network interconnection that allows nodes to connect to each other over one or more network hops; and the separation of send and receive paths, which allows separating input data generators and output data consumers. PFogSim has been designed for evaluating service placement methods, and it offers six predefined service placement methods. It assumes that each end device has a corresponding application service instance, and during the simulation initialization, the implemented service placement method assigns for each device a fog node that is responsible for executing its service requests (i.e., hosts the service instance). During simulation, whenever a device generates a task, it is routed through the network to the assigned fog node.

PFogSim does not model the energy consumption of the system entities nor the failure of links and nodes. The mobility model is very simplistic, as each mobile device gets a randomly assigned direction and speed at the start of the simulation. Further, PFogSim does not support implementing customized methods for runtime load-balancing.

Sphere [56] simulates cloudlet computing environments where small clusters of computing nodes are placed near data sources. The environment is modelled as a directed weighted graph, where the nodes represent the clusters and workload generators in the network, and the weight on an edge between two nodes represents the availability of the network link. Workload is modelled as independent jobs, and one job consists of multiple independent tasks. Sphere allows implementing and evaluating scheduling strategies on two levels: edge infrastructure level and cluster level. Edge infrastructure level orchestration is responsible for shutting down or powering on clusters, and determining the best cluster for a job to be scheduled. Cluster level orchestration, in turn, is responsible for shutting down or powering on machines inside a cluster, as well as scheduling the incoming tasks on the machines.

As Sphere is directed towards evaluating infrastructures and scheduling strategies, the possible control or data dependencies between the jobs are not modelled, nor the details of the data sources (sensors and devices). In fact, one workload generator in Sphere represents a geographic area where the workload patterns and users are assumed to be similar. Consequently, there is no mobility model. Further, the failure of a link or a machine is not modelled, nor is the possible runtime load-balancing by migrating the tasks or jobs between machines or clusters.

FogNetSim++ [57] is a fog simulator built on top of OMNeT++ [58]. OMNeT++ is a network simulator that can simulate low-level networking details. Hence, out of the simulators considered in this thesis, FogNetSim++ is the only one that simulates packet- and protocol-level networking details. It can model network link properties such as packet drop, bit error rate and retransmission, as well as simulate various protocols, such as TCP, UDP, FTP, HTTP, MQTT, CoAP, and AMPQ. FogNetSim++ has been developed for simulating scenarios related to task execution on fog nodes and publish / subscribe communication model. The environment in FogNetSim++ includes sensors, end devices, APs, routers, fog servers, a fog broker, and cloud. Fog broker is a central orchestrator that manages handoffs due to mobility, registers and manages publishers and subscribers, provides a communication link with the cloud datacenter, manages the pricing of the resources, and schedules tasks on fog servers. The scheduling policy of the broker can be customized. FogNetSim++ also offers multiple default mobility models, but does not model the failure of the links or nodes, nor any runtime load-balancing between the fog servers, such as migrating VMs.

PureEdgeSim [18] has been developed for simulating IoT applications in fog and mist computing environments. It is the only fog simulator that also considers the mist computing aspect, that is, it simulates task execution also on end devices and allows an end device to offload its task to another end device for processing. Applications are modelled as independent tasks generated by the end devices, and a customized offloading policy that decides when, where and how the tasks are offloaded can be implemented. The orchestration policy can be run by the cloud, fog nodes or end devices, allowing varying levels of control for the offloading policy. PureEdgeSim offers a more realistic network model for MAN and Wide Area Network (WAN) transmissions, implements one default mobility model, and models node failures due to running out of battery. PureEdgeSim also offers an extensive energy model, because in addition to the energy consumption of computation, the energy consumption in Local Area Network (LAN), MAN, and WAN is measured.

Table 1 gives an overview of the simulators that model applications as graphs where the edges correspond to data or control dependencies. Table 2, in turn, summarizes the simulators that model applications as independent tasks or jobs. The tables state the *programming language* of the simulator, the main *purpose* of the simulator, what *orchestration policies* can be customized and what are the main *infrastructure components* in the simulator. *Control* lists what type of control topologies are included in the simulator for the orchestration policies. *Energy model* indicates whether the simulator measures the energy consumption of the edge / fog nodes or end devices. *Mobility model* indicates whether the simulator offers a mobility model for the end devices. It is further indicated whether the mobility model is simple or realistic. The model is realistic if the simulator allows extracting mobility patterns from real data sets, otherwise the model is simple. *Network model* indicates whether the network links and transmissions on them are simulated. It is further indicated whether the model is high-level or low-level, and whether the model is simplistic or not. The model is low-level only if packet- and protocol-level details are simulated, otherwise the model is high-level. The model is simplistic if the simulator ignores the effect of network congestion on transmission delays. *Failure model* indicates whether the failure of nodes or links is simulated. Finally, *logs* indicates whether the simulator outputs a detailed log of the simulation events.

PureEdgeSim is chosen as the basis for EISim because it is the only simulator that also simulates processing on end devices in addition to processing on fog nodes. Hence, it is the only simulator that allows proper simulation of task offloading from the end devices to the edge, in a sense that it allows to model a device as a rational agent that can decide whether it executes a task locally or offloads it to the edge. Further, PureEdgeSim has been the most active simulator in terms of development, it supports extensibility well and facilitates the implementation of different control topologies for the offloading function.

2.3. Orchestration

The research on edge orchestration is characterized by dispersion. Even though the typical definition of orchestration includes the idea of *automated management* (see e.g. [59, 13, 32]), the notions of what elements are included in the scope of the management, as well as what functions belong to the management are wide and varied. For example, Hong and Varghese [14] focus on the management of fundamental, physical and virtual resources in edge and fog computing, dividing different management algorithms into discovery, benchmarking, load-balancing, and placement. On the other hand, Taleb et al. [12] focus on the management of MEC services and VNFs, seeing that the management functionality includes resource allocation, service placement, platform selection for a service request, and monitoring MEC service deployments.

The research on edge orchestration has also suffered from sticking to the traditional approach of the centralized cloud orchestration, which decouples the fundamental, physical and virtual resources from the application and workflow level resources. As the computing and storage resources in cloud are basically unlimited, it has been possible to approach orchestration from the scope of an application. The Service Level Objectives (SLOs) have characterized the needs of an application in terms of resource usage, cost and quality [28]. Whenever a monitoring function recognizes that the performance of an application has fallen below some threshold on some of the three objective axes (resource usage, cost and quality), resource management functionality is activated to elastically harness the cloud resources in a way that the SLOs are again satisfied [28].

When coming down from the cloud towards the device level, the available resources become more and more limited. The reactive and thresholded management methods from cloud orchestration will not suffice in edge environment, not to mention the whole computing continuum environment. The orchestration of edge and eventually the whole computing continuum must be a lot more proactive and context-aware, which will require distributed and intelligent orchestration methods.

Table 1. Summary of edge and fog simulators that model applications as graphs

Simulator	Prog. language	Purpose	Customizable policies	Infrastructure components	Control	Energy model	Mobility model	Network model	Failure model	Logs
iFogSim [46]	Java	Simulating IoT applications in a fog environment	Placement	Sensors and actuators, fog devices, fog broker, cloud	C	Yes	No	Yes^{HS}	No	No
MyiFogSim [48]	Java	Simulating IoT applications in a fog environment	Placement, migration	(Mobile) sensors, (mobile) actuators, fog and mobile devices, fog broker, cloud	C	Yes	Yes^{S}	Yes^{HS}	No	No
MobFogSim [49]	Java	Simulating IoT applications in a fog environment	Placement, migration	(Mobile) sensors, (mobile) actuators, fog and mobile devices, fog broker, cloud	C	Yes	Yes^{R}	Yes^{HS}	No	No
iFogSim2 [50]	Java	Simulating IoT applications in a fog environment	Placement, scheduling, migration, clustering	(Mobile) sensors, (mobile) actuators, fog and mobile devices, fog broker, cloud	C, D	Yes	Yes^{R}	Yes^{HS}	No	No
YAFS [51]	Python	Simulating IoT applications in a fog computing environment	Placement, scheduling	Sensors and actuators, fog nodes, cloud	C, D	Yes	No	Yes^{H}	Yes	Yes

^{S} Simple mobility model
^{R} Realistic mobility model
^{H} High-level network model
^{HS} High-level, simplistic network model
^{L} Low-level network model

Table 2. Summary of edge and fog simulators that model applications as independent tasks or jobs

Simulator	**Prog. language**	**Purpose**	**Customizable policies**	**Infrastructure components**	**Control**	**Energy model**	**Mobility model**	**Network model**	**Failure model**	**Logs**
EdgeCloudSim [54]	Java	Simulating small-scale edge computing scenarios	Scheduling	Mobile devices, edge servers, cloud	C	No	Yes^{S}	Yes^{H}	No	Yes
PFogSim [55]	Java	Simulating large-scale fog computing scenarios	Placement	(Mobile) end devices, (mobile) fog nodes, cloud	C	No	Yes^{S}	Yes^{H}	No	Yes
Sphere [56]	Scala	Simulating workflow execution in a cloudlet computing environment	Scheduling	Workload generators, cloudlets, cloud	C, H	Yes	No	Yes^{H}	No	No
FogNetSim++ [57]	C++	Simulating task execution in a fog computing environment	Scheduling	(Mobile) end devices, fog nodes and a broker, cloud	C	Yes	Yes^{S}	Yes^{L}	No	No
PureEdgeSim [18]	Java	Simulating IoT applications in fog and mist computing environments	Offloading	(Mobile) end devices, fog nodes, cloud	C, H, D	Yes	Yes^{S}	Yes^{H}	Yes	Yes

^{S} Simple mobility model
^{R} Realistic mobility model
^{H} High-level network model
^{HS} High-level, simplistic network model
^{L} Low-level network model

Holistic visions for the orchestration of the whole computing continuum environment have only recently emerged [5, 28]. Kokkonen et al. [5] advocate for a paradigm shift in edge orchestration, namely moving from the centralized paradigms focused on specific domains towards an intelligent and distributed paradigm that takes a more encompassing view on resources in the computing continuum. The more encompassing view on resources is based on a principle that drops the ambiguous convention of referring to the entities on hardware and OS levels as resources, and entities on middleware and application levels as services. Rather, every entity in the whole continuum is considered to be a resource, as per the definition in Section 2.1, which allows to regard the orchestration simply as the management of resources in the computing continuum. According to the definition in [5], the orchestration is divided into a number of distinct functions, which are as follows:

- Monitoring, which tracks the state and capabilities of each resource, and estimates their performance.
- Workflow management, which includes the aggregation, sharing, offloading and caching of workflow resources (data and tasks).
- Discovery, which includes the registration and subsequent look-up of available resources.
- Allocation, which includes the placement of resources, scheduling access to them, as well as their migration, scaling and replication.
- Life-cycle management, which includes creating, deleting, starting, stopping, and updating resources.

Further, these functions can be implemented in different ways, and the implementation differences can be described by attributes such as security, fault tolerance and control topology (decentralized, hybrid or centralized). The solutions also aim to fulfill different objectives, which on a high-level can be categorized into resources, cost and quality [28, 5]. The definition in [5] also accounts for the multi-domain nature of the orchestration, as there may be many possible stakeholders (e.g., end users, device owners, application providers, infrastructure providers or edge operators) that set the objectives for the orchestration. The presence of many stakeholders reveals the financial aspect of orchestration, which means that resource pricing needs to be considered alongside resource management.

The vision in [5] emphasizes that the orchestration implementations should embrace distributed intelligence, local autonomy and loose coupling. EISim aims to facilitate research on those facets. In its current form, EISim focuses on one slice of the orchestration, namely computational offloading with resource pricing. Hence, the following section presents the current state of the art in offloading and resource pricing. Regarding offloading, the focus in on vertical offloading from end devices to the edge, because that use case is the starting point in EISim.

2.4. Offloading and Pricing

2.4.1. Offloading to the Edge

The core purpose behind any computational offloading approach that focuses on offloading tasks from end devices to the edge is to reduce the task execution latency and preserve the energy of the devices, which leads to increased Quality of Service (QoS) and Quality of Experience (QoE) [60, 61, 62]. However, offloading approaches differ widely based on the system model, optimization objectives, problem formulation and solution methods.

Generally, offloading approaches consider the questions of *what*, *when*, *where* and *how* to offload [61]. What to offload concerns the partitioning of an application into tasks and determining what tasks can be offloaded. Usually studies on offloading omit the details of application partitioning, and what to offload focuses on whether to offload all application tasks (full offloading) or offload some of the tasks while executing the rest locally (partial offloading). Further, the studies differ on whether they regard one task as *atomic* or not, that is, whether one task can be further divided into offloadable and non-offloadable parts. There are also differences regarding whether possible control or data dependencies between application tasks are accounted for.

When to offload concerns the optimal timing of task offloading, which in time-slotted system corresponds to selecting an optimal time slot for the offloading [61]. Studies often propose methods that decide about the offloading and resource allocation for all tasks that have accumulated by the beginning of a new slot. Studies may also assume that all the tasks in the system can be executed inside the slot length (see e.g. [41]), so that the decision-making algorithm does not have to account for the resource use of the tasks allocated in the previous slots.

Where to offload concerns the best location for task execution, which must be decided based on the distribution of the available resources [61]. Finally, how to offload concerns the optimization objectives of the task offloading, as well as the solution methods deployed for finding the optimal partitioning, timing and placement of tasks.

Offloading approaches typically aim to optimize some combination of execution delay, energy consumption (at the end devices) and monetary cost of the execution. These objectives are often conflicting. For example, minimizing energy consumption of the end devices may be achieved by offloading tasks to the edge, which in turn incurs higher delay and monetary cost. Hence, approaches usually aim to find appropriate trade-offs between different objectives, the joint optimization of delay and energy consumption being the most common multi-objective approach [60, 62].

Solution methods to offloading problems can be categorized on a high-level into mathematical optimization algorithms, AI-based approaches, and control theory based approaches [62]. Mathematical optimization algorithms, including strategies from fields such as Mixed Integer Programming (MIP), heuristics, game theory and contract theory, are the most commonly used solution methods, particularly MIP and heuristics based methods [62]. However, mathematical optimization approaches are one-shot and static by nature, meaning that they must be re-executed every time the environment changes, and they do not learn anything about the dynamics of the environment. Further, these approaches generally deploy iterative algorithms for finding optimal or near-optimal solutions. Iterative algorithms are often executed in a centralized manner, and they usually require information exchange between the entities in the system at every iteration, which may lead to significant communication overhead in the system. Iterative algorithms also require time to converge, which means that while the algorithm searches for an optimal solution, the dynamic environment can change, rendering the resulting offloading decision unoptimal.

AI-based solution methods aim to provide the intelligence and learning needed for long-term optimization in highly dynamic systems. For example, supervised Machine Learning (ML) methods could be used to support offloading decisions through predicting dynamic network conditions, workload on the nodes, or users' mobility patterns. Supervised ML has not yet been considerably used in the context of task offloading to edge, but it has been used in, for example, resource allocation in cloud computing and service placement in fog computing (see [62] and the references therein). On the other hand, DRL has been used in some offloading studies to directly learn offloading decisions through experience in the dynamic environment (see e.g. [63, 64]). Generally, DRL can help in tackling the unsuitability of the traditional optimization algorithms for real-time, long-term optimization, which makes it a promising approach for decision making on edge.

Even though AI is a significant key factor in enabling context-aware, proactive and real-time decision making on edge, AI-based offloading methods suffer from the same issues as ML in general. The training of models is resource- and time-consuming, particularly in the case of Deep Neural Networks (DNNs), which is problematic in the resource-limited edge environment. Further, ensuring the accuracy of models in the dynamic environment and adapting the models when data distributions shift are both challenging issues.

Finally, control theory based offloading solutions are another approach to enabling real-time decision making and long-term optimization in the dynamic edge environment, but these methods also aim to guarantee properties such as system stability and reachability. In offloading literature, this approach is the least used one compared to mathematical optimization and AI-based solutions [62]. Most common control theory approach in the context of task offloading is Lyapunov optimization, which is used especially in energy optimization [62, 65, 66]. Lyapunov optimization framework allows to minimize a time average while there are constraints on the time averages of other system variables, such as task queue lengths. Typically, the time average to be minimized is the average of the expected energy consumption of the mobile users over time.

In Lyapunov optimization, the time-averaged constraints are formulated as *virtual queues*. Taking a control action that minimizes the upper bound of a so-called *drift-plus-penalty* expression in every time slot allows to minimize the objective while satisfying the constraints [67]. Further, the control action can be taken solely based on the information about the current system state, without having to know anything about the future states. The main drawback in control theory based offloading is the complexity of designing a control solution, particularly when the system model itself is complex and when there are many system properties to be guaranteed [62].

Overall, the system models in computation offloading studies are wide and varied, as the system details and main assumptions depend on the offloading use case (e.g., internet of vehicles, MEC, smart home, ultra-dense networks, green edge computing).

Each use case imposes its own assumptions and limitations, but some general limitations in the current offloading related literature can be recognized. Mobility is an essential part of the edge environment, but the studies usually either assume that the devices are static or that the tasks can be executed in such a short time that the mobility can be ignored [62, 60]. Another notable limitation concerns the scale of the system. Many works either formulate the problem for one user and one server (e.g., [68, 65, 69]), a group of users and one server (e.g., [63, 70, 71]), or formulate the problem for a group of users and a group of servers, but conduct the experiments with only a few users and servers (e.g., [64]). Further, several offloading studies rely on full centralization, meaning that there is a central controller (e.g., an AP, a Base Station (BS), or a fog broker) that makes the offloading and resource allocation decisions (e.g., [70, 72, 73]).

2.4.2. Resource Pricing

Resource pricing models especially for edge environment have not been comprehensively studied or compared yet in the literature, but the importance of developing pricing models for edge has been gaining more attention recently [74, 75, 76, 77, 78]. In cloud computing, the importance of resource pricing as a part of resource management has been better recognized and researched [79]. Generally, pricing mechanisms should ensure profits for different resource providers (e.g., service, infrastructure and network providers), while also guaranteeing the QoS and QoE of the users. Hence, effective pricing mechanisms optimize the resource utilization for the providers in a profit maximizing way while also satisfying user demands as specified in Service Level Agreements (SLAs).

Developing effective pricing mechanisms is naturally a challenging task in multi-domain, dynamic and complex environments. Consequently, many different approaches to resource pricing have been proposed, both in cloud and edge environments. These approaches can be categorized into market-based pricing, game theoretic and auction based pricing, and Network Utility Maximization (NUM) based pricing [79]. In addition, pricing approaches can be differentiated based on whether the pricing concerns only a single resource (e.g., CPU, memory, bandwidth), or a bundle of resources, which is usually provisioned as a VM [74].

Market-based pricing techniques use economic and financial concepts to formalize the resource pricing problem. A common approach is to model the resource demand as a function of price, as well as model the varying and fixed costs of the provider. Then, a price that maximizes the revenue is searched based on the model. However, the deployed demand models are often very simplistic compared to real markets, and these models often fail to account for market competition [79].

Game theoretic and auction based approaches model the participants in the resource pricing as self-interested agents that aim to maximize their own utility. These approaches are the most prevalent ones in edge environment [75]. Game theoretic approaches can use either non-cooperative or cooperative games to model the resource pricing and resource allocation problems. Currently the most common game model deployed on edge is one-leader multi-follower Stackelberg game, where typically a BS with an integrated edge server acts as a leader that decides on the resource price, whereas mobile users act as followers that decide on the amount of resources they buy based on the price given by the BS [75]. Game theoretic approaches usually consider the pricing of one resource (most often CPU) [76, 80], but there are some approaches the consider the pricing of multiple resources [81]. Further, game theoretic approaches can consider uniform pricing, where every user pays the same unit price (e.g. [76]), or differentiated pricing, where the unit price can be different for different users (e.g. [80]).

On the other hand, auction based approaches propose bidding protocols, where resource provider sells resources to buyers (users) through an auctioneer that often is the seller itself. The resources that the users bid for are invariably different types of VMs [77]. There exists a plethora of different types of auction models that can be considered [77, 82]. Designing auction mechanisms is a complex task, because they should fulfill many important properties, such as fairness, truthfulness and individual rationality [77, 82]. Security is also a very important property, but many proposed mechanisms do not address security problems, such as possible collusion between the bidders and false-name bidding [79]. Further, auctions always rely on a centralized entity (trusted auctioneer) to conduct the auction. For example, in [74], a central auctioneer collects all the bids and resource requirements from the users, after which it allocates the resources both on edge and cloud level and determines the user payments based on the bids.

Game theoretic and auction based pricing approaches suffer from the same issues as the mathematical optimization based offloading approaches (see Section 2.4.1). They are one-shot and static by nature, without the ability to learn anything about the dynamics of the environment. Iterative algorithms are also often deployed for finding the equilibrium solutions. Hence, AI-based methods, such as DRL, could provide the adaptivity, real-time decision making and long-term optimization also needed in resource pricing. So far this aspect has not been widely studied in the context of resource pricing on edge. Some studies do exist, but they mostly consider single agent settings and algorithms, that is, there is only one centralized agent learning and making pricing decisions in the environment. For example, Zhan et al. [83] propose DRL-based pricing for federated learning settings, where a central aggregator located in the cloud uses DRL to learn the price it offers to each mobile user in order to incentivize them to participate in the model training. Chen et al. [84], in turn, consider an offloading scenario where a single AP with an integrated edge server uses DRL to learn and periodically update a unit price for computation.

Finally, NUM based pricing approach is used when the goal is to maximize the social welfare (total utility) of the users in a network under capacity constraints. In other words, each user wants to get a portion of some constrained resource (usually network bandwidth), and the resource provider wants to allocate the resource in a way that the total utility of the users and the profit of the provider is maximized. These techniques often use differentiated pricing [79]. When the formulated NUM problem is convex, it can be solved optimally in a distributed manner through dual decomposition [85], meaning that the provider does not need to know the individual utility functions of the users.

Overall, what resource pricing models and methods would be suitable for edge is largely an open research question. Regardless, it is quite unanimous that dynamic pricing is a key factor in maximizing the profits for resource providers, optimizing the resource utilization, and creating competition between different providers [75, 77, 82]. However, it is good to note that even though a multitude of dynamic pricing schemes has been proposed for cloud and edge, they have largely been only evaluated in simulated settings. In practice, the predominant pricing model in cloud still is a usage-based, static model [86].

2.4.3. Offloading with Pricing

The previous sections offered a general overview of the state of the art in offloading and resource pricing literature. EISim focuses on the intersection of these fields. Hence, this section gives an overview of the studies that are also in the intersection of these fields. These studies propose task offloading solutions while also considering the pricing of the resources. The focus is on studies that consider vertical offloading schemes from end devices towards edge.

A summary of task offloading studies with resource pricing is given in Table 3. In the table, *system model* refers to the system entities considered in the study. *Task model* indicates how the tasks generated by the end devices are characterized in the study. *Optimization objectives* state the optimization targets considered for the different system entities in the study. *Decision variables* state what type of decisions are considered in the system, and the system entity mentioned in the parenthesis is the one to which the decision belongs. Note that this does not necessarily mean that the system entity makes the corresponding decision completely autonomously in the proposed solution, only that in the problem formulation of the study this decision belongs to the system entity. *Problem formulation* refers to the general framework in which the offloading problem is formulated, and *solution method* states the deployed method for finding the optimal or near-optimal solution. *Control* refers to the level of autonomous decision making in the proposed offloading solution, which can be either centralized (C), hybrid (H) or decentralized (D). *Pricing method* states the resource pricing approach used in the study. *Result return ignored* indicates whether the study ignores the overhead (delay and energy consumption) associated with returning the result of the task execution back to the user.

Table 3. Task offloading studies with resource pricing

Study	**Use case**	**System model**	**Task model**	**Optimization objectives**	**Decision variables**	**Problem formulation**	**Solution method**	**Control**	**Pricing method**	**Result return ignored**
Kim et al. 2018 [87]	MEC	Mobile users, one BS connected to edge cloud	Data size, CPU cycles per 1 bit of data	Energy, payment (users); revenue (edge cloud)	Purchased amount of CPU cycles (users); price (edge cloud)	Stackelberg game	Iterative algorithm	C	Price per CPU cycle, uniform	Yes
Liu et al. 2018 [88]	Fog computing	Mobile users, one fog node, cloud	Poisson arrival rate, data size	Energy, delay, payment (users)	Offloading probability, transmit power (users)	Multi-objective optimization	Iterative algorithm	C	Price per task, uniform (static)	Partially
Liu and Liu 2018 [89]	MEC	Mobile users, one BS with edge server	Data size, CPU cycles per 1 bit of data, ratio for return data size	Delay, payment (users); revenue (server)	Offloaded data amount (users); price (server)	Stackelberg game	Iterative algorithm	C	Price per CPU cycle, both uniform and differentiated considered	No
Liu et al. 2019 [90]	VEC	Mobile users, vehicular servers, edge servers, operator	Data size, CPU cycles, delay constraint	Profit of the operator	Offloading destination for tasks, allocated percentages of communication and computation resources (operator)	Deep Reinforcement Learning	DQN-based algorithm	C	Price per bps and price per CPU cycle, uniform (static)	Yes
Wang et al. 2019 [91]	MEC	Mobile users, edge clouds, auctioneer	Data size, CPU cycles	Revenue, processing cost (edge clouds)	Task allocation, payments (auctioneer)	Provider profit maximization	Auction mechanism	C	Price per task, determined by bidding	Yes
Sufyan and Banerjee 2020 [92]	MEC	Mobile users, edge servers, cloud	Poisson arrival rate, data size	Energy, delay, payment (users)	Offloading probability, transmission power (users)	Multi-objective optimization	Iterative algorithm	C	Price per task, uniform (static)	Yes
Yi et al. 2020 [93]	MEC	Mobile users, one AP connected to edge cloud, regulator	Arrival rate, data size, CPU cycles	Local processing cost, transmission energy and delay costs, payment (users); revenue, processing cost (regulator)	Offloading probabilities for users, transmission scheduling order, prices (regulator)	Network Utility Maximization	Optimal solution through mathematical analysis	C	Price per task, differentiated	Yes

Continued on the next page

Table 3. Task offloading studies with resource pricing (continued)

Study	**Use case**	**System model**	**Task model**	**Optimization objectives**	**Decision variables**	**Problem formulation**	**Solution method**	**Control**	**Pricing method**	**Result return ignored**
Liang et al. 2021 [94]	MEC	Mobile users, one edge server	Data size, CPU cycles per 1 bit of data, ratio for return data size	Delay, payment (users); revenue (server)	Offloaded data amount, CPU usage on server (users); price function parameters (server)	Stackelberg game	Iterative algorithm	C	Price per time unit, differentiated	No
Yuan et al. 2022 [41]	MEC with priority classes	Mobile users, one AP with edge server	CPU cycles	energy, delay, payment (users); revenue (server)	Local pre-processing time length, priority class selection (users); priority class prices (server)	Non-cooperative game (users); revenue maximization (server)	Equilibrium solution through analytical method (users); dynamic programming (server)	C	Price per priority class, uniform inside class	Yes
Li et al. 2022 [95]	VEC	Mobile users, vehicles, one RSU, one edge server, operator	Data size, CPU cycles, delay constraint	Delay, payment to server (users); Revenue, energy, payment to RSU (server); Revenue, payment to vehicles (RSU); Revenue, energy (vehicles)	CPU usage on server (users); price for users, CPU usage for task execution, purchased CPU resources from RSU (server); price for server, contracts for vehicle types (RSU)	Three-stage Stackelberg game	Algorithms based on backward induction	C	Price for CPU usage, uniform	Yes
Liu and Fu 2022 [96]	MEC	Mobile users, one edge server	Data size, CPU cycles per 1 bit of data	Delay, payment (users); revenue (server)	Offloaded data amount, number of CRBs (users); price (server)	Stackelberg game	Dynamic programming	C	Price per CRB, differentiated	Yes
Liu et al. 2022 [97]	MEC	Mobile users, APs, edge clouds	Data size, computational demand, return data size, delay constraint	Delay, energy, budget surplus (users); system utility (APs, edge clouds)	Offloading decision, budget allocation for CRBs and RRBs (users); RRB price (APs); CRB price (edge clouds)	Mixed Integer Programming	Microeconomic theory based algorithm for budget allocation; max-flow-min-cut based algorithm for pricing	C	Price per CRB and RRB, uniform	No

Continued on the next page

Table 3. Task offloading studies with resource pricing (continued)

Study	**Use case**	**System model**	**Task model**	**Optimization objectives**	**Decision variables**	**Problem formulation**	**Solution method**	**Control**	**Pricing method**	**Result return ignored**
Zhang et al. 2017 [98]	VEC	Vehicles, edge servers, one backup server	Data size, computational demand, delay constraint	Delay, payment to edge servers (vehicles); revenue, processing cost, payment to backup server (edge servers)	Offloading destination, number of CRBs purchased from edge servers (vehicles); Price, number of CRBs purchased from backup server (edge servers)	Stackelberg game	Iterative algorithm	H	Price per CRB, uniform	Yes
Kim et al. 2018 [99]	MEC	Mobile users, edge servers from one provider	Data size, CPU cycles per 1 bit of data	Energy, payment (users); revenue, energy (provider)	Offloading decision, CPU speed and network interface selections (users); price, server provisioning (provider)	Lyapunov optimization	Minimizing the upper bound of drift-plus-penalty expression	H	Price per CPU cycle, uniform	Yes
Guo et al. 2019 [100]	Smart Home	End users, APs, one edge server	Poisson arrival rate, delay constraint	Delay, payment (users); revenue, payment to server (APs); revenue (server)	Amount of CRBs purchased (users); Prices (APs); Amount of CRBs rented to each AP (server)	Stackelberg game for users and AP; one-to-many matching for APs and server	Equilibrium solution through analytical method; matching algorithm based on preference list	H	Price per CRB, differentiated	Yes
Chen et al. 2021 [84]	MEC	Mobile users, one AP with edge server	Poisson arrival rate, data size, CPU cycles	Delay, energy, payment (users); revenue (server)	Offloading probability (users); price (server)	Non-cooperative game (users); Deep Reinforcement Learning (server)	Equilibrium solution through analytical method (users); Policy gradient based algorithm (server)	H	Service fee for offloading probability, uniform	Yes
Shi et al. 2021 [101]	MEC	Mobile users, edge servers	Data size, CPU cycles	Energy, payment (users); Energy, revenue (servers)	Offloading destination (users); prices (servers)	Deep Reinforcement Learning	DQN-based algorithm (users); DDPG-based algorithm (servers)	H	Price per time unit, uniform	Yes
Su et al. 2021 [76]	MEC	Mobile users, one BS, ESPs, CSPs	Poisson arrival rate, data size, CPU cycles	Delay, energy, payment (users); revenue (providers)	Offloading probabilities (users); prices (providers)	Two-tier Stackelberg game	Iterative algorithms	H	Price per CPU cycle, uniform	Yes

Continued on the next page

Table 3. Task offloading studies with resource pricing (continued)

Study	**Use case**	**System model**	**Task model**	**Optimization objectives**	**Decision variables**	**Problem formulation**	**Solution method**	**Control**	**Pricing method**	**Result return ignored**
Lyu et al. 2022 [102]	MEC on a campus	Mobile users, APs, edge servers, global task scheduler	Task arrival probability, data size, CPU cycles, delay constraint	Delay, energy, payment (users); total revenue (servers, scheduler)	Offloading decision (users); price (servers); load-balancing decision (scheduler)	Deep Reinforcement Learning	DQN-based algorithms	H	Price per CPU cycle, uniform	Yes
Li et al. 2021 [103]	MEC with two priority classes	Mobile users, one AP with edge server	Poisson arrival rate, data size, CPU cycles per 1 bit of data	Energy, delay, payment (users); social welfare (server)	Offloading probability, priority class selection (users); priority class prices (server)	Network Utility Maximization	Iterative search algorithm	D	Service fee for congestion, uniform inside class	Yes
Seo et al. 2022 [104]	MEC	One busy edge server, a new user comes to offload	Data size, CPU cycles per 1 bit of data, delay constraint, ratio for return data size	Delay, payment (user); revenue (server)	Offloaded data amount, CPU usage on server (user); price function parameters (server)	Stackelberg game	Equilibrium solution through analytical method (user); Supervised learning (server)	D	Price per time unit, differentiated	No

Orchestration Control Topologies in the Surveyed Studies

Control in the proposed offloading and pricing solutions is of utmost interest in the context of this thesis. Hence, the main categorization axis used for the studies in Table 3 is the control topology. The studies were categorized based on the definitions for different types of control topologies in Section 2.1.2. Majority of the studies have centralized control [87, 88, 89, 90, 91, 92, 93, 94, 41, 95, 96, 97]. These studies either explicitly state that there is a central controller making all the decisions on behalf of the system entities, or the proposed solution requires strong coordination with a central controller or complete global view on the system.

The works that explicitly state the existence of a central entity that makes all the decisions are the studies of [90, 91, 93, 95]. Liu et al. [90] propose a DRL-based offloading solution for Vehicular Edge Computing (VEC), where a central operator decides upon user request where the task is executed and how much computation and communication resources are allocated to it. Li et al. [95] also consider a VEC use case, where users can offload tasks to an edge server. The edge server can supplement its computing resources by purchasing idle resources from vehicles through a Road Side Unit (RSU) that uses a contract-based incentive mechanism to recruit vehicles. They formulate the problem as a three-stage Stackelberg game, and derive algorithms for finding the optimal solution to each stage through backward induction. These algorithms are run by a central controller with global system view. Wang et al. [91], in turn, consider a MEC system with multiple users and edge clouds, where an auctioneer collects task information from users and then conducts an auction where the providers bid for the tasks. The central auctioneer allocates the tasks and determines the payments for the users. Finally, Yi et al. [93] consider a MEC system with mobile users, one AP and one edge cloud, where a central network regulator is employed to determine a long-term scheme for joint computation offloading and uplink transmission scheduling. The goal is to maximize the social welfare of the system, which refers to the joint utilities of the users and the regulator.

The works that have centralized control due to requiring strong coordination with a central entity are the studies of [87, 89, 94]. These studies all consider a MEC system with multiple users and one edge server. They formulate the offloading problem as a Stackelberg game where the server first decides on the resource price after which the users make their offloading decisions. Even though each study proposes a distributed, iterative algorithm for finding the optimal or near-optimal solution, the decision making is strongly controlled by the edge server. This is because finding the final offloading solution requires that the users report their offloading decisions to the server after every price update until the server terminates the process. Hence, the decision making of the users is tightly coupled with the edge server, as the server imposes heavy control over the final decisions of the users, limiting their autonomy.

The works that have centralized control due to requiring global system view are the studies of [88, 92, 96, 97, 41]. These studies do not explicitly state the existence of a central controller, but any practical consideration of the proposed solution requires that there is some type of centralized entity with global system view.

Liu et al. [88] consider a fog computing scenario with multiple users and one fog node. The users need to decide on their offloading probabilities and transmission powers in a way that the average energy consumption, delay and monetary cost of all users is minimized. They propose an iterative algorithm for finding the optimal offloading probability and transmission power for every user at the same time, which requires that a central entity with global information runs the proposed algorithm. Sufyan and Banerjee [92] have a similar system model, problem formulation and solution method to that in [88], but instead of offloading to one fog node they consider offloading to several edge servers.

Liu and Fu [96] consider a MEC system with multiple users and one edge server. They formulate the offloading problem as a Stackelberg game, where the server first decides on the resource price for each user, after which the users decide how much data they offload and how many Computing Resource Blocks (CRBs) they purchase from the server. They propose a dynamic programming based algorithm that outputs the optimal price, number of CRBs and offloaded data amount for each user at the same time. The algorithm must be run by a central entity that has global knowledge of the system; for example, it must know the private weight factors in the users' utility functions.

Liu et al. [97] consider a MEC system with multiple users, APs and edge clouds. The users have a finite budget for buying CRBs from edge clouds and Radio Resource Blocks (RRBs) from APs for task offloading. The users need to make their offloading and budget allocation decisions in a way that their utility is maximized. On the other hand, the APs decide on their RRB prices and the edge clouds decide on their CRB prices in a way that the system utility is maximized. System utility refers to the geometric mean of the user utilities, weighted by their budget allocation amount for offloading. They propose an algorithm based on microeconomic theory for finding the optimal budget allocation under fixed prices. Then they propose a max-flow-min-cut based algorithm for finding the resource prices that lead to a market equilibrium, where the resource supply meets the resource demand. The algorithm requires a central entity that controls the system entities and their decisions.

Finally, Yuan et al. [41] consider a MEC system with one edge server that offers priority classes for task execution. The users that want to offload tasks first decide how long they locally pre-process the tasks before offloading, as well as decide the priority class for their tasks. The server decides the priority class prices and schedules the communication and computation resources among the users. They formulate the decision making of the users as a non-cooperative game and derive the equilibrium solution through mathematical analysis. For the server, based on the equilibrium solution for the users, they formulate a revenue maximization problem that can be solved with dynamic programming. However, the decisions of both the users and the server requires global knowledge of the system, such as the total number of tasks in the system, the distribution of tasks over the users, as well as the users' private delay and pricing cost sensitivity coefficients.

The works that have hybrid control topology are the studies of [98, 99, 100, 84, 101, 76, 102]. Zhang et al. [98] consider a VEC system where vehicles can offload tasks to the edge servers on the side of a unidirecional road. There exists one backup server in the system from which the edge servers can purchase CRBs in the case that the demand from the vehicles exceeds the capacity of the server. The price of the backup server is predetermined, while the interaction between the edge servers and the vehicles is modelled as a Stackelberg game. In the game, the servers first decide on the resource price and the number of CRBs purchased from the backup server, after which the vehicles decide on the offloading destination and the number of CRBs purchased from the destination server. They propose a distributed, iterative algorithm for finding the optimal strategies for the edge servers. In the algorithm, each edge server is able to decide with local autonomy, but the iterative updating of their strategies imposes control over the decision making of the vehicles, as the vehicles cannot carry out their offloading decisions until the algorithm converges.

Kim et al. [99] consider a time-slotted MEC system where there exists one offloading service provider with several edge servers and multiple mobile users that have subscribed to the offloading service. At the beginning of a slot, provider makes decisions on pricing and server provisioning, after which each mobile user decides its offloading policy, local CPU clock speed and network interface activation (Wi-Fi or cellular). They use Lyapunov optimization to solve the problem, aiming to minimize the energy and monetary costs of each side (provider and users) while guaranteeing that all tasks are processed within a finite time. Users can do their decisions with a simple thresholded method using only local information. Provider requires user-side information for its decision making. However, it is assumed that the provider can estimate most of the needed information through the information that users are required to give when they subscribe to the service, and that some private information can be estimated through pricing and observing user behavior for a few time slots. Hence, this work has hybrid control topology, where one centralized controller (provider) makes the decisions for all the edge servers, while the provider and users are able to decide with local autonomy.

Guo et al. [100] consider a smart home scenario with end users, APs and one edge server. The users send offloading requests to the APs that purchase CRBs from the server and allocate them to the users. Each AP also charges the users for the CBRs. The decision making between an AP and its users is formulated as a Stackelberg game, the equilibrium solution of which is derived through mathematical analysis. The solution for an AP requires global knowledge as the AP must know the users' private utility function weight factors. The resource price of the edge server is predetermined, and the resource allocation between the server and APs, as well as an AP and its users is done through one-to-many matching with preference lists. Hence, the APs function as centralized controllers for the end users, while the APs and the server can do their decisions with local autonomy.

Chen et al. [84] consider a MEC system with multiple users and one AP with an integrated edge server. The server periodically decides a resource price, while each user decides its offloading probability. They propose a DRL-based dynamic pricing algorithm for the edge server. The algorithm allows the server to learn a pricing strategy without any prior knowledge about users' utility functions and task arrival patterns. Note that the study focuses on evaluating the performance of the proposed algorithm with respect to a centralized scheme with perfect system information, which means that their interest is in verifying whether the DRL algorithm can converge to an optimal solution. Hence, the user side decision making after the server's price decision is formulated as a non-cooperative game and solved optimally in a centralized manner. Due to this centralization, the study is classified as having hybrid control, but it is good to note that the proposed scheme has the potential and purpose to be decentralized in practice with further development.

Shi et al. [101] consider a MEC system with mobile users and edge servers. The servers periodically set a price for task execution, after which the users decide their offloading destination (locally or at a chosen server). They propose a Deep Q-Network (DQN) based algorithm that outputs the offloading destination for every user at the same time, and a DDPG-based algorithm that outputs the pricing decision for every server at the same time. The state space for the DQN-based algorithm requires global knowledge of every user, and the state space for the DDPG-based algorithm requires global knowledge of every server. Hence, running these algorithms requires that there is a central entity making all the decisions for the users and another central entity making all the decisions for the servers. Note that the study does not explicitly state the existence of such central controllers, but given their problem formulation and descriptions of the algorithms, the need for central decision making entities is evident.

Su et al. [76] consider a MEC scenario with multiple users, one BS and multiple ESPs and Cloud Service Providers (CSPs). The ESPs deploy their edge servers (one per ESP) at the BS, and each CSP owns a cloud datacenter connected to the BS via backbone networks. The users decide their offloading strategy, while each provider decides the price for task execution on its resources. The offloading problem is formulated as a two-tier Stackelberg game, where a non-cooperative game between the providers on resource pricing is followed by a non-cooperative game between the users on offloading strategies. The proposed algorithms for finding the equilibrium solutions can be executed in a distributed manner, but the system has elements of centralized control. This is because the algorithm for finding the Nash equilibrium between the users requires several rounds where the users broadcast their strategy profile to other users through a BS, after which they update their strategy. This continues until the strategies of all users are fixed. Also one update of provider pricing requires that each provider adjusts its price twice, gets the strategy profile from all users, and then does the final price update based on gradient ascent.

Finally, Lyu et al. [102] consider MEC on a campus, where each building has one dedicated edge server and several APs. The servers are clustered into groups based on load characteristics, and each group jointly trains an RL agent using a DQN-based algorithm to periodically decide on the resource price for each server. Users make an offloading decision based on the price set by the edge server and their own delay requirements. Further, there is a global task scheduler that balances the load between the edge server groups. Scheduler also uses a DQN-based algorithm to learn load-balancing decisions. The users and edge server groups can make their decisions with local autonomy, while the scheduler with a global system view controls the load on the servers.

Decentralized control topology is the rarest one in the studies. Only two studies have it [103, 104].

Li et al. [103] consider a MEC system with multiple users and one AP with an integrated edge server. There are two priority classes for the tasks in the system, one for ordinary tasks and the other for high priority tasks. The users need to decide a priority class for their offloaded tasks, as well as their offloading probability. The offloading problem is formulated as a NUM problem, where the server decides on the priority class prices based on the congestion the users cause to each other. The aim of the server is to incentivize the users to make decisions that lead to the maximization of the social welfare in the system. They propose an iterative search algorithm for finding the socially optimal pricing. In the algorithm, the server first observes the congestion level in the system through the average delay in each priority class, and then calculates the expected task execution delay and price for each priority class. The server broadcasts the prices and expected delays to the users, who can then change their priority class and offloading frequency decisions. Note that the key difference between this algorithm's decentralized control topology and the centralized control topology in the algorithms of [87, 89, 94] is the fact that in this algorithm, the server *observes* the offloading decisions of the users through the congestion levels and aims to *learn* the optimal pricing. On the other hand, in the distributed algorithms of [87, 89, 94], the users must *declare* their decisions to the server after each price update until the server terminates the process, and only after the termination the users can carry out the actual task offloading.

Seo et al. [104] consider a MEC scenario where one edge server is already executing tasks and a new user comes to offload. They formulate the offloading problem as a Stackelberg game, where the server first decides the parameters for a pricing function that gives the price per unit time depending on the amount of computational capacity the user wants to purchase. Given this function, the user can then decide how much computational capacity it purchases and how much task data it offloads. For the server, they propose a supervised learning based approach, where the server uses historical data to train a feedforward neural network to output optimal parameters. The input to the network consists of the status of the edge server and the information the user is required to give when it sends an initial offloading request. For the user, they derive an optimal decision under the given parameters through mathematical analysis. Hence, in their system model, each side is able to make its decision using locally available information without any centralized control.

General Remarks

In general, the most common system model in the studies is a small system with only one edge server. On the other hand, the works that do consider systems with several edge servers strongly rely on centralized entities with global system view. There is a lack of works that would consider practical offloading in large systems with multiple edge servers, APs/BSs and users. Further, the works rarely consider the movement of the users in the system and focus on solving the offloading problem in a system model where the users are assumed to stay in a fixed place during the offloading.

The task models in the studies can be divided into two classes. In the first class, one task is considered to be atomic in a way that it is either completely offloaded or completely executed locally [98, 99, 100, 88, 91, 90, 92, 93, 103, 101, 76, 84, 97, 95, 102], whereas in the second class, one task can be divided into two parts for offloading and local execution [87, 89, 94, 96, 41, 104]. The studies in the second class typically assume that if the size of the user task is H bits, the user can arbitrarily cut this task into two parts, so that I bits are offloaded and $H - I$ bits are executed locally. Further, these studies often also assume that the offloaded and local part can be executed in parallel [89, 94, 96, 104]. Such assumptions of arbitrary cutting on a bit level and independent execution are most likely unrealistic for practical implementation.

The task models can also be differentiated based on whether the study focuses on a stream of task arrivals or on a situation where at some time instant a group of users has a set of tasks they wish to offload. In the first case, the task arrival on a user device is typically modelled as a Poisson process and the interest is in finding an optimal offloading probability for each task [88, 92, 84, 76, 103]. Hence, these works focus on optimizing a long-term offloading ratio. In the second case, each user in the group typically has only one task and the interest is in finding an optimal offloading decision and resource allocation for the tasks [98, 87, 89, 90, 91, 101, 94, 96, 97, 95, 102, 104]. Hence, these works focus more on optimizing instantaneous offloading.

All the studies in Table 3 consider independent tasks where one computational task does not have any control or data dependencies on other tasks. A commonly recurring task model includes determining the task size in bits, as well as an application-specific processing density factor that defines how many CPU cycles are required to process one bit of data. It is also important to note that a great majority of the works completely ignore the delay and energy consumption in returning the task execution result back to the user. This is typically justified by stating that the result is so minuscule in size compared to the input data that it can be disregarded. However, when considering some image and video processing applications and the future use cases of augmented reality and virtual reality applications, this assumption does not always hold.

The studies deploy a variety of pricing approaches, but most commonly the pricing of a single resource is considered, this resource being CPU. A particularly recurring pricing scheme is to determine a unit price for the number of CPU cycles it takes to execute a task. Uniform pricing, where the unit price is the same for all users, is more prevalent than differentiated pricing, where the unit price can vary between users. Most of the studies use a dynamic pricing approach, where the price is one decision variable in the system. A couple of studies have a static pricing approach, where the resource price is predetermined [88, 90, 92].

Stackelberg game is the most common way to formulate the offloading and pricing problem in the studies. However, the proposed solution algorithms usually tightly couple the decision making of the system entities, requiring strong coordination in a centralized manner to reach an optimal solution before any actual offloading can be carried out. Further, majority of the proposed solution methods in the studies are one-shot in a way that they aim to optimize the system in one instant for a group of users rather than aiming for long-term optimization. These methods do not carry any knowledge of the system from a decision making period to another, and they do not consider that decisions in one period would have impact on decisions and system performance in future periods. Further, they must be re-executed every time the dynamic variables in the environment, such as channel state or the set of users, change. Some works, such as the DRL-based approaches in [90, 84, 101, 102] and the Lyapunov optimization based approach in [99], aim for adaptation and long-term optimization, but they mainly rely on centralization or consider a small-scale system.

It is also good to note that many proposed solutions for game models assume that the resource provider knows the general form of the users' utility function, even though it may not know the values of the individual coefficients in the function. Hence, the solution method may be built upon the fact that the provider knows what the users' optimal strategies are given the provider's decision. Naturally, such assumption limits the applicability of the proposed scheme in practice.

Another significant deficit when it comes to the practical applicability of the proposed schemes is related to the communication in the system. It is common for the works to assume that the communication resources of an AP or a BS can be completely divided among offloading users. This ignores the fact that in practical systems APs and BSs should still carry out their original purpose, which is to route other type of traffic in addition to the offloaded data.

Finally, all the studies evaluate their proposed method via simulation. However, because majority of the studies focus on one-shot optimization, many simulation models are static. In other words, the method is evaluated by running it under static conditions at one instant in time. Many runs are conducted by varying the initial conditions (inputs) of the model and the results from these runs are aggregated. This, however, is not able to evaluate the performance of the method in the long run. Further, the models are highly analytical, which is possible due to simplifications and the small-scale of the considered system. Consequently, the long-term performance of the proposed methods as a part of a large-scale, more realistic system remains an open question.

3. METHODS AND TOOLS

3.1. Metropolitan Area Network Creation

The creation of MAN in EISim consists of AP placement, topology creation and server placement. Further, for the hybrid control topology, servers must be clustered and cluster heads must be selected.

3.1.1. Access Point Placement, Topology Creation and Edge Server Placement

The simulation environment is a rectangular area. APs are located into the area using a hexagonal cell grid, which is a commonly used approximation of the cell shape in literature [105]. Hexagonal cell grid is chosen, because it allows to fully cover the area without any gaps between the cells while requiring fewer APs to do so compared to other cell shapes [106].

A MAN topology must be created for the APs. However, it is difficult to find public data of real-life MAN topologies. Hence, the topology creation is based on assumptions of what properties the topology has. It is reasonable to assume that the MAN topology fulfills small-world and scale-free properties, which are commonly recurring features in real world networks [107]. For example, it is known that Internet is a scale-free network [108]. A network is said to have a small-world property if the pairs of nodes in the network have a short average path length. The average path length is typically considered to be short if it increases only as the logarithm of the number of vertices in the network [107]. In turn, a network is said to have a scale-free property if the degree distribution in the network follows a power law, meaning that there are a few nodes with a high degree while majority of the nodes have a low degree [107].

The works that simulate MAN topologies often use Barabási-Albert model to create scale-free networks [109, 110, 111]. In the model, a network is grown by adding one new node at a time. The added node is then connected to a predetermined number of other nodes already in the network. The probability that the added node connects to an already existing node depends on the degree of the already existing node. However, this model has no spatial awareness, which means that it is not suitable for creating a topology between nodes that have already been placed on physical locations. A method for creating a topology for physically placed nodes should take into account the distance between the nodes when creating the connections. Hence, in EISim, the APs are connected using the *Tunable Weight Spanning Tree* (TWST) method proposed in [112]. This method was chosen, because it is simple and efficient, and because it has been developed based on actual data about power grids, which have also been observed to exhibit small-world and scale-free properties.

TWST is a low-complexity method for creating a spanning tree between physically placed nodes. The steps of TWST are shown in Algorithm 1. TWST starts by calculating the average node location $\bar{\boldsymbol{p}} \in \mathbb{R}^2$ in the network. Then a random permutation $\boldsymbol{P}$ of the nodes is created by sampling them without replacement. At each sampling round, a node is sampled from the remaining nodes with a probability proportional to the reciprocal of the Euclidean distance between the node and the average node location. The distance is weighted by a tunable weight parameter κ. After obtaining the permutation, the method connects each node to its nearest neighbour among the nodes that appear before it in the permutation. The value of κ affects the small-world property of the created spanning tree: smaller κ leads to shorter average path length.

Algorithm 1. Tunable Weight Spanning Tree

```
Input  : The total number of APs $n_{AP}$, a set of AP locations (nodes)
         $\{\boldsymbol{p}_i \in \mathbb{R}^2\}_{i=1}^{n_{AP}}$, a parameter $\kappa > 0$
Output: A spanning tree with $n_{AP} - 1$ links
1  Calculate the average node location $\bar{\boldsymbol{p}} = \frac{1}{n_{AP}} \sum_i \boldsymbol{p}_i$
2  Create an index set $\mathcal{I} = \{1, \dots, n_{AP}\}$
3  Initialize an empty array $\boldsymbol{P}$ of size $n_{AP}$
4  for $i = 1, \dots, n_{AP}$ do
5      Sample a node $j$ from $\mathcal{I}$ with probability $\propto \|\boldsymbol{p}_j - \bar{\boldsymbol{p}}\|^{-\kappa}$
6      $\boldsymbol{P}(i) \leftarrow j$
7      $\mathcal{I} \leftarrow \mathcal{I} \setminus \{j\}$
8  end
9  for $i = 2, \dots, n_{AP}$ do
10     Connect node $\boldsymbol{P}(i)$ to node $\boldsymbol{P}(j^*)$ such that
11     $j^* = \arg\min_{j<i} \|\boldsymbol{p}_{\boldsymbol{P}(i)} - \boldsymbol{p}_{\boldsymbol{P}(j)}\|$
12 end
```

EISim also implements a modified version of the link adding procedure presented in the same study [112], which allows to create more complex and robust network topologies. The procedure in [112] is simplified and generalized in EISim, because the original procedure has details that are very specific to the features of the power grid networks. The steps of the link adding procedure are shown in Algorithm 2. In the algorithm, a node i is first sampled from the network with a probability proportional to $d_i^{-\alpha}$, where d_i is the degree of the node. Parameter α is a tunable weight that affects the probability of choosing a low degree node: the larger the value of α, the more likely a low degree node is chosen. Then, another node j is sampled from the remaining nodes with a probability proportional to $\|\boldsymbol{p}_i - \boldsymbol{p}_j\|^{-\beta} d_j^{\eta}$, where $\|\boldsymbol{p}_i - \boldsymbol{p}_j\|$ is the Euclidean distance between the nodes i and j, and d_j is the degree of the node j. β and η are tunable weights that affect the small-world and scale-free properties of the resulting network. If β is large compared to η, new links are more likely to connect nearby nodes, resulting in larger average path length and less nodes with very high degree. If η is large compared to β, new links are more likely to connect nodes to high degree nodes regardless of their distance, resulting in shorter average path length and very high degree nodes.

Algorithm 2. Adding links to TWST

```
  Input  : The number of links to add $n_{add}$, the total number of APs $n_{AP}$, a set
           of AP locations (nodes) $\{\boldsymbol{p}_i \in \mathbb{R}^2\}_{i=1}^{n_{AP}}$, parameters $\alpha$, $\beta$, $\eta > 0$
  Output: A network topology with $n_{add} + n_{AP} - 1$ links
1 Calculate the degree $d_i$ for each node $i$, $i = 1, \ldots, n_{AP}$
2 while total number of links < $n_{add} + n_{AP} - 1$ do
3     Sample a node $i$ with probability $\propto d_i^{-\alpha}$
4     Sample a node $j$ from all other nodes with probability $\propto \|\boldsymbol{p}_i - \boldsymbol{p}_j\|^{-\beta} d_j^{\eta}$
5     if a link between nodes $i$ and $j$ does not exist then
6         Connect node $i$ to node $j$
7     end
8 end
```

Finally, the edge servers are co-located with the APs in EISim. Such co-location is commonly considered in the edge server placement literature, as it reduces the edge server deployment and maintenance costs [113]. In EISim, the edge servers are placed on the created network topology randomly. The probability of choosing an AP node i to host an edge server is proportional to d_i^{ν}, where d_i is the degree of the AP node. ν is a tunable weight parameter that affects the probability of choosing high degree nodes as server locations. The higher the value of ν, the more likely a high degree node is chosen. It is reasonable to assume that edge servers are more likely to be placed on central locations, hence the placement probability depends on a centrality measure. Degree centrality is used due to it being a simple yet effective measure [114].

3.1.2. Edge Server Clustering and Cluster Head Selection

Different types of clustering problems arise in many application domains. Consequently, a wide variety of clustering methods have been developed for different purposes, each with their own set of advantages and disadvantages [115]. In EISim, clustering is used to group edge servers based on proximity. This process is done offline, meaning that the edge server clusters remain static during simulation.

Distance-based clustering methods are very popular due to their simplicity and ease of implementation [115]. In general, distance-based methods can be divided into flat and hierarchical methods [115]. Flat clustering methods divide the data objects into a set of clusters in one shot with the use of prototype objects. These prototype objects, such as the cluster means in K-Means, represent the clusters. The quality of the clustering is improved iteratively with regard to some objective function. At the beginning of each iteration, the data objects are assigned to their closest prototype objects according to some distance function. Then, the prototype objects are adjusted according to the data objects in the cluster. Flat methods generally require that the user gives the number of clusters as input to the algorithm [116].

On the other hand, hierarchical clustering methods create a binary tree-based data structure called the dendrogram [116], which represents the hierarchical clustering result. The leaves of the tree form the base of the hierarchy (the lowest hierarchy level) and correspond to singleton data objects. The root of the tree (the highest hierarchy level) corresponds to the maximal cluster which contains all the data objects. The child nodes at each level of the hierarchy correspond to a set of clusters. In other words, each child node corresponds to a cluster that contains some subset of the data objects, and these data objects can be determined by traversing the tree from the current node to the leaves. The dendrogram can be cut at any level to obtain a corresponding set of clusters.

There are two approaches to building the dendrogram [116]: top-down and bottom-up. Top-down methods are also known as divisive methods that start with the maximal cluster and keep dividing it recursively into two groups until each data object is a cluster of its own or until some termination criterion is met. Bottom-up methods, in turn, are also known as agglomerative methods that start with each data object as its own cluster and combine two clusters at a time until the maximal cluster is formed.

In EISim, an agglomerative hierarchical clustering method is implemented for grouping the edge servers. A hierarchical method is chosen over a flat method mainly because it does not require determining the number of clusters beforehand. A bottom-up approach is chosen over a top-down approach due to lower complexity, as top-down approaches typically require a flat clustering method for splitting a cluster in two [115].

In agglomerative clustering, merging clusters is based on minimizing a linkage criterion that determines the distance between two clusters. It is a function of the pairwise distances between the data objects in the clusters. In EISim, the distance between a pair of edge servers (data objects) is the length of the shortest path between the servers in the created topology. Note that when calculating the shortest path, the weight of an edge is the Euclidean distance between the vertices.

There are several different linkage criteria that can be used [116]. The EISim implementation readily supports three, namely *single*, *complete* and *average*. The single linkage uses the minimum of the distances between all data objects of the two clusters as the criterion that is minimized in merging a pair of clusters. The complete linkage, in turn, uses the maximum of the pairwise distances, while the average linkage uses the average of the pairwise distances.

For obtaining the final clusters, a distance threshold must be specified. This refers to a distance after which no more clusters are merged. Such threshold can be specified by inspecting the dendrogram, which shows the distance at which each pair of clusters was merged.

After the edge server clusters have been obtained, cluster heads must be selected. It is reasonable to assume that a cluster head is a server that is somehow central in its cluster. In EISim, betweenness centrality [107] is used as the measure of centrality. Betweenness centrality of a node is the number of shortest paths (between other pairs of nodes in the network) that pass through the node. The measure characterizes how central a node is in terms of the information flow in the network [107]. Thus, it is a sensible measure for finding a central server in a cluster. Consequently, the server with the highest betweenness centrality is chosen as the cluster head in each edge server cluster.

3.2. Reinforcement Learning

Reinforcement learning studies the online decision making of an interactive agent in an uncertain environment. It has wide applicability to different types of problems and the remarkable ability to develop agents that can learn through trial-and-error without any type of exemplary supervision or complete models of the environment [117]. RL agents have a particular focus on achieving long-term goals as they aim to learn a mapping from the environment states to actions in a way that their expected long-term return is maximized. Such features make RL one prominent approach for developing intelligent orchestration methods in the highly dynamic and uncertain computing continuum, where adaptivity and the consideration of the long-term effects of actions are among the key requirements for orchestration methods [5].

3.2.1. Single-Agent Reinforcement Learning

A single-agent RL problem is generally formulated as a Markov Decision Process (MDP) [117]. MDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is a stochastic transition probability function (i.e., $T(s, a, s')$ is the probability of transitioning from state s to state s' after taking action a), and $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is a real-valued, possibly stochastic reward function. $R(s, a, s')$ gives the immediate reward when taking action a in state s and transitioning into state s'.

An agent interacts with the environment in discrete time steps $t = 0, 1, 2, \ldots$. At each time step t, the agent observes the current state of the environment s_t, uses it to decide the next action a_t, after which the environment transitions to state s_{t+1} and the agent receives a reward $R(s_t, a_t, s_{t+1})$. For selecting its actions, the agent follows a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ that maps each state-action pair (s, a) to the probability of selecting the action a in the state s. This probability is typically expressed as $\pi(a|s)$, where the "|" in the middle is used to highlight the fact that the function π defines a probability distribution over the actions for each state [117]. The goal of the agent is to find an optimal policy that maximizes its expected return, which in this thesis is defined as the expected sum of discounted rewards.

A policy can be evaluated with two types of value functions. The first type of function is called a state value function $V_\pi : \mathcal{S} \to \mathbb{R}$, which is defined as the expected sum of discounted rewards that is obtainable by starting from a state s at a time step t, and then following the policy π. It measures the value of a given state in terms of the expected return, and is formally defined as

$$V_\pi(s) = \mathbb{E}_\pi\left[\sum_{i=t}^{\infty} \gamma^{i-t} R(s_i, a_i, s_{i+1}) \middle| s_t = s\right], \tag{1}$$

where $\gamma \in [0, 1)$ is a discount factor used to control how much the agent values future rewards, and $R(s_i, a_i, s_{i+1})$ is the immediate reward at a time step $i \geq t$. The expectation is taken over the distribution of state-action-reward trajectories induced by the policy π and the environment dynamics.

The second type of value function is a state-action value function $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which is defined as the expected sum of discounted rewards that is obtainable by starting from a state s at a time step t, taking some action a and then following the policy π. It measures the value of an action in a given state in terms of the expected return, and is formally defined as

$$Q_\pi(s,a) = \mathbb{E}_\pi\left[\sum_{i=t}^{\infty} \gamma^{i-t} R(s_i, a_i, s_{i+1}) \middle| s_t = s, a_t = a\right], \tag{2}$$

where the expectation is taken over the distribution of state-action-reward trajectories induced by the policy π and the environment dynamics.

The goal of the agent, which is to find an optimal policy π^* that maximizes the expected sum of the discounted rewards, can be expressed as $\pi^* = \arg\max_\pi \ V_\pi(s)$, or equivalently $\pi^* = \arg\max_\pi \ Q_\pi(s,a)$. Note that V_π and Q_π are connected as $V_\pi(s) = \sum_a \pi(a|s) \cdot Q_\pi(s,a)$. In literature, a variety of solution methods has been proposed for finding optimal policies. These methods can be classified into value-based, policy-based, and actor-critic methods [118].

Value-based methods focus on estimating the optimal state-action value function, namely the Q-function in Equation (2). These methods do not explicitly model the policy of an agent. Instead, the policy is greedily extracted from the approximated Q-function, meaning that the action that maximizes the value of the estimated optimal Q-function is chosen in each state. Consequently, these methods can handle only discrete, low-dimensional action spaces, and the extracted policy is always deterministic. Examples of value-based methods are Q-learning [119] and SARSA [120].

Policy-based methods focus on directly estimating a parameterized policy function without explicitly estimating any type of value function. A central concept in the policy-based methods is policy gradient, which is the gradient of the expected cumulative return with regard to the parameters of the policy function. This gradient is typically estimated by sampling trajectories with the current policy, as is done in REINFORCE algorithm [121]. Then, using gradient ascent, the policy parameters can be updated in the direction of the improvement to find the parameters that produce the highest return. Policy gradient methods have better convergence properties than value-based methods [117, 118]. Further, they have the great advantage of being able to generate stochastic policies and handle continuous or high-dimensional action spaces. However, policy gradient methods suffer from large variance in the gradient estimates [118].

Actor-critic methods aim to combine the strong points of both value- and policy-based methods. This means that these methods aim to learn concurrently approximators for the policy function (actor) and a value function (critic) that usually is the Q-function [117]. The actor typically outputs a probability distribution over the action space, while the critic evaluates the current policy of the actor. The output of the critic is used as a target for the actor's gradient update. The critic itself adapts the value estimates based on the feedback (rewards) from the environment. The main purpose of the actor-critic methods is to preserve the benefits of the policy-based methods while reducing the variance of the gradient estimates and improving the efficiency of the learning. This is achieved through the addition of the critic, even though the lower variance of the gradient estimates is traded for a larger bias at the start of learning when the value estimates of the critic are far from accurate [118].

Currently, popular approximators for the value and policy functions are DNNs, which leads to the field of DRL [122, 123]. The use of the DNNs as function approximators has significant benefits in terms of representation learning and better generalization across states [123], but their use also introduces a lot of problems. These problems include the difficulty of providing theoretical guarantees about convergence, the instability of learning, the lack of interpretability of the DNN's inner workings, and low sample efficiency [123, 124]. Nevertheless, the current state of the art algorithms that have shown significant success in many real-life applications, such as games and robotics, are actor-critic DRL algorithms [123].

3.2.2. Multi-Agent Reinforcement Learning

The computing continuum is inherently a multi-agent environment. The development of RL methods for multi-agent environment requires combining game theory with single-agent RL, which leads to Multi-Agent Reinforcement Learning (MARL). Developing MARL algorithms is extremely challenging, because, from the viewpoint of one agent, the presence of other learning agents with non-stationary policies makes the environment non-stationary as well. This opponent-induced non-stationarity violates the fundamental assumption of the Markov property (a stationary environment) behind single-agent RL algorithms [125]. Other significant problems include partial observability and scalability, as one agent cannot in realistic settings observe the complete state of the environment nor try to model the whole joint action space, the dimension of which grows exponentially with the number of agents [126].

A multitude of MARL algorithms has been proposed in the literature [125, 126]. However, in the current state of the art, the development of MARL algorithms has been focused on the fully cooperative and fully competitive settings. In the fully cooperative setting, all agents have the same objective, while in the fully competitive one there is typically only two agents with opposing goals. More realistic settings, where each agent has their own objective that can be arbitrarily aligned with the objectives of other agents, are generally underexplored [5, 126]. Further, the current state of the art relies heavily on centralization to mitigate the problems of partial observability and non-stationary. For example, in the proposed MARL algorithms, the current state of the art approach to learning is *centralized training with decentralized execution* [126, 127]. This means that the agents share information, such as observations and actions, during training through a centralized entity. This information is typically used by a centralized critic that guides the learning of the actors, which, in turn, use only locally available information. During execution, all this extra information is discarded, and each agent simply follows the learned policy.

In its current form, EISim implements DDPG algorithm, which is a single-agent algorithm. This algorithm was chosen because it is a well-known, seminal algorithm for continuous action spaces. Using a single-agent algorithm in a multi-agent setting corresponds to ignoring the opponent-induced non-stationarity, which is also known as *independent learners* setting [126]. This is the simplest way to use RL in multi-agent settings, but naturally, as the assumptions behind single-agent algorithms are violated, any convergence guarantees are lost [125]. This simple setting was chosen as a starting point in EISim, because empirically, independent learners may be able to achieve satisfiable performance [126]. Implementing more sophisticated methods, both single-agent and multi-agent, is left for future work.

3.2.3. Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient [19] is a model-free off-policy actor-critic algorithm that learns concurrently DNN approximators for a policy function (actor) and a Q-function (critic). Model-free means that DDPG does not need a complete model of the environment dynamics, namely of the transition and reward functions. Off-policy means that the behavioral policy of the agent, which is used to choose actions in the environment during learning, does not need to be the same as the target policy of the agent, which is the optimal policy that the agent tries to learn.

The actor in DDPG can only output deterministic actions. Consequently, policy π is replaced with a deterministic policy function $\mu : \mathcal{S} \rightarrow \mathcal{A}$ that maps each state s into an action a. The policy function and the Q-function are parameterized with $\boldsymbol{\theta}^{\mu}$ and $\boldsymbol{\theta}^{Q}$, respectively. These correspond to the parameters of the DNNs.

Like generally in RL methods, the update of the critic in DDPG is based on the fundamental Bellman equation [117], which decomposes the Q-function in Equation (2) into a recursive form

$$Q_{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim T}\Bigg[r_t + \gamma \mathbb{E}_{a_{t+1} \sim \pi}\Big[Q_{\pi}(s_{t+1}, a_{t+1})\Big]\Bigg], \tag{3}$$

where $r_t = R(s_t, a_t, s_{t+1})$. To avoid the inner expectation and enable off-policy learning, DDPG uses a deterministic policy, which means that Equation (3) can be simplified as

$$Q(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim T}\Big[r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}))\Big]. \tag{4}$$

An approximation to the optimal Q-function is learned by minimizing a loss that measures how close the critic comes to satisfying the Bellman equation in Equation (4). This loss for the parameterized Q-function $Q(s_t, a_t; \boldsymbol{\theta}^{Q})$ is defined as

$$L(\boldsymbol{\theta}^{Q}) = \mathbb{E}\Big[\big(Q(s_t, a_t; \boldsymbol{\theta}^{Q}) - y_t\big)^2\Big], \tag{5}$$

where

$$y_t = r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}; \boldsymbol{\theta}^{\mu}); \boldsymbol{\theta}^{Q}). \tag{6}$$

The loss function in Equation (5) is called *mean square Bellman error*, which is the expected value of the Temporal Difference (TD) error [117]. Equation (6) defines the TD target for the critic. TD target is an estimate of the true Q-value, and its calculation requires an estimate of the optimal Q-value of the next state. In Q-learning, this estimate for the next state is computed as the maximum over all the actions in the discrete action space. However, in DDPG, the action space is continuous, which means that finding the action that maximizes the Q-function would be an arduous task. Hence, the actor network $\mu(s_{t+1}; \boldsymbol{\theta}^{\mu})$ is used to approximate the action that maximizes the Q-function in the next state s_{t+1}.

Equation (6) shows a clear problem in the critic update: the TD target y_t depends on the same parameters $\boldsymbol{\theta}^{Q}$ that are being updated. To stabilize the training, DDPG uses target networks that are time-delayed copies of the original actor and critic networks. Consequently, in Equation (6), the parameters $\boldsymbol{\theta}^{Q'}$ of the target critic are used instead of $\boldsymbol{\theta}^{Q}$. Further, in Equation (6), the optimal action for the next state s_{t+1} is approximated with $\mu(s_{t+1}; \boldsymbol{\theta}^{\mu'})$, where $\boldsymbol{\theta}^{\mu'}$ are the parameters of the target actor. DDPG uses a target network also for the actor, because, according to the findings of the original authors [19], it further stabilizes the training and prevents divergence.

The target parameters for the actor and the critic are updated by having them slowly track the learned networks. That is, the target parameters for both networks are updated with $\boldsymbol{\theta}' \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\boldsymbol{\theta}'$, where $\tau \ll 1$.

For optimizing the actor, the objective is simply to maximize the expected return. In other words, the actor aims to find a deterministic policy $\mu(s_t; \boldsymbol{\theta}^{\mu})$ that maximizes the Q-function $Q(s_t, a_t; \boldsymbol{\theta}^{Q})$. This objective is formulated as

$$J(\boldsymbol{\theta}^{\mu}) = \mathbb{E}\left[Q(s, a; \boldsymbol{\theta}^{Q})|_{a=\mu(s;\boldsymbol{\theta}^{\mu})} \right], \tag{7}$$

where the expectation is taken over the state distribution induced by the behavioral policy of the agent. The parameters of the actor are updated with gradient ascent under the assumption that the Q-function is differentiable with regard to action. The policy gradient used in the gradient ascent is approximated with

$$\nabla_{\boldsymbol{\theta}^{\mu}} J(\boldsymbol{\theta}^{\mu}) \approx \mathbb{E}\left[\nabla_a Q(s, a; \boldsymbol{\theta}^{Q})|_{a=\mu(s)} \nabla_{\boldsymbol{\theta}^{\mu}} \mu(s; \boldsymbol{\theta}^{\mu}) \right]. \tag{8}$$

During learning, training samples, which refer to experience tuples in the form of (state, action, reward, next state), are stored into an experience replay $\mathcal{D}$ with a limited capacity. For updating the actor and critic networks, a training batch is randomly sampled from $\mathcal{D}$. The purpose of using experience replay is to break the temporal correlation between the training samples, as well as increase sample efficiency and reduce the variance in the gradients.

An agent should be able to explore different actions in the environment in order to learn efficiently. In DDPG, exploration during training is enabled by adding noise sampled from a noise process $\mathcal{N}$ into the actor policy. This gives agent an exploration policy μ', which is defined as

$$\mu'(s) = \mu(s; \boldsymbol{\theta}^{\mu}) + \mathcal{N}, \tag{9}$$

where, in the context of this thesis, $\mathcal{N}$ is a zero-mean Gaussian noise process.

Finally, the steps of the DDPG algorithm are summarized in Algorithm 3.

Algorithm 3. Deep Deterministic Policy Gradient

```
1  Initialize the actor and critic parameters $\boldsymbol{\theta}^{\mu}$ and $\boldsymbol{\theta}^{Q}$
2  Copy the initial actor and critic parameters to the target parameters $\boldsymbol{\theta}^{\mu'} \leftarrow \boldsymbol{\theta}^{\mu}$,
   $\boldsymbol{\theta}^{Q'} \leftarrow \boldsymbol{\theta}^{Q}$
3  Initialize the experience replay $\mathcal{D}$
4  for each episode do
5      Initialize the noise process $\mathcal{N}$
6      Observe the initial state $s_1$
7      for each step $t$ do
8          Take action $a_t = \mu(s_t; \boldsymbol{\theta}^{\mu}) + \mathcal{N}$ according to the current policy and
             exploration noise
9          Observe reward $r_t$ and new state $s_{t+1}$
10         Store experience $(s_t, a_t, r_t, s_{t+1})$ in $\mathcal{D}$
11         if time to update then
12             for each update do
13                 Sample a random mini-batch of $B$ experiences $(s_i, a_i, r_i, s_{i+1})$
                     from $\mathcal{D}$
14                 Set $y_i = r_i + \gamma Q(s_{i+1}, \mu(s_{i+1}; \boldsymbol{\theta}^{\mu'}); \boldsymbol{\theta}^{Q'})$
15                 Update the critic by minimizing the loss:
17
16                     $$L(\boldsymbol{\theta}^{Q}) = \frac{1}{B}\sum_i (y_i - Q(s_i, a_i; \boldsymbol{\theta}^{Q}))^2$$
18                 Update the actor policy using the sampled policy gradient:
20
19                     $$\nabla_{\boldsymbol{\theta}^{\mu}} J(\boldsymbol{\theta}^{\mu}) \approx \frac{1}{B}\sum_i \nabla_a Q(s_i, a; \boldsymbol{\theta}^{Q})|_{a=\mu(s_i)} \nabla_{\boldsymbol{\theta}^{\mu}} \mu(s_i; \boldsymbol{\theta}^{\mu})$$
21                 Update the target networks:
23
22                     $$\boldsymbol{\theta}^{Q'} \leftarrow \tau\boldsymbol{\theta}^{Q} + (1-\tau)\boldsymbol{\theta}^{Q'}$$
                       $$\boldsymbol{\theta}^{\mu'} \leftarrow \tau\boldsymbol{\theta}^{\mu} + (1-\tau)\boldsymbol{\theta}^{\mu'}$$
24             end
25         end
26     end
27 end
```

3.3. PureEdgeSim

EISim is built on PureEdgeSim [18] (version 5.1.0). The following sections elaborate on the architecture and use of PureEdgeSim, as these are also the base of EISim.

3.3.1. Inputs

For defining the infrastructure elements, application profiles and simulation settings, PureEdgeSim takes five files as input: *cloud.xml*, *edge_datacenters.xml*, *edge_devices.xml*, *applications.xml*, and *simulation_parameters.properties*.

In *cloud.xml* file, cloud datacenters are defined in terms of energy consumption, memory, storage, CPU cores, and processing capacity per core in Million Instructions Per Second (MIPS). Further, it must be specified for each cloud datacenter whether it has a task orchestrator or not. Same information is needed to define edge datacenters in *edge_datacenters.xml* file, but, in addition, the location must be defined for each edge datacenter, as well as whether a datacenter is peripheral or not. The file must also specify the MAN links between edge datacenters.

In *edge_devices.xml* file, the specifications for different edge device types are given. For each type, the percentage of all devices that are of this type must be specified. The other settings provide a way to define a wide variety of different types of edge devices. Edge devices can be mobile or static, and they can be specified to be battery-powered with a given battery capacity and initial battery level. The computing capabilities of edge devices are also specified in terms of memory, storage, CPU cores, and processing capacity per core in MIPS, but setting these to zero makes PureEdgeSim treat the device type as sensor. The energy consumption must also be defined in terms of idle consumption and maximum consumption when CPU is at 100%. Further, it must be specified whether an edge device type generates tasks and whether it can act as orchestrator for other edge devices. Finally, the LAN connectivity of an edge device type must be given. PureEdgeSim supports three LAN connectivity types, namely cellural, Wi-Fi and ethernet.

The *applications.xml* file is used to specify different application types according to which tasks are generated. For each application type, the percentage of all task-generating devices that have this type must be defined. For the task generation, a rate that determines how many tasks are generated each minute, a delay constraint for the tasks, a task length in Million Instructions (MIs), as well as the sizes of application container, task input and task output must all be specified.

Finally, the *simulation_parameters.properties* file defines the main simulation parameters for PureEdgeSim. These parameters can be roughly categorized into general settings, simulation area settings, computing node settings, network settings, and task orchestration settings.

General settings specify the length of the simulation in minutes, whether simulation scenarios are run in parallel, whether real-time charts are shown and saved, and whether logs are generated. They also determine whether registry is used for downloading application containers, and whether to stop the simulation when the time ends or wait for all the tasks to be processed.

Simulation area settings specify the length and width of the rectangular simulation area in meters. Computing node settings specify the update interval for energy consumption and mobility updates, the radius in which two devices can offload to each other, as well as the coverage of an edge datacenter. Computing node settings also determine the minimum and maximum number of edge devices in the environment, and the step size for updating the edge device count between simulation runs.

Network settings specify the update interval for transfers in the network, as well as the bandwidth, latency and energy consumption of WAN, MAN and each type of LAN (ethernet, cellular, or Wi-Fi) links. There are also settings for enabling one shared WAN link for edge devices, and for enabling a more realistic network model that models data transfers in network links more realistically in terms of link congestion.

Finally, there are four task orchestration settings, namely *enable_orchestrators*, *deploy_orchestrator*, *orchestration_architectures*, and *orchestration_algorithms*. When orchestrators are enabled, tasks are sent to another computing node that makes the offloading decision, otherwise each edge device makes its own offloading decisions. If orchestrators are enabled, *deploy_orchestrator* defines the level on which the orchestrators reside (cloud, edge or mist). Orchestration architectures define which computing nodes can be considered as offloading destinations during task orchestration. The options are *CLOUD_ONLY*, *EDGE_ONLY*, *MIST_ONLY*, *MIST_AND_CLOUD*, *EDGE_AND_CLOUD*, *MIST_AND_EDGE*, and *ALL*. Finally, orchestration algorithms define the names of the implemented task orchestration algorithms that are used to make the offloading decisions. Note that several options can be listed on both orchestration architectures and orchestration algorithms. All possible combinations of architectures, algorithms and edge device counts form the separate simulation scenarios inside PureEdgeSim.

3.3.2. Architecture

The layered, modular architecture of PureEdgeSim is shown in Figure 3. The modules of PureEdgeSim can be organized into three layers. The lowest layer is the simulation core, which consists of modules that create, manage and monitor the simulation environment. *Simulation Manager* is a central module that initializes the simulation environment, starts and ends the simulation, as well as schedules and handles the main simulation events. It also works as a link between all other modules. *Simulation Engine* module is responsible for running the simulation by managing the event queue. *Logger* records simulation events and calculates the performance metrics at the end of a simulation run. *Simulation Visualizer* module is responsible for creating real-time charts of the simulation map, task success rate, and CPU and network utilization. *Scenario Manager* parses the input files and encapsulates all simulation scenarios. Finally, *Datacenter Manager* module is responsible for creating all the computing nodes and network links.

The middle layer consists of modules that are responsible for modelling different aspects of the infrastructure. *Network Model* handles all network related events. It manages the data transfers in the network and allocates the bandwidth of each link by taking into account the current network load. *Mobility Model* handles the location and movement of edge devices. *CPU and Memory Utilization Model* handles the resource allocation of a computing node when it receives and executes a task. Finally, *Energy Model* handles the energy consumption of computing nodes and network links.

The highest layer consists of modules that handle the creation and management of workflow resources, specifically tasks. *Application Model* encapsulates the application profiles according to which the tasks are generated. *Task Orchestration* module

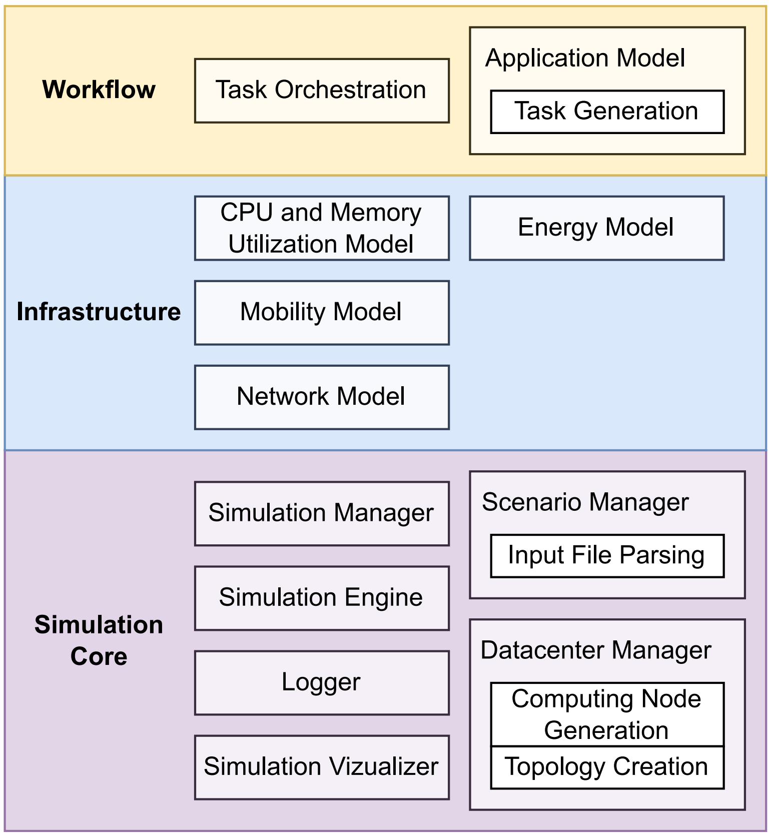


Figure 3. The architecture of PureEdgeSim.

implements the task orchestration algorithms used by orchestrators to decide on offloading.

PureEdgeSim offers a default implementation for each module. Many implementations, namely those of simulation manager, topology creator, network model, mobility model, CPU and memory utilization model, task generator, and task orchestrator, can be easily extended.

3.3.3. Simulation Workflow

The inner workflow of PureEdgeSim after a simulation has been launched is shown in Figure 4. An instance of the *Simulation* class has been created. It uses the scenario manager module to parse the input files and create a list of simulation scenarios. Then, depending on the simulation settings, it either creates one instance of the *SimulationThread* class that sequentially runs all the simulation scenarios, or creates several instances, each of which is responsible for running a given subset of the simulation scenarios. After this, the *startSimulation* method of the *SimulationThread* is called.

The *SimulationThread* object first creates instances of the *SimulationEngine* and *SimulationManager* classes. It also creates instances of the *NetworkModel* and *Orchestrator* classes, as well as uses the datacenter manager to create the computing nodes and network links. All of these inherit from the *SimEntity* class, which is the main class for any entity that can create and handle events during simulation. It is

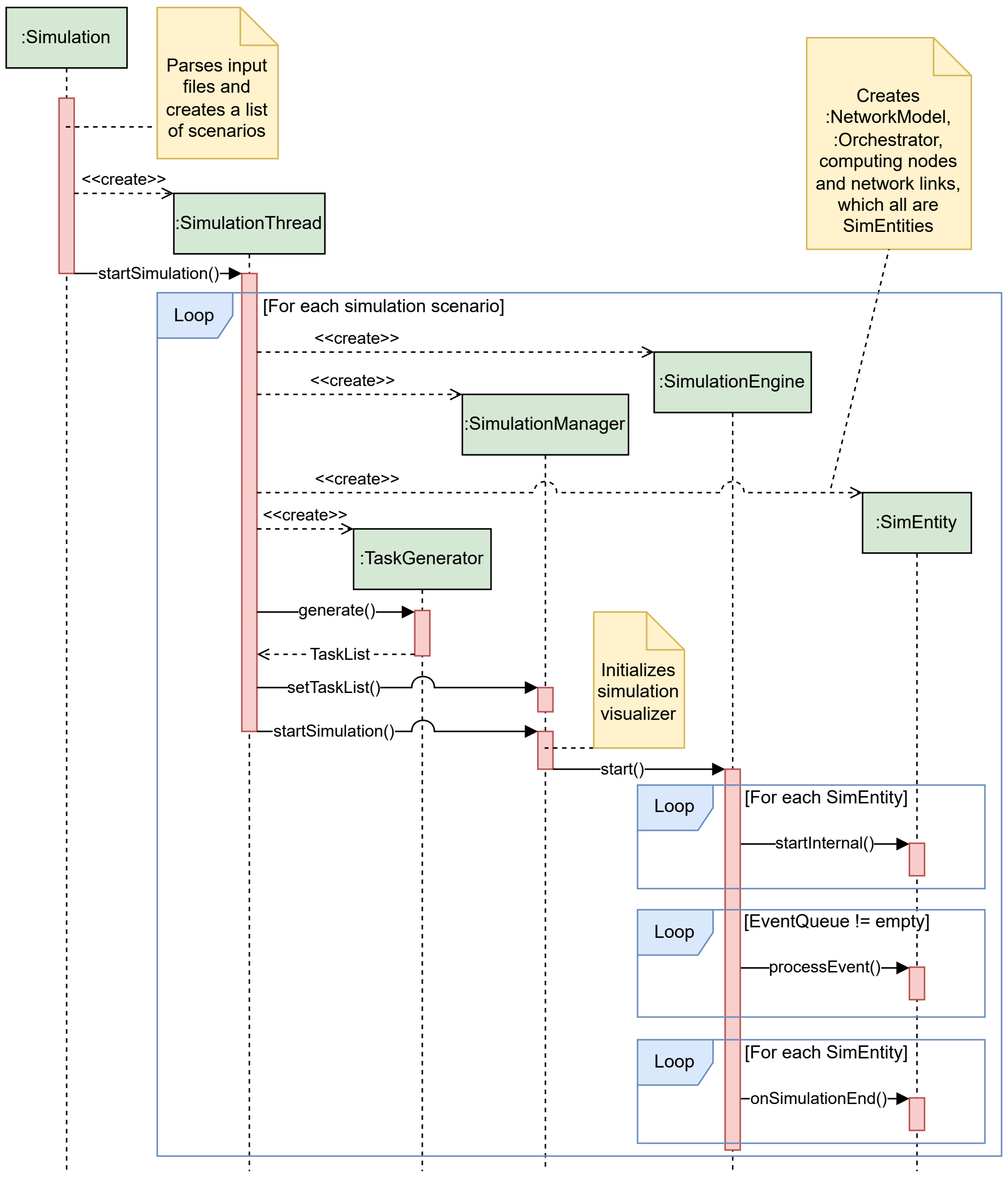


Figure 4. The workflow of PureEdgeSim simulation.

important to note that also the *SimulationManager* class inherits from the *SimEntity* class, which means that an instance of the *SimulationManager* class is also a simulation entity.

After the simulation entities have been created, the *SimulationThread* object creates an instance of the *TaskGenerator* class, which creates and returns a list that contains all the tasks to be generated during the simulation run. This task list is given to the *SimulationManager* object, as it is responsible for scheduling the task generation events. After this, the *startSimulation* method of the *SimulationManager* object is called. Then, the simulation manager initializes the simulation visualizer, after which it calls the *start* method of the simulation engine.

The *SimulationEngine* object first calls the *startInternal* method for each *SimEntity* object. This method is used to schedule initial simulation events. Then the simulation engine inspects the event queue and finds the earliest event. It advances the simulation clock to the time of the event and processes all the events that happen at the same time. Events are processed by calling the *processEvent* method of the *SimEntity* object to which the event has been scheduled. Then the engine finds the next event, advances the clock and processes all the events happening at the same time. This is continued until the event queue is empty, or until the simulation manager terminates the simulation. After this, the *onSimulationEnd* method is called for each *SimEntity* object. This method can be used to do something when the simulation ends. For example, the state of a simulation object could be saved.

4. EISIM IMPLEMENTATION AND EVALUATION

Like PureEdgeSim, EISim also allows the user to simulate a wide variety of scenarios and deployments. However, EISim has a specific focus on evaluating and comparing the performance of intelligent orchestration solutions against different orchestration control topologies. For this, EISim extends and modifies the core modules of PureEdgeSim, as well as adds new features and modules to PureEdgeSim. The following sections explain the architecture, default implementations and use of EISim, specify the changes made with regard to PureEdgeSim, and present the additional tools that come with EISim. Finally, the simulation case study that is used to evaluate and validate EISim is introduced.

4.1. Architecture

The architecture of EISim and the changes made with respect to PureEdgeSim are shown in Figure 5. EISim retains the core architecture of PureEdgeSim. The major changes are the addition of the agent model and clustering modules, and the completely new implementations of the task orchestration and application model modules. To support the new additions and new implementations, other modules had to be extended and modified. The following sections elaborate on the changes.

Clustering

The *Clustering* module is responsible for handling edge server clusters. EISim makes it possible to offer cluster information as a part of the edge datacenter specification in the *edge_datacenters.xml* setting file. For each edge datacenter, cluster information consists of a non-negative integer that specifies the cluster to which the server belongs, and a boolean value that indicates whether the server is the head of the cluster.

Implementing the clustering module required changes to the input file parsing, computing node generation, and computing node implementation. During simulation, each edge server is aware of its cluster members, but the cluster members remain static by default.

Task Orchestration

EISim implements three default orchestration algorithms that correspond to three main orchestration control topologies. The used algorithm is specified in the *simulation_parameters.properties* file by setting the value of *orchestration_algorithms* to be one of the following: *CENTRALIZED*, *HYBRID*, or *DECENTRALIZED*. It is important to note that only one of these algorithms can be given as input to EISim at a time, because each of them makes different assumptions about the edge server clustering. The decentralized orchestration algorithm assumes that each edge server forms a cluster on its own. The centralized orchestration algorithm, in turn, assumes that all edge servers belong to the same cluster with one assigned cluster head. The cluster head functions as a central orchestrator. Finally, the hybrid orchestration

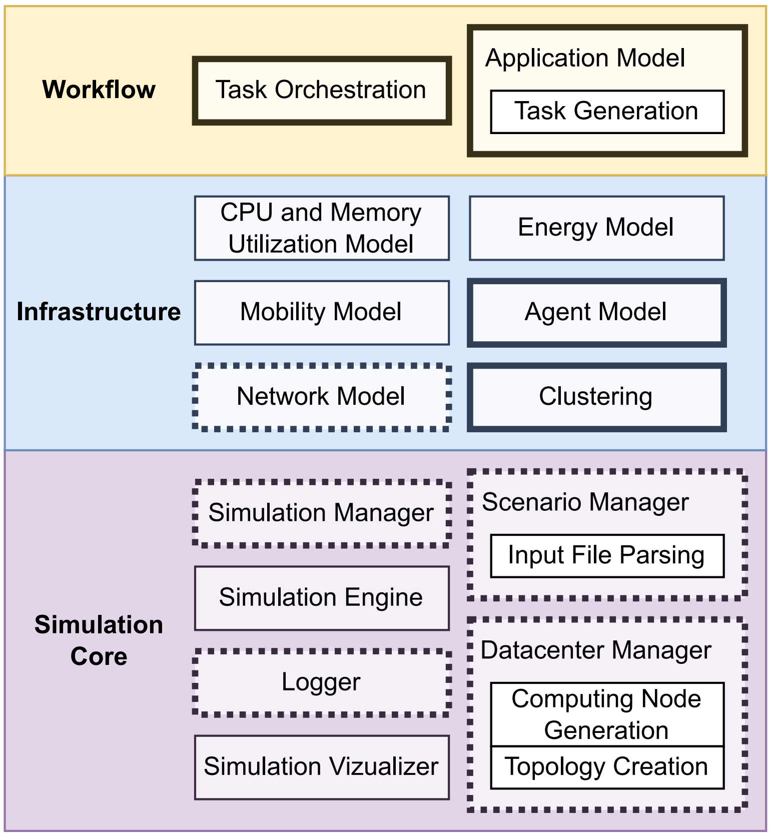
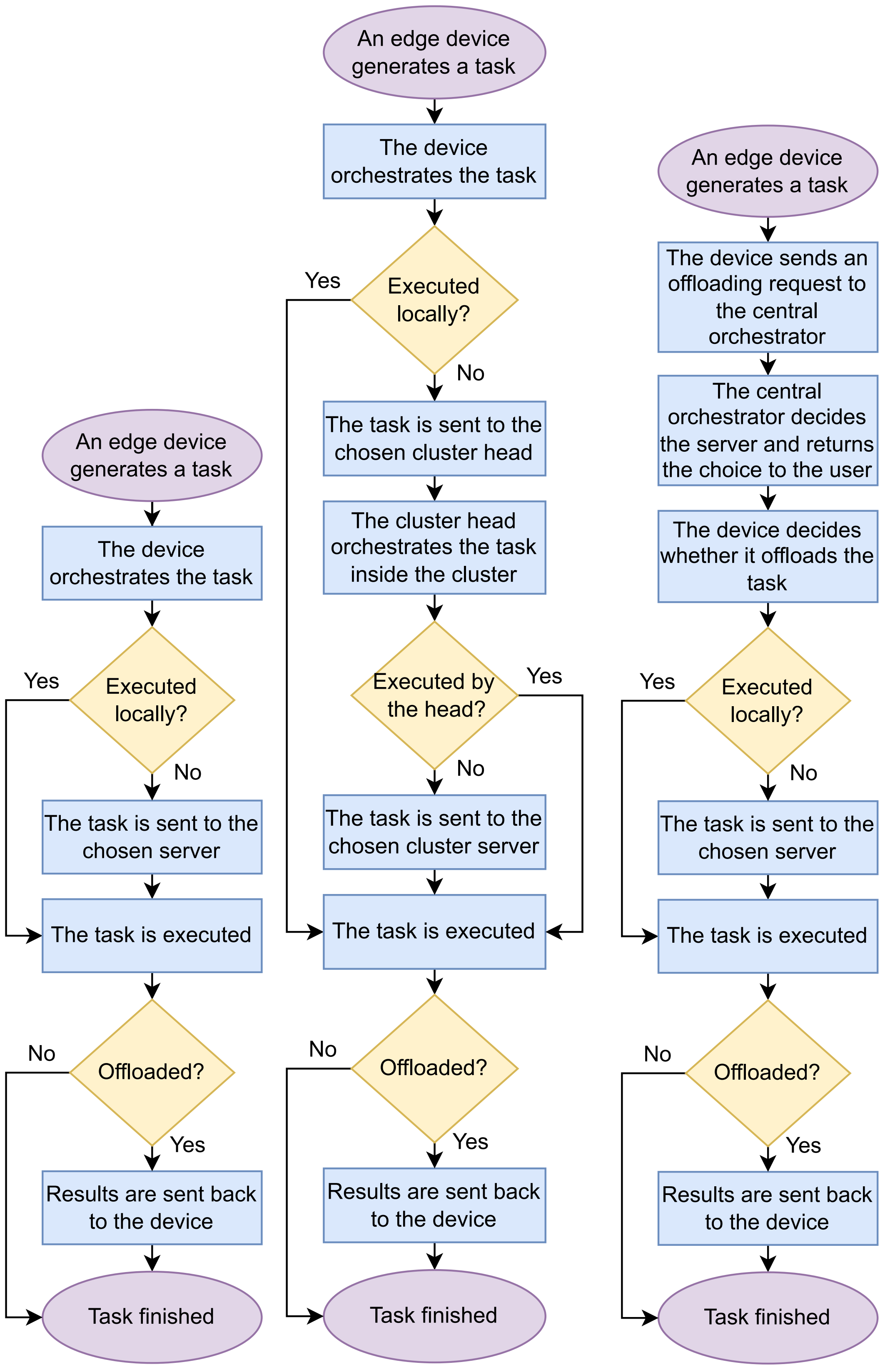


Figure 5. The architecture of EISim. Compared to Figure 3, thick solid line indicates either that the module is a completely new addition to PureEdgeSim, or that the implementation of the module has been completely changed. Thick dashed line indicates that the implementation of the module has been extended for the needs of EISim.

algorithm is intended for any type of grouping that resides between the decentralized and centralized extremes.

Each control topology has its own default orchestration workflow, as shown in Figure 6. Figure 6a shows the workflow of the decentralized control topology. Whenever a task is generated, the edge device orchestrates the task by deciding whether to offload and to which server. To make its decision, it uses information from the edge servers, which includes the prices set by the servers. The workflow of the hybrid control topology, as seen in Figure 6b, introduces a two-phase orchestration, where the edge device first decides whether it offloads and to which cluster. If the device decides to offload, the task is sent to the cluster head, which in turn allocates the task inside the cluster. Finally, in the workflow of the centralized control topology seen in Figure 6c, the device first sends an offloading request to the central orchestrator, which chooses the server for executing the task. The choice is returned to the device along with other necessary information. Then, the device makes the final decision whether it offloads or not.

Implementing the two-phase orchestration of the hybrid control topology required the most changes to the default simulation manager, network model and task model. It is also important to note that the default implementation of the centralized control


An edge device generates a task
The device orchestrates the task
Executed locally?
Yes
No
The task is sent to the chosen server
The task is executed
Offloaded?
No
Yes
Results are sent back to the device
Task finished
An edge device generates a task
The device orchestrates the task
Executed locally?
Yes
No
The task is sent to the chosen cluster head
The cluster head orchestrates the task inside the cluster
Executed by the head?
Yes
No
The task is sent to the chosen cluster server
The task is executed
Offloaded?
No
Yes
Results are sent back to the device
Task finished
An edge device generates a task
The device sends an offloading request to the central orchestrator
The central orchestrator decides the server and returns the choice to the user
The device decides whether it offloads the task
Executed locally?
Yes
No
The task is sent to the chosen server
The task is executed
Offloaded?
No
Yes
Results are sent back to the device
Task finished

(a) Decentralized (b) Hybrid (c) Centralized

Figure 6. The default task orchestration workflows of each control topology.

topology does not simulate the sending of the offloading request and its response through the network due to the minuscule sizes of such requests and responses.

More detailed explanations of the default decision making of both sides (edge devices and cluster heads) for each control topology are provided in Section 4.2.

Agent Model

The *Agent Model* module is responsible for handling the training, decision making and monitoring of agents. In the current form of EISim, an agent refers to a pricing agent. In each control topology, cluster heads function as pricing agents that decide a price for task execution on the resources in their cluster. For these agents, the system time is divided into slots and a new price decision is made at the beginning of a slot. The default slot length is five seconds.

EISim offers a DDPG-based default implementation for a pricing agent, but the users can easily plug in their own implementations. The hyperparameters for pricing agents are given as command-line arguments when starting the simulation. Note that every agent shares the same hyperparameters. More detailed explanations about the possible hyperparameters and the default state space, action space and reward function definitions for each control topology are provided in Section 4.2.

Each pricing agent in the simulation environment logs its state, price and profit for each slot, as well as calculates the cumulative profit over the simulation run. Further, each pricing agent saves its state at the end of a simulation run, and loads the state at the start of a next one. EISim creates agent- and scenario-specific folders for logging and saving the agent state.

Implementing the agent model required adding completely new packages and classes to PureEdgeSim, including argument parser, pricing agent and price logger related implementations. Further, the computing node implementation had to be modified to account for the existence of a pricing agent.

Application Model

EISim changes the application model of PureEdgeSim into a more versatile one. The original PureEdgeSim implementation assigns deterministic task generation rates, task lengths and sizes for each application type. This means that every device that has the same application type generates identical tasks that have exactly the same input, output and container sizes, as well as the same task length in MIs. To generate more realistic tasks that also conform to often used task models in the literature (see Section 2.4.3), EISim takes an approach where, for each application type, task generation rates, task lengths and sizes are stochastic with specified expected values.

In EISim, task generation rate refers to the rate of a Poisson process, meaning that it gives the expected number of task arrivals during a time unit. This is a commonly used way to model task arrival in the literature (see, e.g., [76, 84, 88, 92, 100]). The task input and container sizes, in turn, are drawn uniformly from a specified range. This is another often used method to model task heterogeneity in the literature (see, e.g., [89, 92, 93, 94, 104]). Finally, the output size is determined as a ratio of the task's input size, which is a typical way to model the result size in the few works that do not ignore the return of the task results [89, 94, 104]. This ratio is also drawn uniformly from a specified range. Note that setting the minimum and maximum values of the range to the same value simplifies to the deterministic input, output and container sizes of PureEdgeSim.

In PureEdgeSim, and consequently in EISim, container size is used when registry is enabled for downloading containers. Container size is also used in the default CPU and memory utilization model to specify how much RAM and storage a task uses on a computing node. Input size (also called request size), in turn, is used when sending a task through the network. In the cases where registry is not used, it may be desirable to set the values of a task's container size and request size equal. This is supported in EISim, as setting the minimum and maximum values of the container size to zero makes EISim use the randomly drawn request size also as a container size.

The task length in MIs is drawn from an exponential distribution, which has been used to model task durations in the literature [128]. Hence, for each application type, the given value of task length is interpreted as the expected value of the exponential distribution. Finally, EISim also allows specifying one latency constraint for an application type, which is the same for all the tasks. It can be understood as a desirable maximum execution time for a given application type.

Implementing the new application model required completely new implementations of the application, task generator, and application file parser classes.

Improvements

EISim also modifies the core modules of PureEdgeSim to introduce a set of improvements over the original implementation. These improvements include the creation of edge nodes that only function as APs, the reproducibility of the simulation results, as well as better extensibility of the simulator.

APs are specified alongside edge datacenters in the *edge_datacenters.xml* setting file. Each datacenter element in the file must have a name attribute that contains 'dc' if the node is an edge datacenter, and 'ap' if the node is an AP. These AP nodes do not have any computational capabilities, and they cannot be a part of the edge server clustering. They only route traffic as a part of the MAN. The MAN links between APs and edge datacenters are defined in the same file.

EISim uses a seed generator that seeds all the random number generators used in the simulation. The user of the simulator can provide a seed for the seed generator through command-line arguments. This allows reproducing the results of a simulation run.

Finally, EISim improves the extensibility of the simulator by allowing users to plug in their own implementation of the computing node generator. This is a new addition to the set of the extensible modules in PureEdgeSim (see Section 3.3.2).

4.2. Default Implementations

For each control topology, EISim implements default decision-making algorithms for price and offloading decisions. These default implementations aim to embrace realism, that is, the algorithms are designed so that they could be potentially deployed under a more practical setting in a large-scale, highly dynamical system.

The default implementations come with a set of assumptions about the environment. First of all, they assume that each edge device that generates tasks is its own orchestrator, meaning that it makes the final decision about whether to offload or execute locally. Second, the only potential task execution locations besides the edge device itself are edge servers. Third, all the edge servers are assumed to belong to the same ESP. Finally, all the edge servers are assumed to be homogeneous in terms of capacity.

The assumption of homogeneous capacity is justifiable in the simple simulation environment, where edge devices are first located uniformly at random, after which, in case the default mobility model is used, they may move according to randomly drawn mobility and pause durations. Hence, by default, the simulation area does not exhibit distinct areas with different population densities, the existence of which would create a need for placing higher capacity servers to denser areas [113]. Further, such assumption of homogeneity has also been used in edge server placement related works (see, e.g., [109, 129, 130, 131]).

4.2.1. Price Decisions

In each control topology, the edge platform functions as a time-slotted system, where at the beginning of each slot, the pricing agents decide a price for the task execution. The chosen default pricing scheme is uniform pricing for a task's computational demand, which is the most common pricing method in the existing literature (see Table 3). Consequently, a pricing agent sets a price per MI.

The goal of each pricing agent is to maximize the expected long-term profit. Profit is defined as the revenue obtained from the offloading devices minus the processing costs. By default, the processing costs only include the fixed and varying energy costs.

The pricing agents are trained using the DDPG algorithm (see Section 3.2.3 and Algorithm 3). The default structures for the critic and actor networks are as follows. Critic is a feedforward, fully connected network with two hidden layers, each of which has 64 units and ReLU activation. The final activation before output is linear. Actor has the same structure as critic, but the final activation before output is tanh. Both networks are randomly initialized.

As the decision making in the environment is a continuous task without any terminal states, simulation runs can be considered as pseudo-episodes over which the pricing agents can be trained and evaluated. Agents save their state at the end of a simulation run (episode), and load the state at the beginning of a new one. The agent state in the default DDPG implementation consists of the actor and critic networks, their target counterparts, experience replay, and the state of the noise process.

The number of training steps in an episode depends on the price update interval ι and the total length of the simulation. The updating of the models is started when the size of the experience replay $|\mathcal{D}|$ is larger than the mini-batch size B.

There are several hyperparameters that are used to control the training process: the size of the experience replay $|\mathcal{D}|$, the size of a mini-batch B, discount factor γ, actor and critic learning rates, τ for updating the actor and critic target networks, the number of model updates done at the beginning of a new slot, and the parameters for the noise process. Further, EISim allows specifying a number of random decision steps that are done by each pricing agent at the beginning of an episode to improve exploration. A random decision step is a step during which an agent chooses its action uniformly at random. The values for all of these are given as command-line arguments to EISim. The default values can be seen in Appendix 1 Table 7.

The action space definition is the same for every pricing agent regardless of the control topology. The action space is continuous, consisting of a single real-valued variable between zero and one that corresponds to the price p_t for a slot t. In other words, $\mathcal{A} \triangleq [0,1]$. The definitions of the state space and reward function depend on the control topology.

In the decentralized control topology, each of the edge servers makes a price decision independently. For one edge server, the state $s_t \in \mathcal{S}$ at the beginning of a slot t is defined as (l_t, λ_{t-1}). Here, l_t is the length of the task queue at the beginning of the slot t, and λ_{t-1} is the average arrival rate of the tasks in the previous slot $t-1$ (total number of the arrived tasks divided by the slot length ι). The immediate reward for a server is given as

$$R(s_t, p_t, s_{t+1}) = p_t Q_t^{MI} - \zeta_e \Big(\iota P_{idle} + (P_{max} - P_{idle}) \frac{Q_t^{MI}}{n^c f} \Big), \tag{10}$$

where p_t is the price per MI, Q_t^{MI} is the total number of MIs summed over all the tasks that were offloaded to the server in slot t, ζ_e is an energy cost coefficient that defines a cost per joule (J), ι is the slot length in seconds (s), P_{idle} is the power consumption of the server in Watts (W) when the CPU is idle, P_{max} is the power consumption (W) of the server when the CPU is at 100%, n^c is the number of cores in the server, and f is the computational capacity of one core (MIPS).

In Equation (10), $p_t Q_t^{MI}$ is the revenue from the offloading devices, $\zeta_e \iota P_{idle}$ is the fixed energy cost in a slot, and $\zeta_e (P_{max} - P_{idle}) \frac{Q_t^{MI}}{n^c f}$ is the varying energy cost that takes into account the excess energy consumption in task processing. The fixed energy cost measures the baseline, load-independent energy cost of the edge server during a slot. The idea of the varying energy cost is to capture the dynamic, load-dependent energy cost. The varying cost calculates how long it takes from the server to process all the arrived tasks with its total computational capacity and multiplies this value with the excess energy consumption.

In the hybrid control topology, each of the cluster heads makes a price decision independently. Now the state for a cluster head is defined as $(l_t^{avg}, \lambda_{t-1})$. Here, l_t^{avg} is the average queue length in the cluster at the beginning of a slot t, and λ_{t-1} is the average arrival rate of the tasks in the previous slot $t-1$ (total number of the tasks arrived in the cluster divided by the slot length). The reward function for a cluster head is defined as

$$\begin{aligned} R(s_t, p_t, s_{t+1}) &= p_t Q_t^{MI} - \zeta_e \Big(|\mathcal{C}| \cdot \iota P_{idle} + |\mathcal{C}| \cdot (P_{max} - P_{idle}) \frac{Q_t^{MI}}{|\mathcal{C}| \cdot n^c f} \Big) \\ &= p_t Q_t^{MI} - \zeta_e \Big(|\mathcal{C}| \cdot \iota P_{idle} + (P_{max} - P_{idle}) \frac{Q_t^{MI}}{n^c f} \Big), \end{aligned} \tag{11}$$

where Q_t^{MI} is now the total number of MIs summed over all the tasks that were offloaded to the cluster in slot t, and $\mathcal{C}$ is the set of the edge servers in the cluster.

Compared to the reward function in the decentralized control topology (Equation (10)), the fixed energy cost now takes into account the baseline energy consumption of all the edge servers in the cluster. The varying energy cost, in turn, calculates how long it takes to process all the arrived tasks with the cluster's total computational capacity and multiplies this value with the total excess energy consumption of the cluster.

In the centralized control topology, where the central orchestrator is the only pricing agent in the environment, all the servers form one cluster. Hence, the state space and reward function are defined as in the hybrid control topology.

The immediate reward values given by Equation (10) and Equation (11) can have very large magnitudes. Consequently, the training targets of the critic network have a large output scale, which may cause instability in learning [132]. It has been observed that scaling the rewards with a constant factor may improve the performance of DDPG in some environments [133]. The default implementation of EISim also scales the immediate rewards with a factor of 1e-3 in the decentralized control topology, and with a factor of 1e-4 in the hybrid and centralized control topologies. These factors can be changed by modifying or extending the default implementation.

4.2.2. Task Orchestration Decisions

Each edge device makes offloading decisions independently based on the information provided by the edge platform. For practical considerations, it can be assumed that the environment has a repository (distributed or centralized), where the devices can fetch information about the edge servers, such as the current prices, estimates of queuing time, computational capacities, IP addresses and locations, to support their decisions about offloading destinations. However, such repository is not simulated.

The default implementation of EISim allows each edge device to be connected to only one AP at a time. Further, as EISim considers each task to be independent and atomic, for practical considerations, it can be assumed that the edge device handles the partition of the application into adequate sized tasks, as well as handles the dependencies between tasks.

After an offloaded task has been executed, the results are sent directly back to the edge device. Note that if the device moves to the coverage area of another AP during offloading, the default implementation of EISim does not fail the task. Instead, the task result is rerouted through the MAN to the new location.

Formally, a task is defined as a tuple $(c, d_{in}, d_{out}, D_{max})$, where c is the computational demand of the task in MIs, d_{in} is the length of the input data in bits (can contain software code in addition to input files/parameters), d_{out} is the length of the output data in bits, and D_{max} is the maximum tolerable delay of the task in seconds. EISim sets the task as failed due to delay if the task processing time exceeds D_{max}.

The current default implementation of EISim formulates the offloading decision problem for each task as a one-shot optimization problem without considering the future effects of the task offloading decisions. The decision of an edge device is based on an utility that takes into account the task execution delay, energy consumption and resource price.

The following sections elaborate the edge devices' decision making and utility for each of the control topologies. Further, it is also explained how the cluster heads orchestrate tasks inside the clusters in the hybrid control topology, and how the central orchestrator decides the server for executing a task in the centralized control topology. For the discussions that follow, it is assumed that there are N edge servers in total, and the edge servers are grouped into K clusters in the hybrid control topology. Further, it is assumed that the computing nodes (edge devices and edge servers) use the default CPU utilization model of EISim, which uses First In First Out (FIFO) scheduling and assigns one task to be completely executed by one CPU core.

Decentralized

In the decentralized topology, a device's decision variables form an $N+1$ length vector **x** indicating the offloading destination (local node + N servers); that is, the decision variable $x_j \in \{0, 1\} \; \forall j = 0, \ldots, N$ and $\sum_j x_j = 1$. The device's utility is defined in terms of minimizing the cost of task execution. The cost consists of the execution delay, energy consumption and execution price. Formally, the optimization problem of the edge device can be defined as

$$\min_{\mathbf{x}=[x_0,\ldots,x_N]} \sum_{j=0}^{N} x_j (w_d \frac{D_j}{D_{max}} + w_e \frac{E_j}{B_e} + w_p \frac{p_j}{p_{pref}}) \tag{12}$$

$$\text{s.t. } \; x_j \in \{0, 1\} \; \forall j = 0, \ldots, N \tag{13}$$

$$\sum_j x_j = 1 \tag{14}$$

$$E_j \leq B_e \tag{15}$$

Here, D_j is the task execution delay (s), E_j is the energy consumption of the device (J), and p_j is the current price per MI. The values of these depend on the offloading destination, indicated by x_j. It is good to note that the price per MI for local node p_0 is zero. w_d, w_e and w_p are device-specific, normalized weights (i.e., $w_d + w_e + w_p = 1$), which indicate the importance of each factor (delay, energy consumption, price) for the edge device. EISim generates these weights for each device at the start of the simulation. The possible weight values lie on a triangle formed by the points (1,0,0), (0,1,0) and (0,0,1) inside a unit cube, and the weights are generated by sampling a point randomly from the triangle.

D_{max}, B_e and p_{pref} are used to normalize the values of D_j, E_j and p_j, as well as make each quantity of the cost dimensionless. D_{max} is the maximum tolerable delay of the task (s), B_e is the battery level of the edge device (J), and p_{pref} quantifies how much the edge device prefers to pay per MI. The default implementation uses $p_{pref} = 0.01$ for every edge device.

The constraint in Equation (13) ensures that each decision variable is binary. The second constraint in Equation (14) ensures that only one offloading destination is selected. Finally, the constraint in Equation (15) ensures that the energy consumption does not exceed available energy.

The problem in Equation (12) is an integer programming problem, which in general is NP-hard [134]. However, due to the constraint in Equation (14), the feasible solution set has $N + 1$ elements. Consequently, the edge device can solve the problem in linear time ($\mathcal{O}(N)$) by calculating the cost $w_d \frac{D_j}{D_{max}} + w_e \frac{E_j}{B_e} + w_p \frac{p_j}{p_{pref}}$ for each x_j and setting $x_j = 1$ for the destination j with the lowest cost.

Calculating the task execution delay D_j. When $x_0 = 1$, the offloading destination is the local node itself. The local task execution delay D_0 is the sum of the processing delay and the queuing delay at the local node. The processing delay is calculated as $\frac{c}{f_0}$, where f_0 is the processing capacity of one core in MIPS. The queuing delay is approximated by summing the task lengths of all the tasks currently in the queue and dividing the sum with the total processing capacity of the edge device, which is $n_0^c f_0$, n_0^c being the number of cores in the device's CPU. Denoting the sum of task lengths as Q_0^{MI}, D_0 is calculated as $D_0 = \frac{c}{f_0} + \frac{Q_0^{MI}}{n_0^c f_0}$

When $x_j = 1$ for some $j \in \{1, \ldots, N\}$, the offloading destination is the edge server j. The task delay D_j at the edge server j consists of a communication delay and an execution delay. The communication delay is the sum of transmission and propagation delays. The transmission delay from the edge device to an AP is $\frac{d_{in}}{r^u}$, where r^u is the uplink transmission rate for the device. The transmission delay of the task result from an AP to the device is $\frac{d_{out}}{r^d}$, where r^d is the downlink transmission rate for the device. For calculating the propagation delay, the default implementation simply sums the link latencies of the shortest path between the edge device and the server j.

For practical considerations, the edge device could obtain uplink and downlink transmission rates, for example, by measurement or as a result of a separate communication optimization problem, where the interference and the noise power are measured and the transmission power adaptively updated based on the measurements (see, e.g., the power control scheme in [135]). The propagation delay could be approximated based on the Euclidean distance between the edge device and the edge server, and a conversion factor that estimates the latency per distance unit. However, EISim does not simulate any schemes for obtaining estimates of the rates and the propagation delay, it uses the given input values for link bandwidths and latencies directly.

The execution delay on a server j is the sum of the processing and queuing delays. The processing delay is calculated as $\frac{c}{f}$. The queuing delay is approximated based on a simple heuristic. Whenever an edge server j handles the price update event, it also calculates an estimate of the queuing time with $\frac{Q_j^{MI}}{n^c f}$, where Q_j^{MI} is the total number of MIs summed over all the tasks currently in the queue of the server j, and $n^c f$ is the total processing capacity of the server j (same for all j). EISim uses this estimate as the queuing delay. The idea here is to mimic the practical situation where such estimate would be announced to the edge devices alongside the price. It gives a crude approximation of the queuing time, but can be considered to be an indication of the queuing delay in the case where the slot length ι is short.

Calculating the energy consumption E_j. The local energy consumption E_0 consists of the energy that the edge device spends on the task execution. This is calculated as $E_0 = P_{max} \frac{c}{n_0^c f_0}$. The idea here is to calculate the time it would take from the device to process the task if the whole processing capacity was used, and then multiply the time with the maximum power consumption of the CPU. This can also be interpreted as assuming that the power consumption of one core is $\frac{P_{max}}{n_0^c}$ when it is processing a task.

The energy consumption E_j when the device offloads the task to an edge server j is the same for all j. It consists of the energy the device spends on sending and receiving data. If P_t is the device's transmission power (W) and P_r is the device's receiver power (W), the total energy consumption is calculated as $E_j = P_t \frac{d_{in}}{r^u} + P_r \frac{d_{out}}{r^d}$.

Hybrid

In the hybrid control topology, a device's decision variables form a $K + 1$ length vector $\mathbf{x}$ indicating the offloading destination (local node + K clusters). That is, the decision variable $x_k \in \{0, 1\}\ \forall k = 0, \ldots, K$ and $\sum_k x_k = 1$. The device's optimization problem is formulated equivalently to Equation (12), but now there are $K + 1$ components in $\mathbf{x}$ instead of $N + 1$. The problem is also solved in the same way by iterating over all $K + 1$ options, calculating the cost $w_d \frac{D_k}{D_{max}} + w_e \frac{E_k}{B_e} + w_p \frac{p_k}{p_{pref}}$ for each k and setting $x_k = 1$ for the destination cluster k with the lowest cost.

The energy consumption $E_k\ \forall k = 0, \ldots, K$ and the local task execution delay D_0 are calculated exactly as explained for the decentralized control topology, because their values depend only on the device's local information. The task execution delay D_k at the cluster k is the sum of the communication and execution delays. The communication delay is the sum of the transmission and propagation delays, which are calculated as explained for the decentralized topology. The calculation of the propagation delay only takes into account the communication distance between the edge device and the head of the cluster k. The delays inside the cluster are ignored, because the clusters have been formed based on proximity.

The execution delay inside the cluster is the sum of the processing and queuing delays. The processing delay is calculated in the same way as in the decentralized topology. The queuing delay is again approximated with a simple heuristic. Whenever the head of the cluster k handles the price update event, it calculates an estimate of the queuing time with $\frac{Q_k^{MI}}{|\mathcal{C}_k| n^c f}$, where Q_k^{MI} is the total number of MIs summed over all the queues and tasks in the cluster k, and $\mathcal{C}_k$ is the set of the servers in the cluster k. The idea here is to calculate how long it takes from one server inside the cluster to clear its task queue, and then use the average of these times as an estimate of the queuing time to the edge devices. Once again, this is a crude approximation, but it provides an indication of the congestion level inside the cluster, given that the slot length ι is short.

When an offloaded task arrives at a cluster head, the head must decide how the task is allocated inside the cluster. The default implementation allocates the incoming tasks based on a bottom-up strategy [128], meaning that a task is allocated to the server with the lowest workload. The workload is measured in terms of the task queue length.

Centralized

In the centralized control topology, a device's decision variables form a vector $\mathbf{x}$ of length two indicating the offloading destination (local node or the edge platform). As previously, $x_l \in \{0, 1\}\ \forall l = 0, 1$ and $\sum_l x_l = 1$. The offloading decision reduces to setting $x_1 = 1$ (offload to edge) if $w_d \frac{D_1}{D_{max}} + w_e \frac{E_1}{B_e} + w_p \frac{p_1}{p_{pref}} \leq w_d \frac{D_0}{D_{max}} + w_e \frac{E_0}{B_e}$, otherwise $x_0 = 1$ (process locally).

For practical considerations, the edge device can calculate D_0, E_0, and E_1 based on the local information. For calculating D_1, the device can estimate the transmission time of the task and the latency to the AP. Then the device can send an offloading request to the central orchestrator, informing it about the task characteristics and the device's location. The central orchestrator can calculate the task processing time, estimate propagation delays inside the MAN, and collect queue delay information from the edge servers. Using the information about processing time and delays, it chooses the server with lowest estimated delay. Then it informs the device about the chosen server, estimated delay, and the price p_1, after which the device can decide whether it offloads to the given server or not. The device can directly send the task to the chosen edge server and get the result from it.

As EISim does not simulate the sending and receiving of the offloading requests and responses, the actual default implementation calculates the value of D_j for each edge server j in the same way as explained for the decentralized control topology, and chooses the location that has the lowest estimated cost. The main implementation differences with regard to the decentralized control topology are the use of only one price set by the central orchestrator, and a new event for every edge server that makes them record their queue delay estimate at the beginning of every price slot. The idea here is to mimic the fact that the central orchestrator would collect status information from the edge servers only at the beginning of each price slot in order to reduce overhead, and then use this information when it decides task execution locations during the slot.

4.3. Use and Extensibility

EISim is available on GitHub[1]. It is built on Java SE Platform and uses Maven[2] as a build automation tool. Running simulations with EISim requires five input files: *cloud.xml*, *edge_datacenters.xml*, *edge_devices.xml*, *applications.xml*, and *simulation_parameters.properties*. The definitions in *cloud.xml*, *edge_devices.xml*, and *simulation_parameters.properties* have the same content as in PureEdgeSim (see Section 3.3.1). However, it is important to note that using the default implementation of EISim as is expects that the value of *enable_orchestrators* in the simulation parameters file is set as false, and the only value for *orchestration_architectures* is *EDGE_ONLY*. This is due to the assumptions made in the default implementation (see Section 4.2). The definitions in *edge_datacenters.xml* and *applications.xml* have been modified as explained in Section 4.1.

EISim uses command-line arguments to facilitate the setting of the training hyperparameters. Command-line arguments are also used to provide the folder that contains the input files to EISim, as well as to specify an output folder for saving the simulation results, and a model folder for saving the states of the pricing agents. By default, EISim runs in evaluation mode, meaning that the pricing agents in the environment expect to find trained models in the provided model folder. To run the simulation in training mode, it must be turned on with a flag option. All possible options with their descriptions and default values for the current implementation of EISim can be seen in Appendix 1 Table 7.

[1]`https://github.com/hennas/EISim`

[2]`https://maven.apache.org/`

For implementing the deep learning abilities of the pricing agents, EISim uses Deeplearning4j library[3], which is one of the very few deep learning libraries available for Java. By default, EISim uses the native CPU backend for executing the DNN related computations. This can be easily changed to CUDA GPU backend by changing the value of the *nd4j.backend* property in the Maven project's *pom.xml* file to *nd4j-cuda-X-platform*, where X is the CUDA version. The currently used version of Deeplearning4j (1.0.0-M2.1) supports CUDA versions 11.4 and 11.6.

The *Main* class of EISim is the entry point to the simulator. It parses the command-line arguments and creates the EISim simulation object. The users can modify or replace this class in order to add in their custom implementations. The extensibility is high, as the EISim simulation object offers eight methods that can be used to set a custom implementation class for mobility model, network model, orchestrator, simulation manager, topology creator, computing node generator, computing node, and task generator. Further, a custom task class can be set through the task generator class, and a custom pricing agent class can be set through the computing node class. EISim contains an abstract class for each customizable part of the simulator, and every provided custom class must inherit from the corresponding abstract class.

4.4. Additional Tools

EISim facilitates the research by offering additional tools for environment setup, agent training and result plotting. Even though the simulator itself uses Java programming language, the additional tools for the environment setup and result plotting are made with Python programming language. This is due to Python's simplified syntax and ease of use. Jupyter notebooks[4] are used as an interactive environment for running the Python codes.

Environment Setup

Environment setup consists of three Jupyter notebooks. The first notebook creates the MAN by using the methods explained in Section 3.1.1. The second notebook creates the edge server clusters and assigns the cluster heads by using the methods explained in Section 3.1.2. The final notebook uses the MAN and cluster information saved by the previous notebooks to automatically create the *edge_datacenters.xml* setting file.

The creation of the *edge_datacenters.xml* setting file requires providing the specifications for the edge datacenters. As it is assumed in the current form of EISim that there is only one ESP in the area and that all the edge servers are homogeneous, only one edge server specification needs to be provided. This specification is used for all the edge datacenters in the resulting file.

Agent Training

EISim offers a set of bash scripts that can be used as templates for running simulations for hyperparameter tuning, training and evaluation of the pricing agents.

[3]`https://deeplearning4j.konduit.ai/`
[4]`https://jupyter.org/`

By default, the hyperparameter tuning scripts do a grid search over actor and critic learning rates, testing in total nine combinations. For each combination, the models are trained for 10 rounds with different seeds, after which they are evaluated for five rounds with different seeds. The training scripts, in turn, train the model for 100 rounds with different seeds, plotting the training progress every 20th round. Finally, the evaluation scripts run five evaluation rounds with the trained models, using different seeds.

Result Plotting

EISim provides Python codes for plotting the results of each phase of the simulations. For hyperparameter tuning, there is a Jupyter notebook file that can be used to plot two figures for each simulation scenario. The first figure depicts the average cumulative return (total profit) of the whole edge platform and its confidence interval for each hyperparameter combination. The total profit of the edge platform is the sum of the cumulative returns of the pricing agents. The second figure depicts the same metrics, average cumulative return and its confidence interval, for each pricing agent and each hyperparameter combination. The averages are calculated over the evaluation episodes.

For training, EISim provides a Python file that can be used to observe the training progress of the pricing agents. For each simulation scenario, it plots the cumulative return of the whole edge platform and the cumulative return of each pricing agent against the training episodes. It also plots the average price of each agent against the training episodes. The training bash scripts use this file to create the plots every 20th training round.

Finally, there is a Jupyter notebook file that can be used to create plots for comparing the final performance of the system after several evaluation episodes have been run with the final trained models. The main axis of comparison in the plots is the three different control topologies. Averages and confidence intervals of several metrics related to the task execution times, task failures, network usage, CPU usage, energy consumption and total profit are plotted.

Confidence intervals in plots are calculated with $\bar{m} \pm t^*_{n_e-1}\frac{s_m}{\sqrt{n_e}}$, where $\bar{m}$ is the mean of the metric, s_m is the sample standard deviation, n_e is the number of evaluation episodes, and $t^*_{n_e-1}$ is the upper $\frac{1-C}{2}$ critical value for Student's t-distribution with n_e-1 degrees of freedom. C is the confidence level, which can be given as an argument to the plotting function by the user. By default, $C = 0.95$.

4.5. Evaluation

To verify the end-to-end performance of EISim and to demonstrate the capabilities of EISim particularly with regard to training agents and evaluating orchestration solutions against control topologies, a simulation case study was conducted. The study focused on a large-scale MEC scenario, where mobile users, such as laptops, smartphones and tablets, move in a city area and generate independent tasks. All three control topologies with their associated default pricing and offloading decision-making implementations were simulated on this area. The following sections explain the simulation scenarios, environment and settings used in the simulation study.

4.5.1. Scenarios

For each control topology, eight scenarios are simulated, totalling in 24 simulation scenarios. One whole scenario consists of the control topology, edge server count, and user count. For the edge server count, two options are considered. In the first option, the ESP has located 20 high-capacity servers in the city area. In the second option, the ESP has located 100 low-capacity servers in the city area. The interest in choosing these two options is to compare the performance of different control topologies in contrary situations, where there is either only a small number of high-capacity servers or a large number of low-capacity servers. To mimic a large-scale system, the number of mobile users is varied from 1000 to 4000 with a step size of 1000.

In every scenario, the AP locations and MAN topology are the same. Further, every mobile user has the same application type that generates computationally intensive tasks with a relatively strict latency constraint and varying input and output data sizes. This application type was chosen because it is the most relevant in terms of the future applications, as it has been envisioned that there will be computationally demanding intelligent applications with strict latency requirements. Further, this type aims to capture the most suitable application type for offloading, which is an application that generates computationally demanding tasks with moderately small data sizes [60].

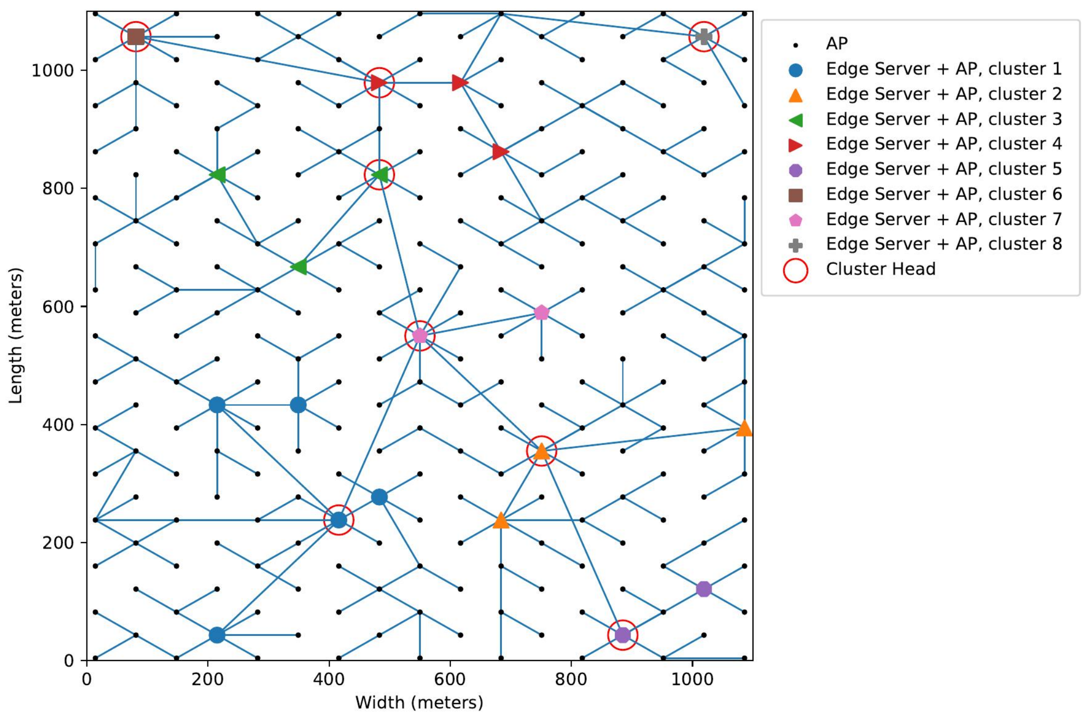
4.5.2. Simulation Environment

The simulation environment is a square area with a side length of 1100 meters. The MAN was created on this area using the environment setup tools of EISim. The AP coverage for placing the APs was set to 45 meters, which resulted in the placement of 247 APs to the area. A tree-topology was created for the APs using the TWST algorithm. The TWST weight parameter κ was set to 0.5. For co-locating the 20 high-capacity servers with the APs, the value of the parameter ν was set to 5. For placing the 100 low-capacity servers, the parameter ν was set to 3.

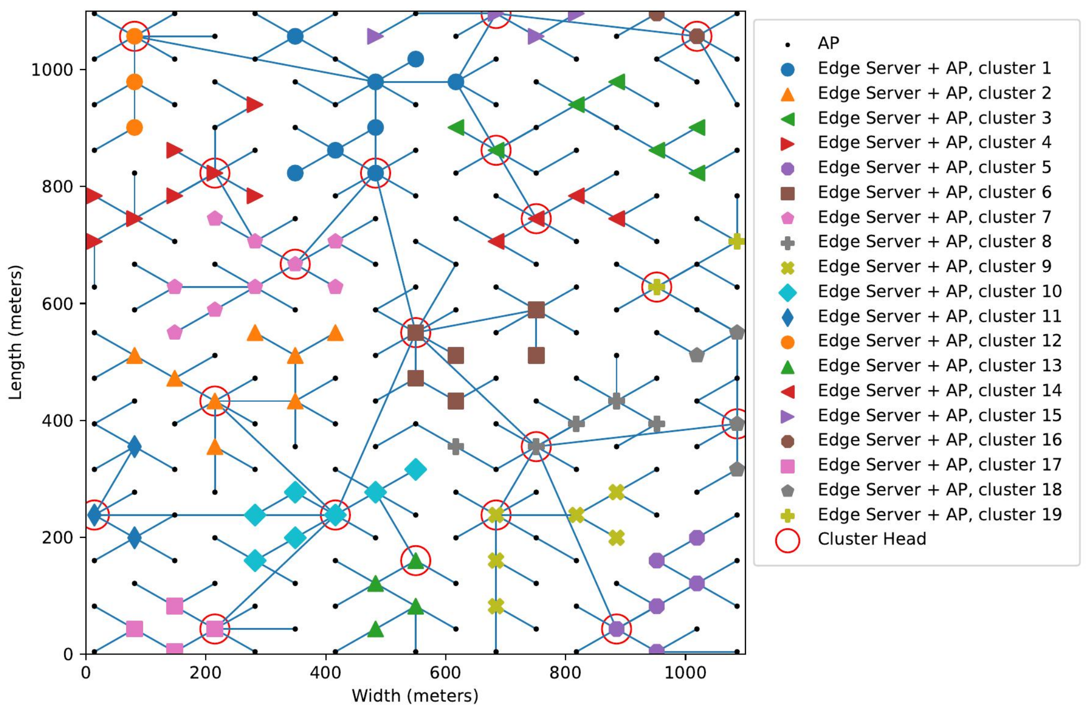
For finding the edge server clusters for the hybrid control topology, all of the supported linkage criteria (single, complete and average) were tested in both edge server scenarios. The distance thresholds were determined based on the dendrogram plots. The final clusters for the 20 high-capacity edge servers were obtained by using the average linkage and a distance threshold of 470, resulting in $K = 8$. The final clusters for the 100 low-capacity servers, in turn, were obtained with the complete linkage and a distance threshold of 500, resulting in $K = 19$.

The resulting AP placement and MAN topology can be seen in Figure 7. Figure 7a shows the edge server placement and clustering result for the 20 high-capacity edge servers. Figure 7b, in turn, shows the edge server placement and clustering result for the 100 low-capacity servers. Even though the created environment is idealized, it can be seen to correspond to a dense deployment of APs in a city centre area.

It is important to note that in the centralized control topology, where all the servers can be seen to form one cluster, the central orchestrator was chosen in the same way as the cluster heads in the hybrid control topology. In both edge server placement scenarios, the edge server in the middle of the area (see Figure 7) was chosen as the central orchestrator.



(a) 20 edge servers



(b) 100 edge servers

Figure 7. AP locations, MAN topology and edge server placement in the simulation environment. The edge server clusters and cluster heads for the hybrid control topology are also shown.

4.5.3. Specifications and Settings

The edge_datacenters.xml file. The specifications of the edge servers for the high-capacity and low-capacity server scenarios are shown in Appendix 2 Table 8. Note that the specifications are done so that the total MIPS, RAM and storage of all the servers are the same in both scenarios. Here the interest is in examining performance differences when the total capacity in the system is the same, but it is more distributed in the low-capacity server scenario. It is also good to note that the values of the energy consumption rates were derived based on examining the specifications and power consumption measurement results of actual servers, such as Lenovo ThinkSystem SE350.

The edge_devices.xml file. Four edge device types are specified for the simulation scenarios, three of which are mobile. The specifications can be seen in Appendix 3 Table 9. Note that the types can be seen to correspond to a higher capacity smartphone, a lower capacity smartphone, a tablet, and a laptop, respectively. All edge device types use Wi-Fi connectivity and are battery-powered. The battery capacities and energy consumption rates were derived based on the measured values used in PureEdgeSim evaluation [18], and by examining the specifications and power consumption measurement results of different real devices.

The applications.xml file. The specification of the used application type is shown in Appendix 4 Table 10. Each edge device generates one task per time unit on average. The tasks are computationally demanding on average with a strict latency constraint. The input data size is randomly drawn from a range that varies from small to moderate data size. The minimum and maximum values of the range were derived based on the data sizes commonly used in the evaluation of task offloading solutions (see the studies in Table 3).

The simulation_parameters.properties file. The values of the simulation parameters that were the same for all simulation scenarios are shown in Appendix 5 Table 11. The length of one simulation run (episode) is one hour. As the default slot length is five seconds, there are 720 price updates (training steps) during one episode. It is also good to note that the values of link bandwidths, latencies and energy consumption rates are largely based on the default values of PureEdgeSim, which, in turn, are based on LEAF [37].

5. RESULTS

5.1. Hyperparameter Tuning

For each of the 24 simulation scenarios, hyperparameter tuning was done to find the best values for the actor and critic learning rates. Other hyperparameters were set to their default values (see Appendix 1 Table 7). One example of the result plots generated by EISim is shown in Figure 8. It shows the hyperparameter tuning results for the hybrid control topology with 100 servers and 2000 mobile users. The upper plot in Figure 8 shows the average cumulative return of the whole edge platform for each tested hyperparameter combination. The thick black line on top of a bar shows the 95% confidence interval of the average over five evaluation episodes. The lower plot shows the average cumulative return of each pricing agent for each hyperparameter combination. The shaded area shows the 95% confidence interval.

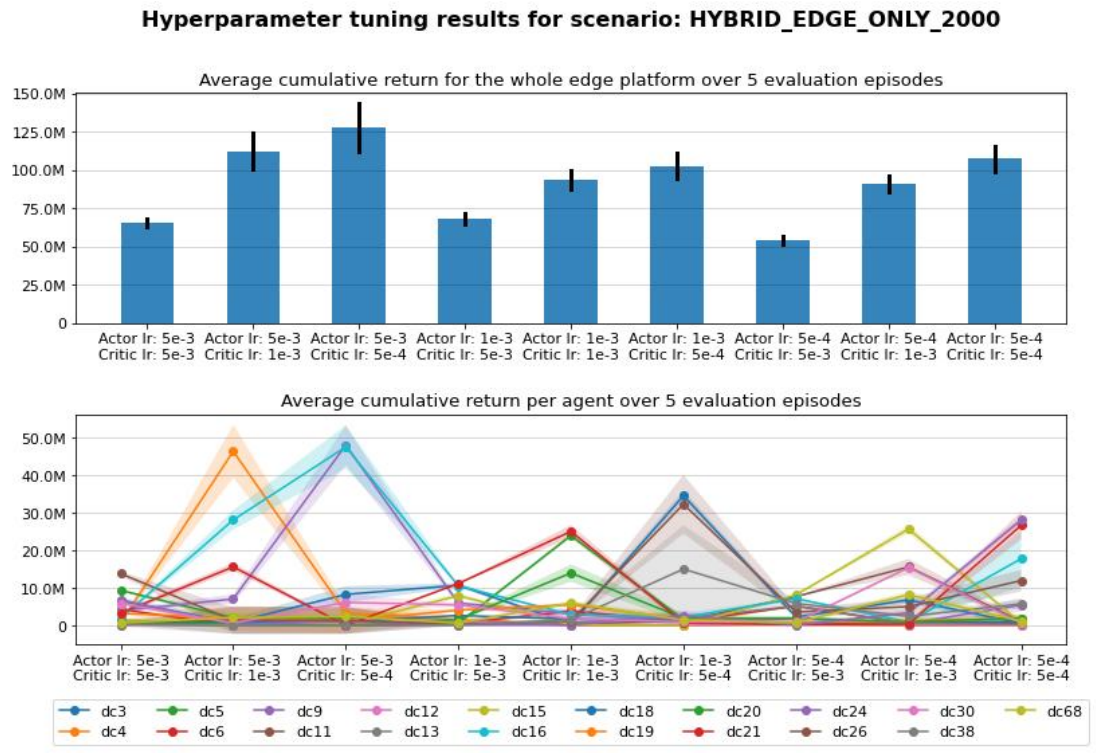


Figure 8. An example of a hyperparameter tuning result plot. Results are shown for the hybrid control topology scenario with 100 edge servers and 2000 mobile users.

In Figure 8, the higher impact of the critic learning rate on the performance is evident. The highest tested learning rate for the critic produces the lowest performance overall and per agent regardless of the value of the actor learning rate. Based on this, the lowest tested learning rate of 5e-4 was chosen for the critic. For the actor learning rate, the highest tested value in combination with the critic learning rate of 5e-4 produces the highest overall average. However, this result also has the highest degree of uncertainty, as the confidence interval is the widest. Further, when observing the performance of single agents, the high result is based on the good performance of only two agents that are way above the others in the environment. Hence, when taking into account both the overall performance and the performance per agent, the actor learning rate of 5e-4 was chosen.

The example above demonstrates well how the result plots generated by EISim can be used for selecting the best hyperparameter values. Similar analyses were also done for the other scenarios in the simulation study. In the end, the learning rate of 5e-4 was chosen for both actor and critic training in all scenarios.

5.2. Training

Every simulation scenario was trained for 100 episodes. To improve exploration, agents chose their action uniformly at random for 500 steps during the first four training episodes, after which only the first price decision was made at random. An example of the training progress plots generated by EISim is shown in Figure 9. It shows the training progress for the decentralized control topology with 20 servers and 2000 mobile users. The highest plot in Figure 9 shows the total cumulative return per training episode. The thick red line is the simple moving average of the total cumulative return calculated with a window of size 10. The middle plot shows the cumulative return of each agent per training episode. The lowest plot shows the average price of each agent per training episode.

It is evident in Figure 9 that the presence of multiple independent learners in the environment makes it difficult for a single agent to learn an optimal policy due to the inference caused by the other agents. This is a realistic result, as in addition to the reward itself being stochastic, the other agents bring a new source of stochasticity with their evolving policies and action exploration, making it difficult for a single agent to learn the effect of its actions. Further, the agents learn based on the experience saved in the experience replay, but the non-stationarity caused by the other agents means that the dynamics that generated the experience no longer represent the current dynamics for the learners. In other words, the experience can become obsolete very quickly.

Figure 9 shows an interesting strategy for agent ‘dc4‘. It learns very quickly to keep a price level that is way above the average prices of other agents. With this strategy it can occasionally gain a very high return in an episode, as seen in the middle plot. This agent is located at (215, 823) on the simulation map (see Figure 7a). It only has one very close competitor in its area, namely ‘dc14‘ located at (349, 667). The agent ‘dc14‘ keeps a much lower price level. The examination of its training logs shows that from time to time the exploration noise makes it set the price to zero, which heavily floods the server.

When taking into account the application profile of the edge devices and the default offloading decision-making logic with the randomly generated importance weights, it is clear that some devices may value low latency much more than low price. These devices may occasionally generate very long tasks, and in case they are in the area of ‘dc4‘ while ‘dc14‘ is flooded, they may accept its high price in exchange for low-latency execution. This is a perfect example of the destructive effect of the non-stationarity, as the reason why ‘dc4‘ thinks that its policy is feasible is a consequence of the exploration noise of ‘dc14‘. The examination of the evaluation logs shows that the agent ‘dc4‘ indeed keeps a very high price level constantly, while the agent ‘dc14‘ using its learned strategy without any exploration noise keeps a significantly lower

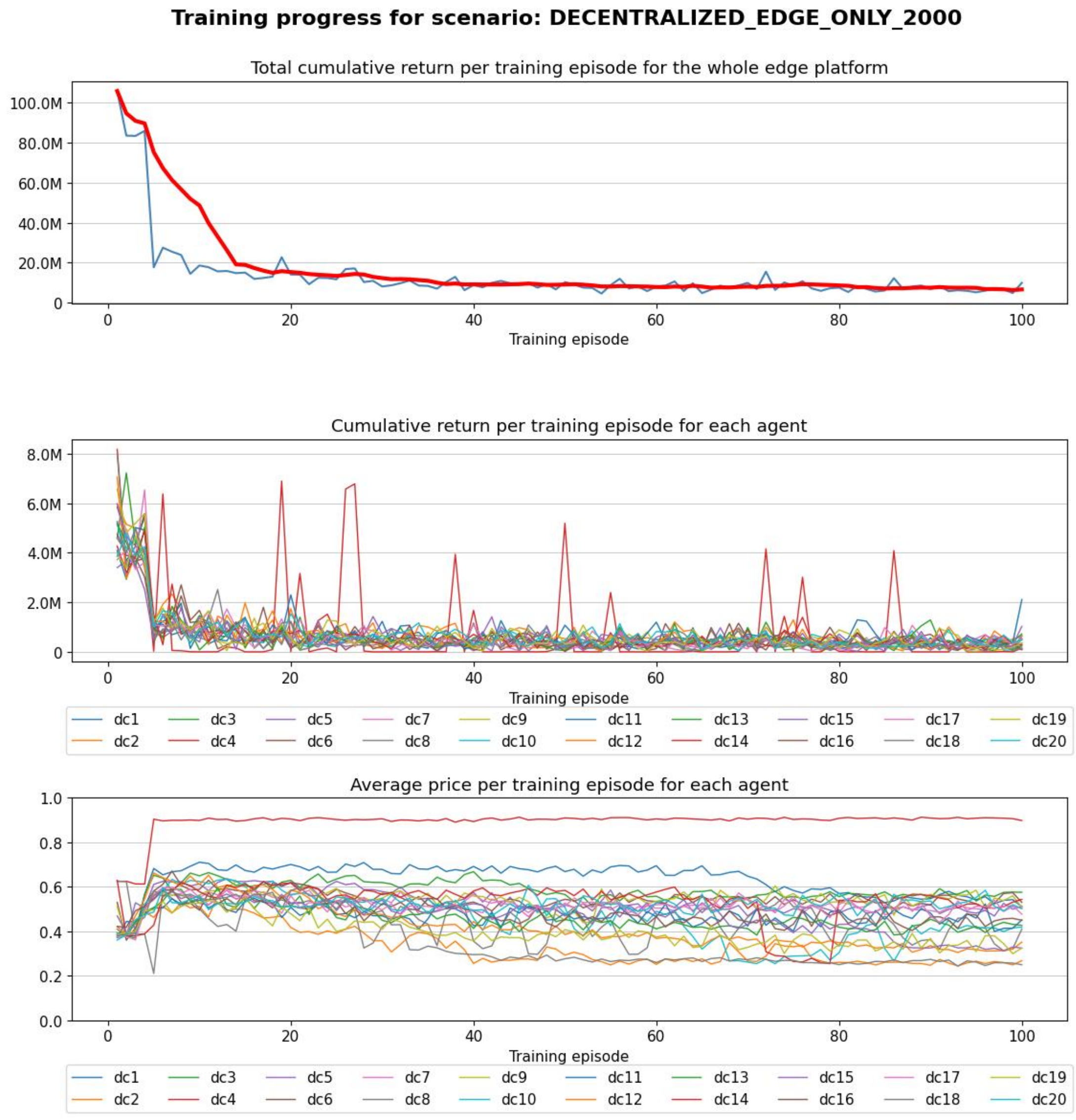


Figure 9. An example of a training plot. The training progress is shown for the decentralized control topology scenario with 20 edge servers and 2000 mobile users.

price level. Consequently, no edge device offloads to ‘dc4‘ during evaluation, as the potential offloaders nearby prefer the low price of ‘dc14‘.

The training progress plots of other multi-agent scenarios are similar to Figure 9. In Figure 10, an example of the training progress for the centralized control topology with 20 servers and 2000 mobile users is shown. Here it can be seen that the central orchestrator very quickly learns to price at a very high level. This is most likely due to similar reasons why the agent ‘dc4‘ learns to price high in Figure 9. In other words, some devices with long tasks accept the high price in exchange for the low-latency execution, which generates high profits for the orchestrator. Similar training progress also happens in other centralized scenarios. It is beneficial for the central orchestrator to keep the price as high as possible, as there are no other competitors in the environment. It is also good to note that the default reward function only takes into account the profit, meaning that the orchestrator is not incentivized to optimize the resource utilization on the platform.

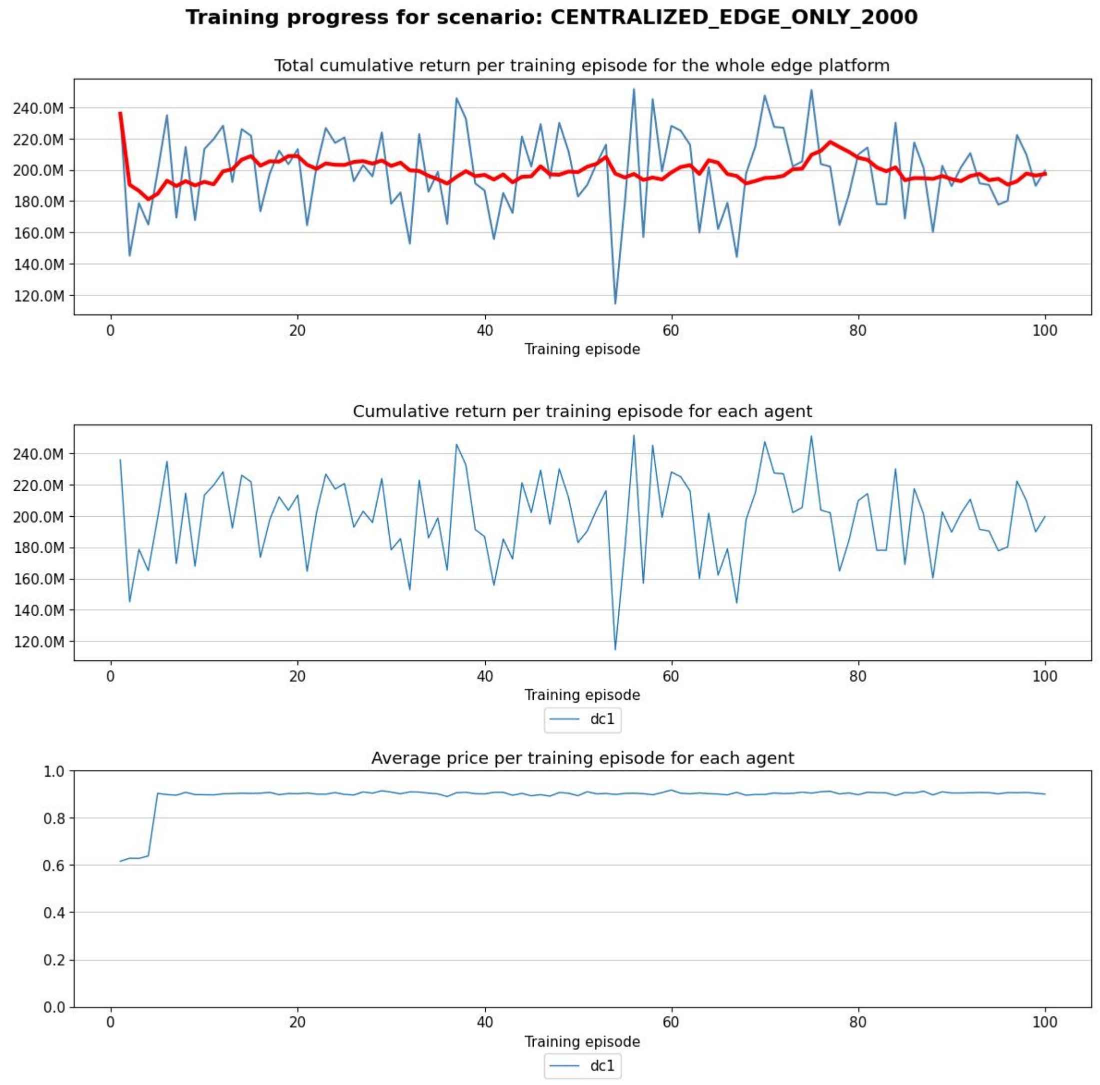


Figure 10. An example of a training plot. The training progress is shown for the centralized control topology scenario with 20 edge servers and 2000 mobile users.

The examples in this section show that the plots generated by EISim provide good insight into the training progress of the agents. Researchers can use the feedback provided by the plots to develop training methods for the agents. For example, based on the analysis of Figure 9, it is clear that the training of the agents in the multi-agent setting requires testing and developing interference avoidance techniques.

5.3. Evaluation

EISim readily plots multiple metrics that can be used to compare the performance across different control topologies after the agents in the environment have been trained. These metrics are listed in Table 4. Note that the raw logs of EISim include many more metrics, such as cloud related metrics. Additional metrics can be added to the plotted metrics by the user.

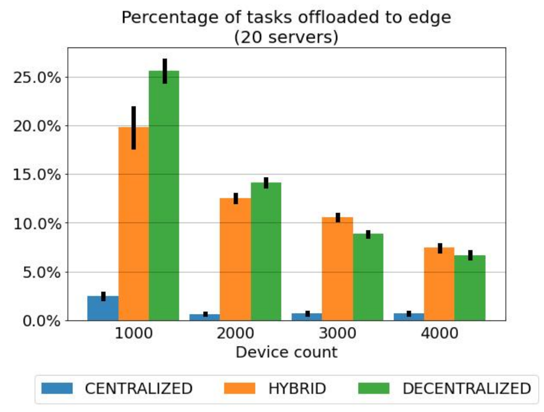
Table 4. Evaluation metrics plotted by EISim

Group	**Metrics**
Task processing	Tasks offloaded to edge (%)
	Tasks failed due to delay (%)
	Tasks successfully executed on edge servers (%)
	Tasks successfully executed on edge devices (%)
	Average execution delay per task (s)
	Average waiting time per task (s)
CPU utilization	Average CPU usage on edge servers (%)
	Average CPU usage on edge devices (%)
Energy consumption	Average energy consumption per edge server (Wh)
	Average energy consumption per edge device (Wh)
	Dead devices count
	Average remaining power per edge device (%)
Network	Network usage (s)
	LAN usage (s)
	MAN usage (s)
	Average network usage per offloaded task (s)
	Total network traffic (MB)
	Average bandwidth per task (Mbps)
Profit	Total cumulative return for the whole edge platform

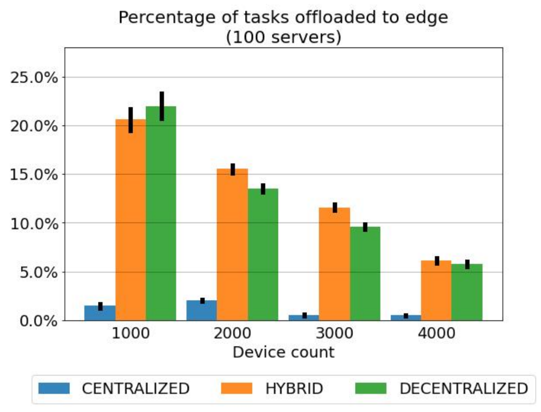
EISim generates grouped bar plots for each main scenario and each metric. The main scenario in the conducted simulation study refers to the two server options (20 high-capacity servers or 100 low-capacity servers). In one plot, each control topology has its own element, namely a bar group, and the x-axis values correspond to different edge device counts. The y-axis value for a control topology and edge device count combination is the average of the evaluation metric over the evaluation episodes. The confidence interval of this average is also plotted as a thick line on top of the bar. For the plots shown in this thesis, the confidence level is 95%.

Next, plots for some of the evaluation metrics in Table 4 are shown and analyzed. It is exemplified how the plots generated by EISim can be used to compare different control topologies across scenarios. Further, it is shown that EISim is able to output sensible and consistent results.

Figure 11 shows the percentage of tasks offloaded to the edge servers for the 20 high-capacity servers (Figure 11a) and the 100 low-capacity servers (Figure 11b). Here it can be seen that in all scenarios, the edge devices offload the least amount of tasks in the centralized control topology due to the high price set by the central orchestrator. In both hybrid and decentralized control topologies, increasing the number of edge devices decreases the percentage of offloaded tasks. This is most likely explained by the pricing strategies of the agents. The examination of the price logs shows that many agents in the environment learn to price a little higher when the edge device count is higher, which also reduces the willingness of the edge devices to offload. It is also interesting to note that for the majority of the scenarios, the percentage of tasks offloaded is around the same regardless of the number of the edge servers.



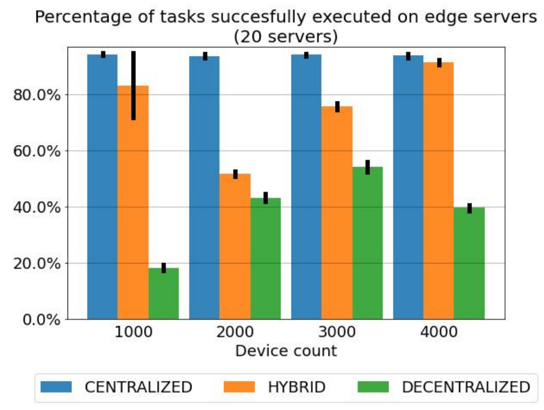
(a) 20 servers



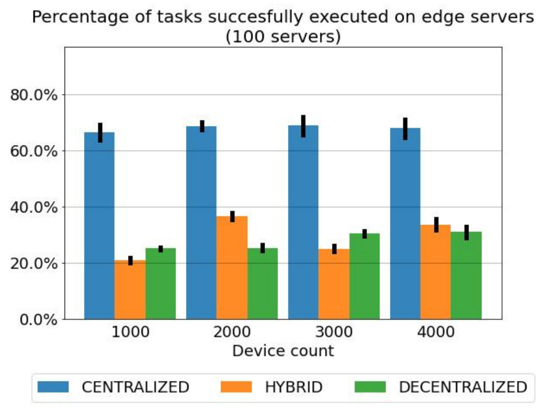
(b) 100 servers

Figure 11. Evaluation plots for the percentage of tasks offloaded to edge.

Figure 12 shows the percentage of the offloaded tasks successfully executed on the edge servers. A task was executed successfully if the latency constraint was satisfied. Here it is good to note that the used constraint is very strict relative to the average task length. In both scenarios with 20 servers (Figure 12a) and 100 servers (Figure 12b), the success rate in the centralized control topology is the highest due to the low number of tasks offloaded. Note that the success rate of the centralized topology is lower in the 100-server scenario, which is in line with the lower computational capacity of a single server in the 100-server case. When there are 20 high-capacity servers, hybrid control topology is able to achieve success rates closer to the success rate of the centralized control topology than when there are 100 low-capacity servers. The reduction in the success rate is most likely due to the lower capacity of a single server and the bigger cluster size in the 100-server case.



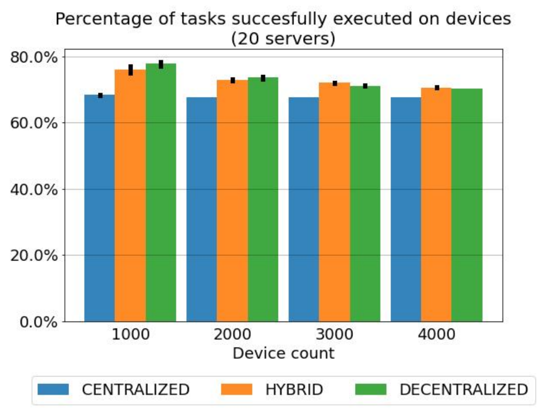
(a) 20 servers



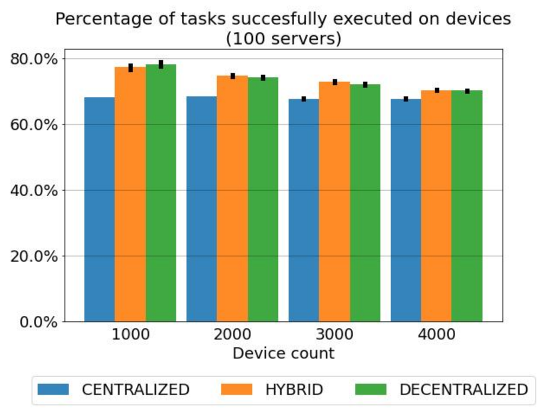
(b) 100 servers

Figure 12. Evaluation plots for the percentage of tasks successfully executed on edge servers.

Figure 13 shows the percentage of the local tasks successfully executed on the edge devices. Here the benefit of the offloading for the edge devices can be seen, as the success rate of the edge devices increases in the hybrid and decentralized control topologies, where more tasks are offloaded when compared to the centralized control topology. This is because devices are more likely to offload lengthy tasks, leaving them with the shorter ones. Further, it is good to note that the increase in the success rate for every scenario is in line with the percentage of offloaded tasks (see Figure 11).



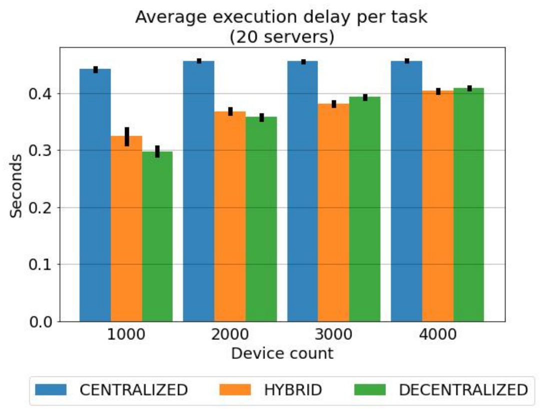
(a) 20 servers



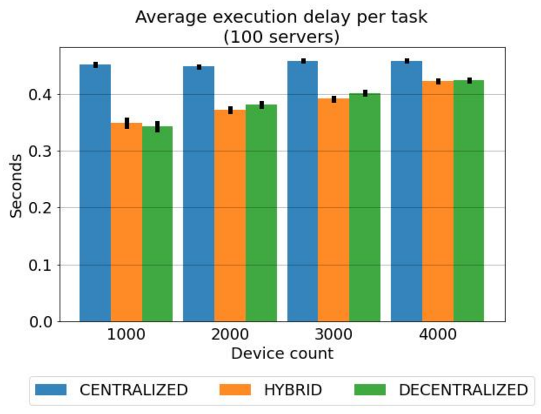
(b) 100 servers

Figure 13. Evaluation plots for the percentage of tasks successfully executed on edge devices.

Figure 14 shows the average execution delay per task. Note that this metric only takes into account the time a task spent on CPU. Further, it is calculated over all the tasks executed during the simulation, that is, it takes into account both the executions on the edge devices and edge servers. Here it can be seen that reduction in the average execution time is in line with the percentage of offloaded tasks (see Figure 11). In other words, the more tasks are offloaded, the lower the average time spent on CPU, as expected due to the higher computational capacity of the edge servers. Further, when the servers have a higher capacity (Figure 14a), the average execution delay is a little lower than when the servers have a lower capacity (Figure 14b), as expected.



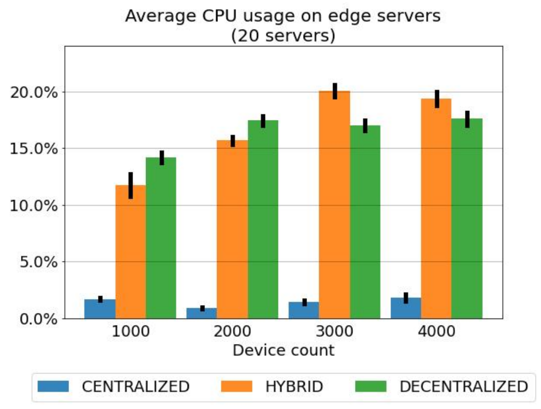
(a) 20 servers



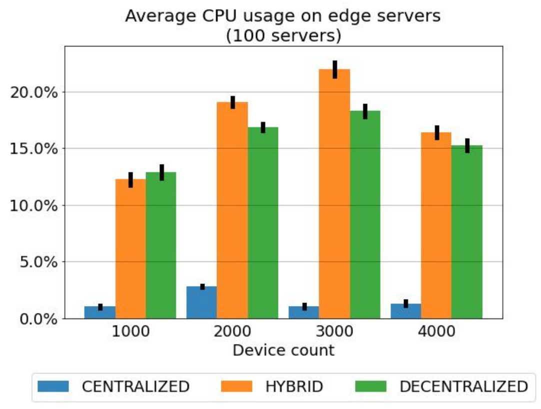
(b) 100 servers

Figure 14. Evaluation plots for the average execution delay (time on CPU) per task.

Figure 15 shows the average CPU utilization on edge servers. When comparing the average CPU usage level in different control topologies inside one edge server count and edge device count combination, it can be seen that it is in line with the percentage of offloaded tasks (see Figure 11). In other words, the more tasks are offloaded, the higher the CPU usage level on edge servers, as expected. When comparing the average CPU usage level across different device counts, it is important to note that even though proportionally the edge devices offload less tasks when the total number of edge devices is higher, the actual number of offloaded tasks can be higher. This is reflected in the higher average CPU usage level when the number of devices increases.



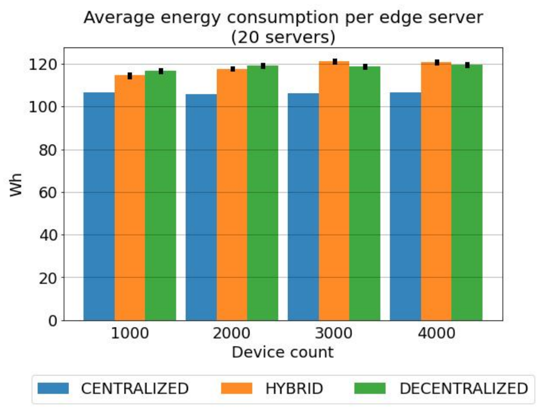
(a) 20 servers



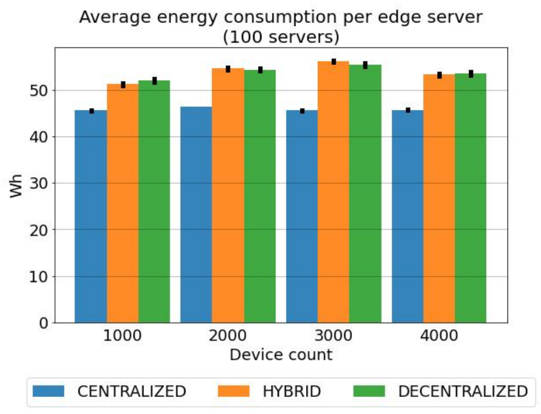
(b) 100 servers

Figure 15. Evaluation plots for the average CPU utilization on edge.

Figure 16 shows the average energy consumption per edge server. Here it can be seen for both edge server counts that the average energy consumption in the centralized control topology corresponds almost exactly to the idle consumption rate of an edge server, as expected due to the low number of offloaded tasks. The increased number of offloaded tasks is reflected in the average energy consumption for other control topologies.



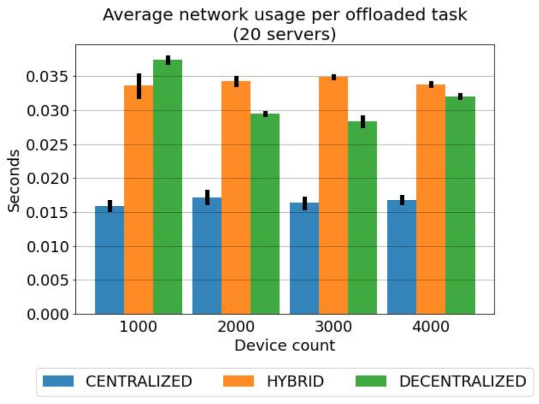
(a) 20 servers



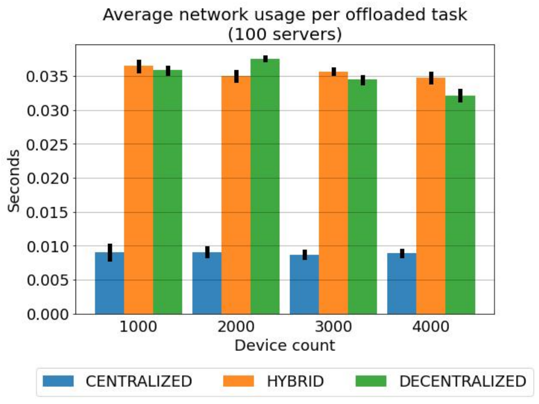
(b) 100 servers

Figure 16. Evaluation plots for the average energy consumption per edge server.

Figure 17 shows the average network usage per offloaded task. The effect of the network congestion on the average can be seen for both edge server counts. As the edge devices do not offload many tasks in the centralized control topology, the average network time per offloaded task is the lowest. Further, the average for the centralized control topology is lower in the 100-server case (Figure 17b) than in the 20-server case (Figure 17a). This is most likely because the deployment of edge servers is denser in the 100-server case, meaning that the central orchestrator is able to allocate an edge server that is geographically closer to the offloading device than in the 20-server case. In the hybrid and decentralized control topologies, having a denser deployment of edge servers does not cause a similar reduction in the average network time as in the centralized control topology. Particularly in the decentralized control topology, having more servers can increase the average network time. This may reflect the fact that in the 100-server case, a device has more options for offloading in its vicinity, each option with its own price, meaning that the device may choose a server that is not geographically closest to it due to a lower price in a server further away. This increases the distance a task must travel in the network and the congestion in the MAN links.



(a) 20 servers

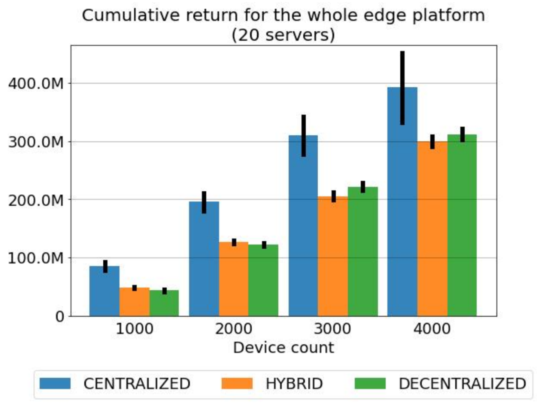


(b) 100 servers

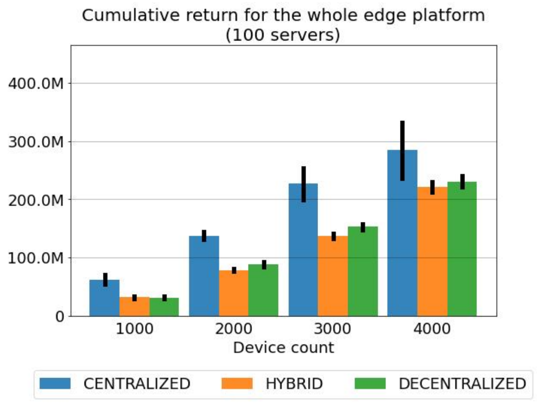
Figure 17. Evaluation plots for the average network usage per offloaded task.

Figure 18 shows the total cumulative return for the ESP. The centralized control topology always achieves the highest profit due to the high price set by the central orchestrator. However, the more there are devices in the environment, the more uncertain the profit of the ESP becomes, as reflected in the increased confidence interval for the average. It is also interesting to note that the more there are devices, the closer the profit from the hybrid and decentralized control topologies gets to the centralized one in terms of confidence. In the 100-server case (Figure 18b) with 4000 devices, the confidence intervals of all three control topologies overlap, indicating that the hybrid and decentralized control topologies are able to generate more stable profit close to the profit in the centralized control topology with a better resource utilization on the edge platform (see Figure 15). Additionally, the fact that the hybrid and decentralized control topologies consistently achieve around the same amount of profit inside one edge server count and edge device count combination is an interesting subject for further research.

Another interesting fact is that in the 20-server case (Figure 18a), the profit in every scenario is higher than in the 100-server case. This is most likely due to the lower capacity of a single server in the 100-server case. In other words, edge devices are not willing to pay as much as in the 20-server case, because the benefit of the lower task execution time is not as great as when a server has a higher capacity.



(a) 20 servers



(b) 100 servers

Figure 18. Evaluation plots for the total cumulative return.

5.4. Efficiency of EISim

The time complexity of EISim mainly depends on how many events are generated during a simulation run and how complex the event handling procedures are. The total number of events depends on multiple factors, such as the run-length of the simulation, the number of agents in the simulation environment, the total number of generated tasks, and the lengths of the price, network and mobility update intervals. The complexity of the orchestration logic and agent training also heavily affects the time complexity of EISim.

To provide insight into the time complexity of EISim, Table 5 reports the average time of a single simulation run for each control topology and edge server count combination. The average times are reported separately for the training and evaluation modes. For the training mode, the average time and its 95% confidence interval are calculated based on 100 simulation runs. For the evaluation mode, the average and the confidence interval are calculated based on 5 simulation runs. It is important to note in the reported values that one simulation run executed all four scenarios with different edge device counts (1000, 2000, 3000, 4000) in parallel. Based on the application profile used in the simulation study, the average number of tasks generated by each number of edge devices during a one-hour simulation run is 3.6 million, 7.2 million, 10.8 million, and 14.4 million, respectively.

For the comparison of the simulation times, it is important to note that two different machines were used to run the simulations. All the simulations for the hybrid control topology, as well as the simulations with 100 servers for the decentralized control topology were run on a Nokia AirFrame Rackmount server with two Intel Xeon E5-2680 v4 @ 2.40 GHz CPUs and 128 GB of RAM. More specifically, the simulations were run inside a Docker Ubuntu container with no resource constraints. All the simulations for the centralized control topology, as well as the simulations with 20 servers for the decentralized control topology were run on a desktop computer with Intel Core i7-7800X @ 3.50 GHz CPU and 128 GB of RAM. The native CPU backend was used in Deeplearning4j Java library to execute the DNN related computations.

Another important thing to note is that the machines were running simulation rounds for different scenarios at the same time, which also affects the simulation times.

Table 5. Average simulation times and 95% confidence intervals

	Training		**Evaluation**	
	20 servers	**100 servers**	**20 servers**	**100 servers**
Centralized	7 min 11 s ± 4 s	7 min 10 s ± 5 s	4 min 38 s ± 4 s	4 min 36 s ± 3 s
Hybrid	15 min 4 s ± 13 s	21 min 46 s ± 40 s	4 min 57 s ± 17 s	5 min 10 s ± 12 s
Decentralized	11 min 56 s ± 4 s	65 min 8 s ± 16 s	4 min 20 s ± 6 s	6 min 21 s ± 4 s

The training and evaluation rounds of the centralized control topology for both edge server counts were run simultaneously. Same for the training and evaluation of the hybrid control topology. On the other hand, the training and evaluation rounds of the decentralized control topology with 20 servers were mostly run standalone on the machine. Same for the decentralized control topology with 100 servers.

In Table 5, it can be seen that the training of the agents has a significant impact on the average run time. The more there are agents in the environment, the longer it takes to run one training round. On the other hand, having more agents does not significantly prolong the length of an evaluation round.

As for the memory complexity of EISim, the memory usage during one simulation run depends heavily on the number of agents in the simulation environment. Running one training round for the centralized control topology used around 15 to 20 GB of RAM. Note that one training round again includes running four scenarios with different edge device counts in parallel. On the other extreme, running one training round for the decentralized control topology with 100 agents used around 30 to 35 GB of RAM.

6. DISCUSSION

6.1. Significance and Limitations

The main significance of EISim stems from the fact that, to the best of the author's knowledge, it is the first openly available simulator that specifically supports the simulation of intelligent, DRL-based orchestration solutions and different orchestration control topologies. The default implementations for different control topologies provide a good and sensible starting point for research. A user can also easily modify these implementations due the high extensibility of EISim. Further, the user can simulate different pricing strategies, that is, how the cluster heads make the price decisions. The default implementation once again provides an excellent starting point that sets up the foundation for simulating DRL-based solutions. The user can easily modify the existing implementation or plug in a completely new, custom pricing implementation.

In its current form, EISim can provide answers to multiple significant research questions. It can be used to investigate what type of pricing strategies would be the most beneficial for different types of control topologies. It can also be used to provide insights into what type of pricing models would be suitable for the edge environment, which is an important, open research question (see Section 2.4.2). EISim can also be used to study the advantages and disadvantages of different types of control topologies in varying edge deployments and use cases. Further, it can be used to study emergent phenomena, such as whether the DRL-based pricing agents learn to collude, meaning that they learn to price above the competitive level without explicitly communicating with each other. Such *tacit collusion* between RL-based pricing agents has been a recent subject of concern in some studies [136, 137].

The simulation study done in this thesis shows that EISim can provide excellent feedback for choosing hyperparameters (see Section 5.1), for investigating the learning behavior of agents and developing training methods (see Section 5.2), and for comparing the performance of different control topologies (see Section 5.3). The comparison of the control topologies can also bring forth new, interesting research directions. For example, the hybrid and decentralized control topologies show similar performance for different metrics and different scenarios in Section 5.3. Whether such similarity emerges in different use cases and under different training methods is an interesting subject for future research.

The simulation study also shows that EISim can be used to simulate large-scale scenarios, as at most the environment had 100 learning agents and 4000 mobile devices. Existing works on offloading and resource pricing typically experiment in small environments. For example, Shi et al. experimented with only four edge servers and six mobile users in their DRL-based solution [101], while Chen et al. experimented with one edge server and from 20 to 40 users in their DRL-based solution [84]. Further, EISim enables evaluating the long-term performance of methods as a part of a large-scale, more realistic and more dynamic system, which is not often done for proposed offloading and pricing solutions (see Section 2.4.3 and the works in Table 3).

The additional tools in EISim also greatly facilitate research efforts. The environment setup tools provide a quick way to create varying environments. The bash scripts provide good templates for running simulations with EISim. Finally, the result plotting tools generate informative plots for result analysis, as exemplified in Chapter 5.

The main limitations of EISim concern the scope of the simulator and the assumptions made in the default implementations. The current scope of EISim is limited to simulating task offloading with independent tasks and resource pricing. In addition, there is only one pre-implemented DRL algorithm, and the agent model focuses on pricing agents.

The default implementations readily support scenarios where the task processing is done at the edge servers or the edge devices, and the tasks are vertically offloaded from the device level. The servers are also assumed to belong to the same ESP and have the same capacity. Further, the current implementation for the hybrid control topology supports only flat hierarchies with static clusters, which corresponds to the more traditional approach to orchestration control in edge orchestration literature (see Section 2.1.2).

Finally, the use of EISim requires that the user has a certain level of proficiency in both Java and Python programming, which can be perceived as a limitation.

6.2. Future Work

EISim has a multitude of potential future development directions. These are summarized in Table 6, which lists possible development directions for both parts of EISim, namely the simulator itself and the additional tools. Table 6 categorizes the development directions based on the development target. The target depicts a feature that is improved as a result of the development.

For improving the scope of the simulator, adding support for other orchestration functions besides offloading and pricing is important. There also needs to be simulation and result analysis support for other types of agents besides pricing agents, such as for RL agents that decide about offloading. To develop the hybrid control topology more towards the vision in [5], there must be a better, ready support for multi-level, loosely coupled hierarchies and dynamic clusters. Further, there must be support for modelling multiple service providers in the environment.

Expanding the scope and set of the default implementations is also important for having a better, ready support for simulating varying use cases with EISim. For example, there should be default implementations, where the cloud is also taken into account as a possible offloading destination. There could also be more sophisticated default implementations, where long-term optimization is also considered in the edge device side decision making. Besides RL, this could mean, for example, a Lyapunov optimization based method, where a device makes sure that the average money spent over time stays below some predetermined value. Finally, expanding the set of pre-implemented DRL algorithms is also important for facilitating research efforts with EISim.

For improving the simulation model, it needs to be investigated whether it is beneficial to have a more detailed simulation model. For example, currently the communication model of LAN is simplified. The distance of an edge device from an AP does not affect the link quality, nor does the number of users simultaneously connected to the same AP. The rates of the downlink and uplink are also set to be

Table 6. Possible future development directions for EISim

	Target	**Development Directions**
Simulator	Scope	Support for a wider set of orchestration functions Support for a wider variety of agents Support for multi-level hierarchies and dynamic clusters Support for multiple service providers Default implementations with wider focus More ready implementations for different DRL algorithms
	Modelling	Investigating whether it is worth to have a more detailed simulation model, e.g., for networking Modelling virtualized resources More dynamic model, e.g., simulating the time of the day
	Efficiency	Parallelizing agent model updates
	Validation	Building real testbeds for data collection Wider simulations over a variety of use cases
Additional tools	Scope	Implementing more methods for MAN creation and clustering Designing new result plots
	Ease of use	Creating a GUI

the same. The extent to which the realism of the LAN communication model should be increased needs to be examined. Further, to support other orchestration functions such as migration, there needs to be a more detailed model for virtualized resources, such as VMs and containers. However, when it comes to complicating the simulation model, it is important to note that generally increasing the model complexity beyond a certain point does not significantly alter the simulation results, but it does increase the time complexity of the simulation [35]. Finally, improving the simulation model into a more dynamic one, where, for example, the time of the day could by simulated by modelling changes in the task arrival rates and the dynamic arrivals and departures of edge devices, is another interesting development direction.

The training of agents has a significant impact on the average simulation time, as seen in Section 5.4 and Table 5. Currently, the updating of the models is tied to an update event, which is handled for each agent sequentially. Parallelizing the model updates of the agents is an important development direction for improving the efficiency of EISim.

Further validation of EISim is a necessity for increasing the confidence on the simulator. Hence, future work should also look into building real testbeds for collecting comparable data that could be used to validate EISim. Further validation should also be done by testing EISim over a wider variety of use cases.

For improving the scope of the additional tools, future work could look into implementing more methods for MAN creation and clustering. Currently there is one pre-implemented way for AP placement, topology creation, edge server placement, and edge server clustering. These already provide a way for creating varying environments due to the randomness in the methods, but to further facilitate the research in different use cases, there should be a wider set of possible methods that could be used. For example, the current method for the AP placement locates the APs evenly on an area. Future work could look into methods that could create environments where the density of the APs on the area varies.

For the result plotting tools, new informative plots could be designed. For example, it could be investigated whether there is an efficient way to visualize the distributions of the prices, profits and state variables for the pricing agents over simulation runs. New types of plots for convergence and training stability analysis should also be designed. This includes investigating efficient ways to visualize the actor and critic losses and other related metrics for multiple agents.

Finally, to improve the ease of use for the additional tools, a Graphical User Interface (GUI) could be designed and implemented for the environment setup and result plotting tools.

7. SUMMARY

A simulation platform called Edge Intelligence Simulator (EISim) was presented in this thesis. First, definitions for a set of key terms used in this thesis were given. Next, an encompassing overview of the state of the art in edge and fog simulation, orchestration, offloading, and resource pricing was given, providing a proper basis for the work on EISim. Then, a set of methods and tools that form the foundation of the current EISim implementation was presented. This was followed by a detailed description of the architecture, default implementations and use of EISim along with its additional tools for environment setup, agent training, and result plotting.

Next, the thesis reported the details of the simulation case study that was used to validate EISim and demonstrate its capabilities with regard to training agents and evaluating orchestration solutions against control topologies. The study focused on a large-scale MEC scenario, where mobile users move in a city area and generate independent tasks. All three control topologies (centralized, hybrid and decentralized) with their associated default pricing and offloading decision-making implementations were simulated on this area.

The results of the simulation study verified the end-to-end performance of EISim and showed its capability to produce sensible and consistent results. It was exemplified how the plots generated by EISim can aid in choosing the best hyperparameters, examining the training progress of agents, and comparing the performance of different control topologies.

EISim is developed towards supporting the easier testing and evaluation of intelligent, DRL-based orchestration methods against different orchestration control topologies. In its current form, EISim supports simulating scenarios related to task offloading and resource pricing. The thesis gave a comprehensive overview of the potential future development directions of EISim that it should follow in order to fully live up to its name. The most significant ones include adding support for other orchestration functions, multiple service providers and multi-level hierarchies, expanding the scope of the default implementations, and expanding the set of the pre-implemented DRL algorithms.

EISim is, to the best of the author's knowledge, the first openly available simulator that has a specific support for simulating intelligent orchestration solutions and orchestration control topologies. EISim makes it possible to evaluate the long-term performance of different solutions as a part of a large-scale, more realistic and more dynamic system, and the default implementations provide a solid foundation for further research. Even though the current scope of EISim is limited, it can provide answers to multiple important, open research questions, such as what type of pricing strategies would be the most efficient for different types of control topologies or how different types of control topologies perform in varying edge deployments and use cases.

8. REFERENCES

[1] Gilchrist A. (2016) Industry 4.0. Apress Berkeley, CA. DOI: http://dx.doi.org/10.1007/978-1-4842-2047-4.

[2] Qadri Y.A., Nauman A., Zikria Y.B., Vasilakos A.V. & Kim S.W. (2020) The future of healthcare internet of things: A survey of emerging technologies. IEEE Communications Surveys & Tutorials 22(2), pp. 1121–1167. DOI: http://dx.doi.org/10.1109/COMST.2020.2973314.

[3] Levinson J., Askeland J., Becker J., Dolson J., Held D., Kammel S., Kolter J.Z., Langer D., Pink O., Pratt V., Sokolsky M., Stanek G., Stavens D., Teichman A., Werling M. & Thrun S. (2011) Towards fully autonomous driving: Systems and algorithms. In: 2011 IEEE Intelligent Vehicles Symposium (IV), Baden-Baden, Germany, pp. 163–168. DOI: http://dx.doi.org/10.1109/IVS.2011.5940562.

[4] Ren J., Zhang D., He S., Zhang Y. & Li T. (2020) A survey on end-edge-cloud orchestrated network computing paradigms: Transparent computing, mobile edge computing, fog computing, and cloudlet. ACM Computing Surveys 52(6), pp. 1–36. DOI: http://dx.doi.org/10.1145/3362031.

[5] Kokkonen H., Lovén L., Motlagh N.H., Partala J., González-Gil A., Sola E., Angulo I., Liyanage M., Leppänen T., Nguyen T., Pujol V.C., Kostakos P., Bennis M., Tarkoma S., Dustdar S., Pirttikangas S. & Riekki J. (2022), Autonomy and intelligence in the computing continuum: Challenges, enablers, and future directions for orchestration. arXiv:2205.01423. DOI: http://dx.doi.org/10.48550/ARXIV.2205.01423.

[6] Lovén L., Leppänen T., Peltonen E., Partala J., Harjula E., Porambage P., Ylianttila M. & Riekki J. (2019) EdgeAI: A vision for distributed, edge-native artificial intelligence in future 6G networks. In: The 1st 6G Wireless Summit, Levi, Finland, pp. 1–2.

[7] Deng S., Zhao H., Fang W., Yin J., Dustdar S. & Zomaya A.Y. (2020) Edge intelligence: The confluence of edge computing and artificial intelligence. IEEE Internet of Things Journal 7(8), pp. 7457–7469. DOI: http://dx.doi.org/10.1109/JIOT.2020.2984887.

[8] Xu D., Li T., Li Y., Su X., Tarkoma S., Jiang T., Crowcroft J. & Hui P. (2021) Edge intelligence: Empowering intelligence to the edge of network. Proceedings of the IEEE 109(11), pp. 1778–1837. DOI: http://dx.doi.org/10.1109/JPROC.2021.3119950.

[9] Park J., Samarakoon S., Bennis M. & Debbah M. (2019) Wireless network intelligence at the edge. Proceedings of the IEEE 107(11), pp. 2204–2239. DOI: http://dx.doi.org/10.1109/JPROC.2019.2941458.

[10] Park J., Samarakoon S., Elgabli A., Kim J., Bennis M., Kim S.L. & Debbah M. (2021) Communication-efficient and distributed learning over

wireless networks: Principles and applications. Proceedings of the IEEE 109(5), pp. 796–819. DOI: `http://dx.doi.org/10.1109/JPROC.2021.3055679`.

[11] Zhou Z., Chen X., Li E., Zeng L., Luo K. & Zhang J. (2019) Edge intelligence: Paving the last mile of artificial intelligence with edge computing. Proceedings of the IEEE 107(8), pp. 1738–1762. DOI: `http://dx.doi.org/10.1109/JPROC.2019.2918951`.

[12] Taleb T., Samdanis K., Mada B., Flinck H., Dutta S. & Sabella D. (2017) On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. IEEE Communications Surveys & Tutorials 19(3), pp. 1657–1681. DOI: `http://dx.doi.org/10.1109/COMST.2017.2705720`.

[13] Saraiva de Sousa N.F., Lachos Perez D.A., Rosa R.V., Santos M.A. & Esteve Rothenberg C. (2019) Network service orchestration: A survey. Computer Communications 142-143, pp. 69–94. DOI: `http://dx.doi.org/10.1016/j.comcom.2019.04.008`.

[14] Hong C.H. & Varghese B. (2019) Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. ACM Computing Surveys 52(5). DOI: `http://dx.doi.org/10.1145/3326066`.

[15] Zhong Z., Xu M., Rodriguez M.A., Xu C. & Buyya R. (2022) Machine learning-based orchestration of containers: A taxonomy and future directions. ACM Computing Surveys 54(10s). DOI: `http://dx.doi.org/10.1145/3510415`.

[16] Riekki J. & Mämmelä A. (2021) Research and education towards smart and sustainable world. IEEE Access 9, pp. 53156–53177. DOI: `http://dx.doi.org/10.1109/ACCESS.2021.3069902`.

[17] Mämmelä A. & Riekki J. (2021) Subsidiarity and weak coupling in wireless networks. In: 2021 Joint European Conference on Networks and Communications 6G Summit (EuCNC/6G Summit), Porto, Portugal, pp. 598–603. DOI: `http://dx.doi.org/10.1109/EuCNC/6GSummit51104.2021.9482591`.

[18] Mechalikh C., Taktak H. & Moussa F. (2021) PureEdgeSim: A simulation framework for performance evaluation of cloud, edge and mist computing environments. Computer Science and Information Systems 18(1), pp. 43–66. DOI: `http://dx.doi.org/10.2298/CSIS200301042M`.

[19] Lillicrap T.P., Hunt J.J., Pritzel A., Heess N., Erez T., Tassa Y., Silver D. & Wierstra D. (2019), Continuous control with deep reinforcement learning. arXiv:1509.02971. DOI: `https://doi.org/10.48550/arXiv.1509.02971`.

[20] Russell S. & Norvig P. (2016) Artificial intelligence: A modern approach. Prentice Hall series in artificial intelligence, Pearson, Boston, 3rd global ed.

[21] Bonomi F., Milito R., Zhu J. & Addepalli S. (2012) Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing (MCC '12), Association for Computing Machinery, New York, NY, USA, pp. 13–16. DOI: `http://dx.doi.org/10.1145/2342509.2342513`.

[22] Atlam H., Walters R. & Wills G. (2018) Fog computing and the internet of things: A review. Big Data and Cognitive Computing 2(2). DOI: `http://dx.doi.org/10.3390/bdcc2020010`.

[23] Khan W.Z., Ahmed E., Hakak S., Yaqoob I. & Ahmed A. (2019) Edge computing: A survey. Future Generation Computer Systems 97, pp. 219–235. DOI: `http://dx.doi.org/10.1016/j.future.2019.02.050`.

[24] Wang X., Han Y., Leung V.C.M., Niyato D., Yan X. & Chen X. (2020) Convergence of edge computing and deep learning: A comprehensive survey. IEEE Communications Surveys & Tutorials 22(2), pp. 869–904. DOI: `http://dx.doi.org/10.1109/COMST.2020.2970550`.

[25] Shi W., Cao J., Zhang Q., Li Y. & Xu L. (2016) Edge computing: Vision and challenges. IEEE Internet of Things Journal 3(5), pp. 637–646. DOI: `http://dx.doi.org/10.1109/JIOT.2016.2579198`.

[26] Naha R.K., Garg S., Georgakopoulos D., Jayaraman P.P., Gao L., Xiang Y. & Ranjan R. (2018) Fog computing: Survey of trends, architectures, requirements, and research directions. IEEE Access 6, pp. 47980–48009. DOI: `http://dx.doi.org/10.1109/ACCESS.2018.2866491`.

[27] Balouek-Thomert D., Renart E.G., Zamani A.R., Simonet A. & Parashar M. (2019) Towards a computing continuum: Enabling edge-to-cloud integration for data-driven workflows. The International Journal of High Performance Computing Applications 33(6), pp. 1159–1174. DOI: `http://dx.doi.org/10.1177/1094342019877383`.

[28] Dustdar S., Casamajor Pujol V. & Donta P.K. (2022) On distributed computing continuum systems. IEEE Transactions on Knowledge and Data Engineering 35(4). DOI: `http://dx.doi.org/10.1109/TKDE.2022.3142856`.

[29] Huang D. & Wu H. (2018) Chapter 1 - Mobile cloud computing taxonomy. In: D. Huang & H. Wu (eds.) Mobile Cloud Computing, Morgan Kaufmann, pp. 5–29. DOI: `http://dx.doi.org/10.1016/B978-0-12-809641-3.00002-8`.

[30] Banks J. (2000) Introduction to simulation. In: 2000 Winter Simulation Conference Proceedings (Cat. No.00CH37165), Orlando, FL, USA, pp. 9–16. DOI: `http://dx.doi.org/10.1109/WSC.2000.899690`.

[31] Mämmelä A. & Riekki J. (2022) New network architectures will be weakly coupled. IEEE Future Networks Tech Focus Issue 14.

[32] Costa B., Bachiega J., de Carvalho L.R. & Araujo A.P.F. (2022) Orchestration in fog computing: A comprehensive survey. ACM Computing Surveys 55(2), pp. 1–34. DOI: `http://dx.doi.org/10.1145/3486221`.

[33] Masip X., Marín E., Garcia J. & Sànchez S. (2020) Collaborative mechanism for hybrid fog-cloud scenarios. In: Y. Yang, J. Huang, T. Zhang & J. Weinman (eds.) Fog and Fogonomics, John Wiley & Sons, pp. 7–60. DOI: `http://dx.doi.org/10.1002/9781119501121.ch2`.

[34] Banks J., Carson J.S., Nelson B.L. & Nicol D.M. (2001) Discrete event system simulation. Prentice Hall, 3rd ed.

[35] Robinson S. (2004) Simulation: The practice of model development and use. John Wiley.

[36] Svorobej S., Takako Endo P., Bendechache M., Filelis-Papadopoulos C., Giannoutakis K.M., Gravvanis G.A., Tzovaras D., Byrne J. & Lynn T. (2019) Simulating fog and edge computing scenarios: An overview and research challenges. Future Internet 11(3). DOI: `http://dx.doi.org/10.3390/fi11030055`.

[37] Wiesner P. & Thamsen L. (2021), LEAF: Simulating large energy-aware fog computing environments. arXiv:2103.01170. DOI: `http://dx.doi.org/10.48550/ARXIV.2103.01170`.

[38] Ahvar E., Orgerie A.C. & Lebre A. (2022) Estimating energy consumption of cloud, fog, and edge computing infrastructures. IEEE Transactions on Sustainable Computing 7(2), pp. 277–288. DOI: `http://dx.doi.org/10.1109/TSUSC.2019.2905900`.

[39] Jalali F., Hinton K., Ayre R., Alpcan T. & Tucker R.S. (2016) Fog computing may help to save energy in cloud computing. IEEE Journal on Selected Areas in Communications 34(5), pp. 1728–1739. DOI: `http://dx.doi.org/10.1109/JSAC.2016.2545559`.

[40] Meng X., Wang W. & Zhang Z. (2017) Delay-constrained hybrid computation offloading with cloud and fog computing. IEEE Access 5, pp. 21355–21367. DOI: `http://dx.doi.org/10.1109/ACCESS.2017.2748140`.

[41] Yuan Y., Yi C., Chen B., Shi Y. & Cai J. (2022) A computation offloading game for jointly managing local pre-processing time-length and priority selection in edge computing. IEEE Transactions on Vehicular Technology 71(9), pp. 9868–9883. DOI: `http://dx.doi.org/10.1109/TVT.2022.3177432`.

[42] Gill M. & Singh D. (2021) A comprehensive study of simulation frameworks and research directions in fog computing. Computer Science Review 40. DOI: `http://dx.doi.org/10.1016/j.cosrev.2021.100391`.

[43] Margariti S., Dimakopoulos V. & Tsoumanis G. (2020) Modeling and simulation tools for fog computing-a comprehensive survey from a cost

perspective. Future Internet 12(5). DOI: http://dx.doi.org/10.3390/fi12050089.

[44] Kunde C. & Mann Z.A. (2020) Comparison of simulators for fog computing. In: Proceedings of the 35th Annual ACM Symposium on Applied Computing, Association for Computing Machinery, New York, NY, USA, pp. 1792 – 1795. DOI: http://dx.doi.org/10.1145/3341105.3375771.

[45] Aral A. & Maio V. (2020) Simulators and emulators for edge computing. In: J. Taheri & S. Deng (eds.) Edge Computing: Models, technologies and applications, pp. 291–311. DOI: http://dx.doi.org/10.1049/PBPC033E_ch14.

[46] Gupta H., Dastjerdi A.V., Ghosh S.K. & Buyya R. (2017) iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Software: Practice and Experience 47(9), pp. 1275–1296. DOI: https://doi.org/10.1002/spe.2509.

[47] Calheiros R.N., Ranjan R., Beloglazov A., De Rose C.A.F. & Buyya R. (2011) CloudSim: A toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience 41(1), pp. 23–50. DOI: http://dx.doi.org/10.1002/spe.995.

[48] Lopes M.M., Higashino W.A., Capretz M.A. & Bittencourt L.F. (2017) MyiFogSim: A simulator for virtual machine migration in fog computing. In: Companion Proceedings of The 10th International Conference on Utility and Cloud Computing (UCC '17 Companion), Austin, Texas, USA, pp. 47–52. DOI: http://dx.doi.org/10.1145/3147234.3148101.

[49] Puliafito C., Gonçalves D.M., Lopes M.M., Martins L.L., Madeira E., Mingozzi E., Rana O. & Bittencourt L.F. (2020) MobFogSim: Simulation of mobility and migration for fog computing. Simulation Modelling Practice and Theory 101. DOI: http://dx.doi.org/10.1016/j.simpat.2019.102062.

[50] Mahmud R., Pallewatta S., Goudarzi M. & Buyya R. (2022) iFogSim2: An extended iFogSim simulator for mobility, clustering, and microservice management in edge and fog computing environments. Journal of Systems and Software 190. DOI: http://dx.doi.org/10.1016/j.jss.2022.111351.

[51] Lera I., Guerrero C. & Juiz C. (2019) YAFS: A simulator for iot scenarios in fog computing. IEEE Access 7, pp. 91745–91758. DOI: http://dx.doi.org/10.1109/ACCESS.2019.2927895.

[52] Center for Applied Internet Data Analysis (2023), Macroscopic internet topology data kit (ITDK). URL: https://www.caida.org/catalog/datasets/internet-topology-data-kit/, Accessed 27 September 2023.

[53] Medina A., Lakhina A., Matta I. & Byers J. (2001) BRITE: An approach to universal topology generation. In: MASCOTS 2001, Proceedings Ninth International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, Cincinnati, OH, USA, pp. 346–353. DOI: http://dx.doi.org/10.1109/MASCOT.2001.948886.

[54] Sonmez C., Ozgovde A. & Ersoy C. (2018) EdgeCloudSim: An environment for performance evaluation of edge computing systems. Transactions on Emerging Telecommunications Technologies 29(11). DOI: http://dx.doi.org/10.1002/ett.3493.

[55] Shaik S., Hall J., Johnson C., Wang Q., Sharp R. & Baskiyar S. (2022) PFogSim: A simulator for evaluation of mobile and hierarchical fog computing. Sustainable Computing: Informatics and Systems 35. DOI: http://dx.doi.org/10.1016/j.suscom.2022.100736.

[56] Fernández-Cerero D., Fernández-Montes A., Ortega F.J., Jakóbik A. & Widlak A. (2020) Sphere: Simulator of edge infrastructures for the optimization of performance and resources energy consumption. Simulation Modelling Practice and Theory 101. DOI: http://dx.doi.org/10.1016/j.simpat.2019.101966.

[57] Qayyum T., Malik A.W., Khan Khattak M.A., Khalid O. & Khan S.U. (2018) FogNetSim++: A toolkit for modeling and simulation of distributed fog environment. IEEE Access 6, pp. 63570–63583. DOI: http://dx.doi.org/10.1109/ACCESS.2018.2877696.

[58] Varga A. & Hornig R. (2008) An overview of the OMNeT++ simulation environment. In: Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools '08), ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, pp. 1–10.

[59] Guerzoni R., Vaishnavi I., Perez Caparros D., Galis A., Tusa F., Monti P., Sganbelluri A., Biczók G., Sonkoly B., Toka L., Ramos A., Melián J., Dugeon O., Cugini F., Martini B., Iovanna P., Giuliani G., Figueiredo R., Contreras-Murillo L.M., Bernardos C.J., Santana C. & Szabo R. (2017) Analysis of end-to-end multi-domain management and orchestration frameworks for software defined infrastructures: An architectural survey. Transactions on Emerging Telecommunications Technologies 28(4). DOI: http://dx.doi.org/10.1002/ett.3103.

[60] Mach P. & Becvar Z. (2017) Mobile edge computing: A survey on architecture and computation offloading. IEEE Communications Surveys & Tutorials 19(3), pp. 1628–1656. DOI: http://dx.doi.org/10.1109/COMST.2017.2682318.

[61] Jiang C., Cheng X., Gao H., Zhou X. & Wan J. (2019) Toward computation offloading in edge computing: A survey. IEEE Access 7, pp. 131543–131558. DOI: http://dx.doi.org/10.1109/ACCESS.2019.2938660.

[62] Saeik F., Avgeris M., Spatharakis D., Santi N., Dechouniotis D., Violos J., Leivadeas A., Athanasopoulos N., Mitton N. & Papavassiliou S. (2021) Task offloading in edge and cloud computing: A survey on mathematical, artificial intelligence and control theory solutions. Computer Networks 195. DOI: http://dx.doi.org/10.1016/j.comnet.2021.108177.

[63] Huang L., Bi S. & Zhang Y.J.A. (2020) Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks. IEEE Transactions on Mobile Computing 19(11), pp. 2581–2593. DOI: http://dx.doi.org/10.1109/TMC.2019.2928811.

[64] Zhao R., Wang X., Xia J. & Fan L. (2020) Deep reinforcement learning based mobile edge computing for intelligent internet of things. Physical Communication 43. DOI: http://dx.doi.org/10.1016/j.phycom.2020.101184.

[65] Han D., Chen W. & Fang Y. (2020) Joint channel and queue aware scheduling for latency sensitive mobile edge computing with power constraints. IEEE Transactions on Wireless Communications 19(6), pp. 3938–3951. DOI: http://dx.doi.org/10.1109/TWC.2020.2979136.

[66] Liu C.F., Bennis M., Debbah M. & Poor H.V. (2019) Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing. IEEE Transactions on Communications 67(6), pp. 4132–4150. DOI: http://dx.doi.org/10.1109/TCOMM.2019.2898573.

[67] Pu L., Chen X., Xu J. & Fu X. (2016) D2d fogging: An energy-efficient and incentive-aware task offloading framework via network-assisted d2d collaboration. IEEE Journal on Selected Areas in Communications 34(12), pp. 3887–3901. DOI: http://dx.doi.org/10.1109/JSAC.2016.2624118.

[68] Yu S., Wang X. & Langar R. (2017) Computation offloading for mobile edge computing: A deep learning approach. In: 2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC), Montreal, QC, Canada, pp. 1–6. DOI: http://dx.doi.org/10.1109/PIMRC.2017.8292514.

[69] Mao Y., Zhang J. & Letaief K.B. (2016) Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE Journal on Selected Areas in Communications 34(12), pp. 3590–3605. DOI: http://dx.doi.org/10.1109/JSAC.2016.2611964.

[70] Yang B., Cao X., Bassey J., Li X., Kroecker T. & Qian L. (2019) Computation offloading in multi-access edge computing networks: A multi-task learning approach. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, pp. 1–6. DOI: http://dx.doi.org/10.1109/ICC.2019.8761212.

[71] Guo H. & Liu J. (2018) Collaborative computation offloading for multiaccess edge computing over fiber–wireless networks. IEEE Transactions on Vehicular Technology 67(5), pp. 4514–4526. DOI: http://dx.doi.org/10.1109/TVT.2018.2790421.

[72] Wang Z., Zhao Z., Min G., Huang X., Ni Q. & Wang R. (2018) User mobility aware task assignment for mobile edge computing. Future Generation Computer Systems 85, pp. 1–8. DOI: http://dx.doi.org/10.1016/j.future.2018.02.014.

[73] Hussein M.K. & Mousa M.H. (2020) Efficient task offloading for iot-based applications in fog computing using ant colony optimization. IEEE Access 8, pp. 37191–37201. DOI: http://dx.doi.org/10.1109/ACCESS.2020.2975741.

[74] Bahreini T., Badri H. & Grosu D. (2022) Mechanisms for resource allocation and pricing in mobile edge computing systems. IEEE Transactions on Parallel and Distributed Systems 33(3), pp. 667–682. DOI: http://dx.doi.org/10.1109/TPDS.2021.3099731.

[75] Huang X., Zhang B. & Li C. (2022) Incentive mechanisms for mobile edge computing: Present and future directions. IEEE Network 36(6), pp. 199–205. DOI: http://dx.doi.org/10.1109/MNET.107.2100652.

[76] Su Y., Fan W., Liu Y. & Wu F. (2021) Game-based distributed pricing and task offloading in multi-cloud and multi-edge environments. Computer Networks 200. DOI: http://dx.doi.org/10.1016/j.comnet.2021.108523.

[77] Sharghivand N., Derakhshan F. & Siasi N. (2021) A comprehensive survey on auction mechanism design for cloud/edge resource management and pricing. IEEE Access 9, pp. 126502–126529. DOI: http://dx.doi.org/10.1109/ACCESS.2021.3110914.

[78] Baek B., Lee J., Peng Y. & Park S. (2020) Three dynamic pricing schemes for resource allocation of edge computing for iot environment. IEEE Internet of Things Journal 7(5), pp. 4292–4303. DOI: http://dx.doi.org/10.1109/JIOT.2020.2966627.

[79] Luong N.C., Wang P., Niyato D., Wen Y. & Han Z. (2017) Resource management in cloud networking using economic analysis and pricing models: A survey. IEEE Communications Surveys & Tutorials 19(2), pp. 954–1001. DOI: http://dx.doi.org/10.1109/COMST.2017.2647981.

[80] Xiong Z., Feng S., Wang W., Niyato D., Wang P. & Han Z. (2019) Cloud/fog computing resource management and pricing for blockchain networks. IEEE Internet of Things Journal 6(3), pp. 4585–4600. DOI: http://dx.doi.org/10.1109/JIOT.2018.2871706.

[81] Chen Y., Li Z., Yang B., Nai K. & Li K. (2020) A Stackelberg game approach to multiple resources allocation and pricing in mobile edge computing. Future Generation Computer Systems 108, pp. 273–287. DOI: http://dx.doi.org/10.1016/j.future.2020.02.045.

[82] Kumar D., Baranwal G. & Vidyarthi D.P. (2022) A survey on auction based approaches for resource allocation and pricing in emerging edge technologies. Journal of Grid Computing 20(1). DOI: http://dx.doi.org/10.1007/s10723-021-09593-9.

[83] Zhan Y. & Zhang J. (2020) An incentive mechanism design for efficient edge learning by deep reinforcement learning approach. In: IEEE INFOCOM 2020 - IEEE Conference on Computer Communications, Toronto, ON, Canada, pp. 2489–2498. DOI: http://dx.doi.org/10.1109/INFOCOM41043.2020.9155268.

[84] Chen S., Li L., Chen Z. & Li S. (2021) Dynamic pricing for smart mobile edge computing: A reinforcement learning approach. IEEE Wireless Communications Letters 10(4), pp. 700–704. DOI: http://dx.doi.org/10.1109/LWC.2020.3039863.

[85] Palomar D. & Chiang M. (2006) A tutorial on decomposition methods for network utility maximization. IEEE Journal on Selected Areas in Communications 24(8), pp. 1439–1451. DOI: http://dx.doi.org/10.1109/JSAC.2006.879350.

[86] Jennings B. & Stadler R. (2015) Resource management in clouds: Survey and research challenges. Journal of Network and Systems Management 23(3), pp. 567–619. DOI: http://dx.doi.org/10.1007/s10922-014-9307-7.

[87] Kim S.H., Park S., Chen M. & Youn C.H. (2018) An optimal pricing scheme for the energy-efficient mobile edge computation offloading with OFDMA. IEEE Communications Letters 22(9), pp. 1922–1925. DOI: http://dx.doi.org/10.1109/LCOMM.2018.2849401.

[88] Liu L., Chang Z., Guo X., Mao S. & Ristaniemi T. (2018) Multiobjective optimization for computation offloading in fog computing. IEEE Internet of Things Journal 5(1), pp. 283–294. DOI: http://dx.doi.org/10.1109/JIOT.2017.2780236.

[89] Liu M. & Liu Y. (2018) Price-based distributed offloading for mobile-edge computing with computation capacity constraints. IEEE Wireless Communications Letters 7(3), pp. 420–423. DOI: http://dx.doi.org/10.1109/LWC.2017.2780128.

[90] Liu Y., Yu H., Xie S. & Zhang Y. (2019) Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. IEEE Transactions on Vehicular Technology 68(11), pp. 11158–11168. DOI: http://dx.doi.org/10.1109/TVT.2019.2935450.

[91] Wang Q., Guo S., Wang Y. & Yang Y. (2019) Incentive mechanism for edge cloud profit maximization in mobile edge computing. In: ICC 2019 - 2019 IEEE International Conference on Communications (ICC), Shanghai, China, pp. 1–6. DOI: `http://dx.doi.org/10.1109/ICC.2019.8761241`.

[92] Sufyan F. & Banerjee A. (2020) Computation offloading for distributed mobile edge computing network: A multiobjective approach. IEEE Access 8, pp. 149915–149930. DOI: `http://dx.doi.org/10.1109/ACCESS.2020.3016046`.

[93] Yi C., Cai J. & Su Z. (2020) A multi-user mobile computation offloading and transmission scheduling mechanism for delay-sensitive applications. IEEE Transactions on Mobile Computing 19(1), pp. 29–43. DOI: `http://dx.doi.org/10.1109/TMC.2019.2891736`.

[94] Liang B., Fan R., Hu H., Zhang Y., Zhang N. & Anpalagan A. (2021) Nonlinear pricing based distributed offloading in multi-user mobile edge computing. IEEE Transactions on Vehicular Technology 70(1), pp. 1077–1082. DOI: `http://dx.doi.org/10.1109/TVT.2020.3045473`.

[95] Li Y., Yang B., Wu H., Han Q., Chen C. & Guan X. (2022) Joint offloading decision and resource allocation for vehicular fog-edge computing networks: A contract-Stackelberg approach. IEEE Internet of Things Journal 9(17), pp. 15969–15982. DOI: `http://dx.doi.org/10.1109/JIOT.2022.3150955`.

[96] Liu Z. & Fu J. (2022) Resource pricing and offloading decisions in mobile edge computing based on the Stackelberg game. The Journal of Supercomputing 78(6), pp. 7805–7824. DOI: `http://dx.doi.org/10.1007/s11227-021-04246-w`.

[97] Liu J., Guo S., Liu K. & Feng L. (2022) Resource provision and allocation based on microeconomic theory in mobile edge computing. IEEE Transactions on Services Computing 15(3), pp. 1512–1525. DOI: `http://dx.doi.org/10.1109/TSC.2020.3000050`.

[98] Zhang K., Mao Y., Leng S., Maharjan S. & Zhang Y. (2017) Optimal delay constrained offloading for vehicular edge computing networks. In: 2017 IEEE International Conference on Communications (ICC), Paris, France, pp. 1–6. DOI: `http://dx.doi.org/10.1109/ICC.2017.7997360`.

[99] Kim Y., Kwak J. & Chong S. (2018) Dual-side optimization for cost-delay tradeoff in mobile edge computing. IEEE Transactions on Vehicular Technology 67(2), pp. 1765–1781. DOI: `http://dx.doi.org/10.1109/TVT.2017.2762423`.

[100] Guo S., Hu X., Dong G., Li W. & Qiu X. (2019) Mobile edge computing resource allocation: A joint Stackelberg game and matching strategy. International Journal of Distributed Sensor Networks 15(7). DOI: `http://dx.doi.org/10.1177/1550147719861556`.

[101] Shi B., Chen F. & Tang X. (2021) Deep reinforcement learning based task offloading strategy under dynamic pricing in edge computing. In: H. Hacid, O. Kao, M. Mecella, N. Moha & H.y. Paik (eds.) Service-Oriented Computing, Springer, Cham, pp. 578–594. DOI: http://dx.doi.org/10.1007/978-3-030-91431-8_36.

[102] Lyu F., Cai X., Wu F., Lu H., Duan S. & Ren J. (2022) Dynamic pricing scheme for edge computing services: A two-layer reinforcement learning approach. In: 2022 IEEE/ACM 30th International Symposium on Quality of Service (IWQoS), Oslo, Norway, pp. 1–10. DOI: http://dx.doi.org/10.1109/IWQoS54832.2022.9812869.

[103] Li L., Siew M., Chen Z. & Quek T.Q. (2021) Optimal pricing for job offloading in the mec system with two priority classes. IEEE Transactions on Vehicular Technology 70(8), pp. 8080–8091. DOI: http://dx.doi.org/10.1109/TVT.2021.3090935.

[104] Seo H., Oh H., Choi J.K. & Park S. (2022) Differential pricing-based task offloading for delay-sensitive iot applications in mobile edge computing system. IEEE Internet of Things Journal 9(19), pp. 19116–19131. DOI: http://dx.doi.org/10.1109/JIOT.2022.3163820.

[105] Baltzis K.B. (2011) Hexagonal vs circular cell shape: A comparative analysis and evaluation of the two popular modeling approximations. In: A. Melikov (ed.) Cellular Networks - Positioning, Performance Analysis, Reliability, InTech, pp. 103–122. DOI: http://dx.doi.org/10.5772/14851.

[106] MacDonald V.H. (1979) Advanced mobile phone service: The cellular concept. Bell System Technical Journal 58(1), pp. 15–41. DOI: http://dx.doi.org/10.1002/j.1538-7305.1979.tb02209.x.

[107] Newman M.E.J. (2010) Networks: An Introduction. Oxford University Press, Oxford, New York.

[108] Albert R., Jeong H. & Barabasi A.L. (1999) Diameter of the world-wide web. Nature 401, pp. 130–131. DOI: http://dx.doi.org/10.1038/43601.

[109] Jia M., Cao J. & Liang W. (2017) Optimal cloudlet placement and user to cloudlet allocation in wireless metropolitan area networks. IEEE Transactions on Cloud Computing 5(4), pp. 725–737. DOI: http://dx.doi.org/10.1109/TCC.2015.2449834.

[110] Meng J., Shi W., Tan H. & Li X. (2017) Cloudlet placement and minimum-delay routing in cloudlet computing. In: 2017 3rd International Conference on Big Data Computing and Communications (BIGCOM), Chengdu, China, pp. 297–304. DOI: http://dx.doi.org/10.1109/BIGCOM.2017.58.

[111] Yao H., Bai C., Xiong M., Zeng D. & Fu Z. (2017) Heterogeneous cloudlet deployment and user-cloudlet association toward cost effective fog computing. Concurrency and Computation: Practice and Experience 29(16). DOI: http://dx.doi.org/10.1002/cpe.3975.

[112] Soltan S. & Zussman G. (2016) Generation of synthetic spatially embedded power grid networks. In: 2016 IEEE Power and Energy Society General Meeting (PESGM), Boston, MA, USA, pp. 1–5. DOI: `http://dx.doi.org/10.1109/PESGM.2016.7741383`.

[113] Lähderanta T., Leppänen T., Ruha L., Lovén L., Harjula E., Ylianttila M., Riekki J. & Sillanpää M.J. (2021) Edge computing server placement with capacitated location allocation. Journal of Parallel and Distributed Computing 153, pp. 130–149. DOI: `http://dx.doi.org/10.1016/j.jpdc.2021.03.007`.

[114] Golbeck J. (2013) Chapter 3 - Network structure and measures. In: J. Golbeck (ed.) Analyzing the Social Web, Morgan Kaufmann, Boston, pp. 25–44. DOI: `http://dx.doi.org/10.1016/B978-0-12-405531-5.00003-1`.

[115] Aggarwal C.C. (2014) An introduction to cluster analysis. In: C.C. Aggarwal & C.K. Reddy (eds.) Data Clustering: Algorithms and Applications, Chapman and Hall/CRC, pp. 1–28. DOI: `http://dx.doi.org/10.1201/9781315373515-1`.

[116] Reddy C. & Vinzamuri B. (2014) A survey of partitional and hierarchical clustering algorithms. In: C.C. Aggarwal & C.K. Reddy (eds.) Data Clustering: Algorithms and Applications, Chapman and Hall/CRC, pp. 87–110. DOI: `http://dx.doi.org/10.1201/9781315373515-4`.

[117] Sutton R.S. & Barto A.G. (2018) Reinforcement learning: An introduction. MIT Press, Cambridge, MA, 2nd ed.

[118] Grondman I., Busoniu L., Lopes G.A.D. & Babuska R. (2012) A survey of actor-critic reinforcement learning: Standard and natural policy gradients. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42(6), pp. 1291–1307. DOI: `http://dx.doi.org/10.1109/TSMCC.2012.2218595`.

[119] Watkins C. & Dayan P. (1992) Q-learning. Machine Learning 8, pp. 279–292. DOI: `http://dx.doi.org/10.1007/BF00992698`.

[120] Rummery G. & Niranjan M. (1994), On-line q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166.

[121] Williams R.J. (1992) Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning 8, pp. 229–256. DOI: `http://dx.doi.org/10.1007/BF00992696`.

[122] Wang X., Wang S., Liang X., Zhao D., Huang J., Xu X., Dai B. & Miao Q. (2022) Deep reinforcement learning: A survey. IEEE Transactions on Neural Networks and Learning Systems (Early access), pp. 1–15. DOI: `http://dx.doi.org/10.1109/TNNLS.2022.3207346`.

[123] Li Y. (2018), Deep reinforcement learning: An overview. arXiv:1701.07274. DOI: `http://dx.doi.org/10.48550/ARXIV.1701.07274`.

[124] Hernandez-Leal P., Kartal B. & Taylor M.E. (2019) A survey and critique of multiagent deep reinforcement learning. Autonomous Agents and Multi-Agent Systems 33(6), pp. 750–797. DOI: http://dx.doi.org/10.1007/s10458-019-09421-1.

[125] Hernandez-Leal P., Kaisers M., Baarslag T. & de Cote E.M. (2019), A survey of learning in multiagent environments: Dealing with non-stationarity. arXiv:1707.09183. DOI: http://dx.doi.org/10.48550/ARXIV.1707.09183.

[126] Zhang K., Yang Z. & Başar T. (2021) Multi-agent reinforcement learning: A selective overview of theories and algorithms. In: K.G. Vamvoudakis, Y. Wan, F.L. Lewis & D. Cansever (eds.) Handbook of Reinforcement Learning and Control, Springer, Cham, pp. 321–384. DOI: http://dx.doi.org/10.1007/978-3-030-60990-0_12.

[127] Gronauer S. & Diepold K. (2022) Multi-agent deep reinforcement learning: A survey. Artificial Intelligence Review 55(2), pp. 895–943. DOI: http://dx.doi.org/10.1007/s10462-021-09996-w.

[128] Lovén L., Peltonen E., Ruha L., Harjula E. & Pirttikangas S. (2022) A dark and stormy night: Reallocation storms in edge computing. EURASIP Journal on Wireless Communications and Networking 2022(1). DOI: http://dx.doi.org/10.1186/s13638-022-02170-y.

[129] Li B., Wang K., Xue D. & Pei Y. (2018) K-means based edge server deployment algorithm for edge computing environments. In: 2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI), Guangzhou, China, pp. 1169–1174. DOI: http://dx.doi.org/10.1109/SmartWorld.2018.00203.

[130] Guo Y., Wang S., Zhou A., Xu J., Yuan J. & Hsu C.H. (2020) User allocation-aware edge cloud placement in mobile edge computing. Software: Practice and Experience 50(5), pp. 489–502. DOI: http://dx.doi.org/10.1002/spe.2685.

[131] Wang S., Zhao Y., Xu J., Yuan J. & Hsu C.H. (2019) Edge server placement in mobile edge computing. Journal of Parallel and Distributed Computing 127, pp. 160–168. DOI: http://dx.doi.org/10.1016/j.jpdc.2018.06.008.

[132] LeCun Y.A., Bottou L., Orr G.B. & Müller K.R. (2012) Efficient backprop. In: G. Montavon, G.B. Orr & K.R. Müller (eds.) Neural Networks: Tricks of the Trade, Springer, Berlin, Heidelberg, pp. 9–48. DOI: http://dx.doi.org/10.1007/978-3-642-35289-8_3.

[133] Henderson P., Islam R., Bachman P., Pineau J., Precup D. & Meger D. (2018) Deep reinforcement learning that matters. Proceedings of the AAAI Conference on Artificial Intelligence 32(1). DOI: http://dx.doi.org/10.1609/aaai.v32i1.11694.

[134] Papadimitriou C.H. (1981) On the complexity of integer programming. Journal of the ACM 28(4), pp. 765–768. DOI: http://dx.doi.org/10.1145/322276.322287.

[135] Xiao M., Shroff N. & Chong E. (2003) A utility-based power-control scheme in wireless cellular systems. IEEE/ACM Transactions on Networking 11(2), pp. 210–221. DOI: http://dx.doi.org/10.1109/TNET.2003.810314.

[136] Calvano E., Calzolari G., Denicolò V. & Pastorello S. (2020) Artificial intelligence, algorithmic pricing, and collusion. American Economic Review 110(10), pp. 3267–3297. DOI: http://dx.doi.org/10.1257/aer.20190623.

[137] Beneke F. & Mackenrodt M.O. (2018) Artificial intelligence and collusion. IIC - International Review of Intellectual Property and Competition Law 50, pp. 109–134. DOI: http://dx.doi.org/10.1007/s40319-018-00773-x.

9. APPENDICES

Table 7. Command-line arguments available in EISim

Option	Long Option	Required	Description	Default
i	input	Yes	Path to the folder that contains the setting files	-
o	output	Yes	Path to the folder where the simulation results are saved	-
m	model-folder	Yes	Path to the folder where the agent models are saved	-
s	seed	No	A random seed for the simulation	-
T	train	No	A flag for turning on the training mode	-
R	random-steps	No	Determines how many times at the beginning of a simulation each pricing agent decides the price randomly	0
r	replay-size	No	The length of the experience replay	2000
b	batch-size	No	The size of a mini-batch for training	128
d	discount-factor	No	The reward discount factor	0.99
a	actorlr	No	The learning rate for actor network	0.001
c	criticlr	No	The learning rate for critic network	0.001
t	tau	No	The parameter used for updating the actor and critic target networks	0.005
u	updates	No	Determines how many times models are updated at the beginning of a new slot	1
N	noisesd	No	The standard deviation for a zero-mean Gaussian noise process	0.5
D	noise-decay	No	The rate at which noise in action selection is decayed during training	1e-6

Table 8. Edge server specifications used in EISim evaluation

	20 high-capacity servers	**100 low-capacity servers**
Idle consumption (W)	105	45
Max consumption (W)	185	95
Cores	15	6
MIPS per core	20,000	10,000
RAM (MB)	80,000	16,000
Storage (MB)	1,280,000	256,000

Table 9. Mobile device specifications used in EISim evaluation

	Device type 1	**Device type 2**	**Device type 3**	**Device type 4**
Percentage	30	40	20	10
Speed (m/s)	1.1	1.1	0.6	0
Min pause duration (s)	60	60	180	0
Max pause duration (s)	300	300	600	0
Min mobility duration (s)	60	60	60	0
Max mobility duration (s)	300	300	300	0
Battery capacity (Wh)	19.25	15.4	25.9	56.5
Idle consumption (W)	0.9	0.6	1.1	1.7
Max consumption (W)	6.2	5.5	6.5	23.6
Cores	6	4	4	6
MIPS per core	6,000	4,000	3,000	7,000
RAM (MB)	6,000	4,000	2,000	8,000
Storage (MB)	128,000	64,000	32,000	256,000

Table 10. Application specification used in EISim evaluation

Parameter	**Value**
Poisson rate	1
Latency (s)	0.5
Request size (kB)	U(100, 1000)
Container size (kB)	Equal to request size
Result size (kB)	U(0.2, 0.8) * request size
Task length (MIs)	exp(2000)

Table 11. Simulation parameters used in EISim evaluation

Parameter	**Meaning**	**Value**
simulation_time	Simulation time in minutes	60 (1 hour)
parallel_simulation	Whether to run simulation scenarios in parallel	true
update_interval	Mobility and energy consumption update interval for the computing nodes in seconds	1
display_real_ time_charts	Whether to display charts	false
length	Length of the simulation area in meters	1100
width	Width of the simulation area in meters	1100
edge_devices_ range	The radius in which two devices, or a device and an edge datacenter, can offload to each other in meters	48
edge_datacenters_ coverage	Coverage in meters	48
enable_registry	Whether the device will download the application from a registry after receiving the offloaded task	false
enable_orchestrators	When enabled, tasks will be sent to a another device/server to make the offloading decisions, otherwise each device makes its own offloading decisions	false
wait_for_all_tasks	Whether to stop the simulation when the time ends or wait for all the tasks to get executed	true
batch_size	For scheduling tasks in batches to reduce the event queue size	100
min_number_ of_edge_devices	Minimum number of edge devices	1000
max_number_ of_edge_devices	Maximum number of edge devices	4000
edge_device_ counter_size	Step size for the edge device count	1000
realistic_network_ model	Whether to simulate transfers in network links more realistically	false
network_update_ interval	Transfer update interval for network links in seconds	1

Continued on the next page

Table 11. Simulation parameters used in EISim evaluation (continued)

Parameter	**Meaning**	**Value**
man_bandwidth	MAN link bandwidth in megabits per seconds (Mbps)	1000
man_latency	MAN link latency in seconds	0.005
man_nanojoules_ per_bit	MAN link energy consumption per transferred bit	0
wifi_bandwidth	Wi-Fi bandwidth in Mbps	1300
wifi_device_ transmission_ nanojoules_per_bit	Edge device energy consumption per transmitted bit	283.17
wifi_device_ reception_ nanojoules_per_bit	Edge device energy consumption per received bit	137.01
wifi_access_ point_ transmission_ nanojoules_per_bit	AP energy consumption per transmitted bit	23.8
wifi_access_ point_ reception_ nanojoules_per_bit	AP energy consumption per received bit	23.8
wifi_latency	Wi-Fi link latency in seconds	0.0025
orchestration_ architectures	Defines which nodes can be considered as offloading destinations in orchestration	EDGE_ ONLY