TU WIEN Informatics

Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm for latency sensitive applications

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Manuel Lindner, BSc
Matrikelnummer 01528224

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Ivona Brandić

Wien, 14. Mai 2020

Manuel Lindner

Ivona Brandić

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

Informatics

Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm for latency sensitive applications

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Manuel Lindner, BSc
Registration Number 01528224

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Ivona Brandić

Vienna, 14th May, 2020

Manuel Lindner   Ivona Brandić

Erklärung zur Verfassung der Arbeit

Manuel Lindner, BSc

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

Wien, 14. Mai 2020

[illegible]

Manuel Lindner

Danksagung

Ich möchte mich recht herzlich bei meiner Betreuerin Univ. Prof. Dr. Ivona Brandić bedanken, die mich tatkräftig bei der Erstellung der Diplomarbeit unterstützt hat. Sie hat mir immer wieder wertvolles und konstruktives Feedback übermittelt, das mir geholfen hat, die Arbeit laufend zu verbessern.

Außerdem möchte ich mich bei meiner Familie bedanken, die mir während meines gesamten Studiums sozialen als auch finanziellen Rückhalt gegeben hat.

Acknowledgements

I would like to thank my supervisor Univ. Prof. Dr. Ivona Brandić who supported me in creating this diploma thesis. She has always provided valuable and constructive feedback, which helped me to constantly improve my work.

Additionally, I would also like to express great gratitude to my family for their moral and financial support throughout the whole studies.

Kurzfassung

In den letzten Jahrzehnten haben es technologische Verbesserungen ermöglicht, die Komplexität in mobilen Applikationen zu erhöhen und sie einer großen Anzahl von Benutzern zur Verfügung zu stellen. Daher ist das Konzept des Task-Offloading zu einer attraktiven Lösung für Mobilgeräte geworden, um den Problemen der begrenzten Rechenkapazität und Akkulaufzeit entgegenzuwirken, da Aufgaben zur Ausführung an eine Remote-Infrastruktur gesendet werden. Zusätzlich hat das Konzept des Edge Computing in den letzten Jahren an Bedeutung gewonnen, um End-to-End-Kommunikation in Echtzeit zu erreichen. Dies wird ermöglicht, da sich die Edge Server in geringer Entfernung zu den mobilen Endgeräten befinden.

In dieser Arbeit beschäftigen wir uns mit der Herausforderung, verschiedene Aufgaben einer mobilen Anwendung mit unterschiedlicher Wichtigkeit zu behandeln, um die Latenz von sensiblen Aufgaben zu verringern. Aus diesem Grund haben wir einen Priority Based Mobile Edge Cloud Offloading (PBMECO) Algorithmus entwickelt, der in einer Edge Cloud Computing Umgebung ausgeführt wird. Dabei werden Aufgaben der mobilen Anwendung entweder an Edge-Knoten oder die öffentliche Cloud gesendet. Unser Ansatz trifft Offloading-Entscheidungen basierend auf den vordefinierten Prioritäten HIGH, MEDIUM und LOW sowie den Ressourcenanforderungen der Aufgabe. Diese drei Prioritäten können sowohl der Kommunikations- als auch der Berechnungslatenz zugewiesen werden, um eine granulare Differenzierung für das Finden einer passenden Zielinstanz zu ermöglichen. Die Kommunikationslatenz spezifiziert hierbei die Netzwerkübertragung, währenddessen die Berechnungslatenz verwendet wird, um die Priorität für die Ausführungszeit festzulegen.

Zusätzlich haben wir noch ein Simlationsframework entwickelt, das auch zur Bewertung unserer PBMECO-Implementierung verwendet wird. Das Simlationsframework basiert auf einer Monte Carlo Simulation und wir haben die Simulation mit drei unterschiedlichen realen Anwendungen durchgeführt, um Einblicke in die Performance unseres Algorithmus im Vergleich zur zufälligen Auswahl von Zielknoten unter Vernachlässigung von Prioritäten zu erhalten. Die Simulationsergebnisse haben gezeigt, dass die Performance unserer Lösung bei steigender Anzahl von Aufgaben mit hoher Priorität konstant bleibt. Darüber hinaus haben die numerischen Ergebnisse gezeigt, dass die Kommunikationslatenz von Aufgaben mit hoher Priorität um bis zu 65% reduziert werden kann, und dass unser Offloading-Ansatz eine Latenzreduktion von bis zu 30% für Aufgaben mit hoher Berechnungslatenz erreichen kann.

Abstract

In the last decades, technological improvements have made it possible to increase the complexity in mobile applications and to offer them to a large number of users. Therefore, the concept of task offloading has become an attractive solution for mobile applications to overcome the problems of limited processing capabilities and limited battery life because tasks are sent for execution to a remote infrastructure. Additionally, the concept of Edge Computing has gained considerable attention in recent years to achieve near real-time end-to-end communication, which is possible due to the close proximity of edge servers to mobile devices.

In this work, we are tackling the challenge that different tasks of a mobile application need to be treated with different importance to reduce the latency of latency sensitive tasks. Therefore, we have implemented a Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm, which is operating within an Edge Cloud Computing environment. This means, that the tasks of the mobile application can be either sent to edge nodes or the public cloud. Our approach makes a joint offloading decision based on the pre-defined priorities HIGH, MEDIUM and LOW and the task's resource requirements. These three priorities can be assigned to each task for both the communication and computation latency to enable more granular differentiation for finding offloading targets. Hereby, the communication latency specifies the importance with respect to network transfer whereas computation latency is used to set the priority for the execution time.

We have additionally proposed a simulation framework, which is also used to evaluate our PBMECO implementation. The simulation framework is using Monte Carlo Simulation and we have applied three different real-world mobile applications to get insights about the performance of the algorithm compared to randomly selecting target nodes and ignoring task priorities. Firstly, the simulation results have illustrated that the performance of our solution remains constant when the number of HIGH prioritized tasks increases. Furthermore, the numerical results have shown that the communication latency of HIGH priority tasks can be reduced by up to 65% and it has been demonstrated that our offloading approach can achieve a latency reduction by up to 30% for tasks with computation latency HIGH.

Contents

CHAPTER 1

Introduction

This chapter starts with the motivation for this diploma thesis. Afterwards, it provides the description of the problem statement, which this diploma thesis is tackling. The problem statement is followed by the defined research questions, expected results and methodological approach. Finally, this chapter closes with an overview about how this diploma thesis is structured to provide a guideline for the reader.

1.1 Motivation

The two topics Internet of Things (IoT) and Mobile Computing can be tracked back to the end of the 20th and the beginning of the 21st century [A$^+$09, p. 1] [Pie01, p. 1]. Since the invention of those two paradigms, a growing demand for mobile devices can be recorded and we have reached a notable turning point in October 2016, when more users were accessing the internet via mobile devices than desktop computers [Lö17]. This statement highlights, that the use of smartphones or wearable / IoT devices is already widespread in the community and therefore, it is indispensable in our everyday life. Examples for using mobile devices are areas like healthcare, transports, e-commerce or telecommunication [SGFW10] [Sto02, p. XVII].

As handheld devices have gained more interest in recent years, mobile applications consequently become more complex and provide an increasing amount of services [DMB18, p. 1]. The growing number of available applications leads also to an increase of data that is produced and must be transferred over the network. Cisco Systems Inc. states, that the total amount of data created by IoT-devices will reach 600 ZB per year by 2020, up from 145 ZB per year in 2015 [Cis16, p. 3], which means quadrupling the amount of data. Additionally, the increasing complexity and functionality of the mobile applications requires more processing power which in further consequence leads to higher energy consumption [DMB18, p. 1].

Therefore, mobile devices are facing two challenges. On the one hand, mobile devices have limited processing capabilities and on the other hand they have only a limited battery lifetime. To overcome these two problems, the concept of task offloading has become an interesting approach for researchers and companies. Kumar and Lu have shown that offloading helps to save energy on the mobile devices [KL10, p. 1] whereas Palmer et al. emphasize that more complex operations can be executed when offloading tasks to a Grid [PKKB09, p. 3]. Task offloading means that the execution of tasks is done on a remote environment including physical or virtual instances. In the beginnings, the target environment was the cloud infrastructure because it provides almost unlimited processing power. But due to the rising demand in near real-time applications and the increasing network traffic, Cloud Computing has reached its limits and the new concept of Edge Computing has been introduced. "Near Real-Time means effectively Real-Time but without guarantees of hitting specific deadlines. Also known as soft real-time." [OnL11]. Hereby, a key-driver for handling the increasing data is the invention of the 5G network. It is built as an ultra-dense network including a huge number of small cell and macro cell base stations [CHQ+16]. To achieve near real-time communication, the Edge Computing paradigm has been proposed to provide computing services with short delay and high performance [CH18, p. 1]. The edge environment consists of so-called Micro Data Centers (MDCs) or cloudlets and are used to bring the processing power closer to the end devices and build the path along the edge nodes and the cloud data center [SCZ+16, p. 1] [GHMP08, p. 2] [SBCD09, p. 6]. This enables the execution of application tasks on remote instances that are in close proximity to the end user. Examples of near real-time applications that benefit from Edge Computing are augmented reality, face recognition or navigation systems.

In order to perform remote task executions, it is necessary that each mobile devices is equipped with an offloading algorithm. The aim of the algorithm is to reduce latency times in order to achieve near real-time communication. Therefore, on the one hand it has to identify which task can be sent to a remote instance for execution and on the other hand to find the most suitable target instance where the task can be deployed.

1.2 Problem Statement

This diploma thesis deals with the topic of offloading tasks of a mobile application into an Edge Cloud Computing environment. In theory, Edge Computing is a natural extension of the Content Delivery Network (CDN) architecture. It is a new concept which pushes business logic as well as data processing from corporate data centers out to proxy servers at the "Edge" of the network [PT04, p. 1]. This means that Edge Computing tries to bring the processing power closer to end users, sensors, machines or devices [Sem16] [SCZ+16, p. 1]. The paradigm of Edge Computing states that edge devices can communicate with each other and make decisions without interacting with the cloud [Sem16]. Emerging applications in the field of Internet of Things (IoT) like Connected Cars, Industry 4.0 or Smart Cities, where the fundamental aspects are real-time data processing and decision-making based on the calculated data, rely on this new approach of Edge Computing [Sem16].

To achieve near or real-time end-to-end communication the main metrics that needs to be focused on is network and computation latency [Roe17] [TKS05]. According to Beal [Bea] network latency describes the time taken until a packet has been transmitted from source to destination. Based on this definition the essential point to focus on is the distance between the producer and the consumer. The larger the distance for data transmission the higher is the latency. And with respect to computation latency, it is essential that the execution is done on instances with high computational power.

To tackle the previously mentioned problem regarding distance and the impact on latency, the end user device needs to find an edge device nearby. But start scanning the device's surroundings every time on-demand, which means when there is a need to process something [Pre] on an edge node, time is lost. Even if it is just 1 Milli Second (ms), this time is essential for latency sensitive applications and may cause them to fail.

Furthermore, another problem arises when such an application is deployed in an Edge Computing environment and all parts of the application like tasks are treated equally. This may cause unimportant tasks to block resources. As a result, urgent tasks cannot make use of the best edge nodes (like closest or most powerful nodes) which in further consequence leads again to higher latency times.

1.3 Research Questions

Based on the problems mentioned within the problem statement the following main research question is derived:

What is an appropriate means for the algorithm to reduce the latency of latency sensitive tasks on mobile devices in an Edge Cloud Computing environment that have execution dependencies and can be offloaded to data centers?

In order to answer the given main research question, it is split into the following sub-questions and objectives:

Q1. What benefits and limitations exist in Edge and Cloud Computing for offloading latency sensitive tasks?
When dealing with task offloading in an Edge Cloud Computing environment, the first step is to identify what benefits and limitations exist in both paradigms. Therefore, both concepts must be analyzed towards these aspects to get general information about their strengths and weaknesses and how one's benefits can help to overcome the other's limitations.

Q2. What is an appropriate means to find nearby nodes to offload the task?
To quickly perform the offloading of a task – especially a latency sensitive task – the algorithm must be able to find and select a nearby suitable node in a fast way. To achieve this goal, this sub-question deals with the implementation of an appropriate concept.

Q3. What is an appropriate means for the algorithm to process the application's tasks?

a) *What is a proper way to handle execution dependencies between tasks?*
In this sub-question a mechanism which is able to deal with dependencies between tasks is developed. This is necessary because some upcoming tasks may rely on data from previous tasks.

b) *What is a better way to prioritize the upcoming tasks and decide where to offload?*
This sub-question will result in a concept on how to handle tasks with different priorities regarding latency sensitivity. When the algorithm is processing an application, tasks with high priority must take precedence over low prioritized one's in order to minimize latency.

c) *What is an appropriate means to deal with unforeseen tasks?*
When e.g. an emergency application is started during a regular offloading process it is required that the algorithm can handle the offloading of the emergency application too. Therefore, in this sub-question a concept is provided which allows the algorithm to deal with this scenario.

Q4. Propose a simulation framework in order to evaluate the performance of the algorithm.
In order to evaluate whether the designed and implemented algorithm fulfills the previously mentioned requirements like task dependencies, task prioritization and unforeseen task processing, multiple simulations must be executed. Thus, a simulation framework needs to be designed to measure the average latency of the tasks. The results of several simulation runs applying the structured algorithm versus random offloading of tasks can be compared afterwards to show that the algorithm can reduce latency times, especially for latency sensitive tasks.

1.4 Expected Results

In the end of this diploma thesis the following results should be achieved:

- Overview about benefits and limitations of Edge and Cloud Computing for task offloading
- Latency times for high priority tasks during the offloading process can be decreased by applying the implemented Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm
- The implemented simulation framework can be used to simulate different task offloading scenarios in an Edge Cloud Computing environment

1.5 Methodological Approach

This diploma thesis is guided by the following methodological approach for answering the research questions described above:

Systematic mapping study

A systematic mapping study based on Peterson et al. [PFMM08] is made to gather the benefits and limitations of Cloud and Edge Computing. Furthermore, the advantages and disadvantages of both concepts are analyzed to find out how each concept can be integrated in and supports the task offloading process.

Cloud Computing aspects

The advantages of the concept of Cloud Computing are applied to the task offloading process in order to provide a fully reliable algorithm. Hereby, the high availability and scalability of Cloud Computing can be perfectly used as fallback mechanism when no edge node is available.

Edge Computing aspects

The Edge Computing paradigm and its benefits are used to support the task offloading process to minimize the latency of sensitive tasks. Due to the proximity to end user nodes this concept is best suited to be applied in this area. Furthermore, as the edge nodes are interconnected via a high-speed connection, Edge Computing also enables migrations between the different edge nodes.

First Fit (FF) offloading algorithm

When executing the task offloading process the algorithm must select the best node where to offload the task. The best node is defined as "the closest node which has enough resources to execute the task". The closest nodes are passively tracked and sorted in the background. Therefore, the algorithm acts as First Fit (FF) offloading algorithm because during the offloading process the sorted nodes are checked one by one and it takes the first node which fulfills all resource requirements of the offloaded task with respect to task priorities.

Directed Acyclic Graph (DAG)
The dependencies between the tasks of an application are modelled as a Directed Acyclic Graph (DAG). By using this technique, the predecessors for each task can be defined to identify when a task can be offloaded or must wait for others.

Predictive model
The algorithm also makes use of a predictive model based on historical data to assume how many resources will be needed per task priority for upcoming tasks. This information is used to potentially reserve resources on close nodes for high priority tasks.

Event-driven Monte Carlo Simulation
The Monte Carlo Simulation helps to simulate the expected offloading latency times when using the implemented algorithm compared to random offloading of tasks. This simulation method supports to model the tasks' volatile resource requirements and the movement of an end user node. By executing a high number of simulation runs and based on the Law of Large Numbers we can expect a confident result for the simulated latency times [WH16, p. 8]. As the state of the model is only changed when an event occurs (e.g. task is offloaded or mobile node is moved) an event-driven Monte Carlo Simulation is used [WH16, p. 79].

1.6 Structure of Work

This diploma thesis starts in Chapter 2 by providing the necessary background information about the several concepts like Edge, Cloud and Mobile Computing, and Task Offloading that are used throughout the work. In Chapter 3 it continues by pointing out existing publications in the area of Mobile Edge Cloud Offloading (MECO) and a discussion on the lack of knowledge, which this thesis is tackling. Afterwards, Chapter 4 covers all the details about the conducted systematic mapping study for identifying the benefits and limitations of both the paradigms Cloud and Edge Computing. In Chapter 5 we are describing firstly what concepts we are using within our algorithm. Secondly, we provide a generic overview of the algorithmic procedure followed by a detailed technical description of the whole implementation of our PBMECO algorithm. Chapter 6 includes covers all information about the evaluation we have made for measuring the performance of our solution. Finally, this diploma thesis closes with a discussion of our findings and points out limitations that are not covered within this work in Chapter 7. This chapter also gives some ideas about future research that can be made upon this thesis.

CHAPTER 2

Background

This introductory chapter provides some background information about the concepts Cloud, Edge and Mobile Computing to get an understanding how they differ. The chapter covers details about the evolution as well as a description of each paradigm is given. Furthermore, the chapter is completed by explaining what task offloading actually is and what Mobile Edge Cloud Offloading (MECO) means.

2.1 Cloud Computing

This chapter provides information about the origin of Cloud Computing. Furthermore, the concept and the infrastructure is described in detail.

2.1.1 Evolution

To get an understanding about today's concept of Cloud Computing it is necessary to know its origins and what is expected in the future.

The first concepts of Cloud Computing have been originated in 1957 when the so-called time-sharing concept has been invented by John McCarthy. This concept states that idle CPU times where dynamically distributed to several users. In the beginning, large companies offered their mainframe computing resources, where software packages, programming environments, file storage and printing services were included [BLRK11, p. 9]. The users gained access to the mainframe computer from connected stations called dumb terminals [God18].

A milestone for Cloud Computing has been achieved in the mid-1960s when scientist J.C.R. Licklider presented an interconnected system of computers which lead to the development of the first network of computers in different physical locations called Advanced Research Projects Agency Network (ARPANET) [God18]. In the 1990s, there was the next new trend for centralizing data storage and businesses began offering virtual private networks which is still true for the current concept of Cloud Computing [BLRK11, p. 9] [God18].

Those concepts - time-sharing, centralized data storage and virtual private networks - had to be adopted by today's main data center operators like Amazon or Google. Böhm et al. [BLRK11, p. 10] explains that those big players had to find a way to provide their computing resources to users because any application needs a model of computation, a model of storage and a model of communication [AFG+10, p. 3].

To achieve this goal, today's paradigm of Cloud Computing makes use of already well-established concepts like grid computing, virtualization or Software-as-a-Service (SaaS). But the real innovation within this new paradigm is how cloud providers have made computing services like software applications, programming platforms, data-storage or computing infrastructure available to the customer [BLRK11, p. 10]. The data center operators have also identified the opportunity in gaining profit by constructing and operating their large-scale data centers at low-cost locations [AFG+10, p. 3].

Over the past decade Cloud Computing has been widely adopted at consumer and enterprise level. Most modern enterprises could not run their business any more without using cloud services [God18]. 451 Research has proposed a study that about 90% of organizations will be interacting with the cloud until 2020 [Mac18]. Furthermore, Cisco Systems Inc. [Cis16, pp. 9-10] has written in *Cisco Global Cloud Index: Forecast and Methodology, 2015-2020* that the annual global cloud traffic will reach 14.1 Zeta Byte (ZB) (1.2 ZB per month) by 2020. This is an increase of about 260% compared to 2015 (3.9 ZB per year in 2015).

2.1.2 Paradigm

The National Institute of Standards and Technology (NIST) describes Cloud Computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [MG+11]. It is characterized by the following five properties [MG+11, p. 6]:

1. **On-demand self-service:** Consumers have the possibility to self-provision the required computing capabilities without human interaction between the consumer and the service provider.
2. **Broad network access:** The cloud resources are accessible through standard mechanisms over the network.
3. **Resource pooling:** The provider's computing resources are pooled using a multi-tenant model and the consumer is dynamically assigned and released physical and virtual resources, where the consumer has no exact control over the location.

4. **Rapid elasticity:** Based on the consumer's demand, the cloud provider can dynamically scale resources.

5. **Measured service:** Cloud services are monitoring the resource usage at some level of abstraction (e.g. storage, bandwidth) to provide transparency for both the provider and consumer.

The following graphic demonstrates the Cloud Computing structure [SCZ+16, p. 2]:

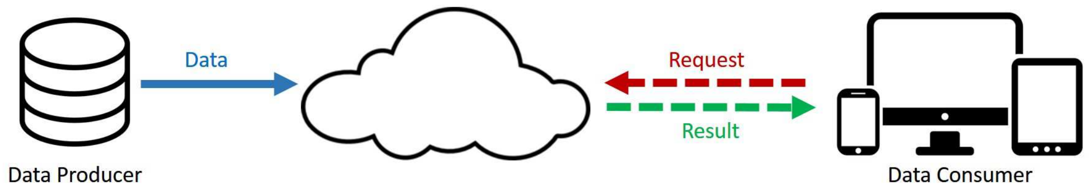

Figure 2.1: Cloud computing paradigm

Figure 2.1 illustrates that there are two actors which are interacting with the cloud environment. On the left side there are the data producers who are generating raw data and sending it to the cloud. And on the right side data consumers are requesting specific data - based on their needs - and the cloud environment responds with the requested data if it exists [SCZ+16, p. 2].

NIST specifies three service models of today's paradigm of Cloud Computing [MG+11, pp. 6-7]:

1. **Software-as-a-Service (SaaS):** SaaS defines the possibility for a consumer to access the provider's applications that are running on a cloud infrastructure via a thin client or program interface. The consumer is neither responsible nor permitted to manage the underlying cloud infrastructure except application specific settings.

2. **Platform-as-a-Service (PaaS):** PaaS allows a consumer to deploy custom applications implemented with provider supported languages, libraries, services and tools. The consumer is neither responsible nor permitted to manage the underlying cloud infrastructure except configuration settings for the application-hosting environment.

3. **Infrastructure-as-a-Service (IaaS):** IaaS provides a consumer the capability to provision computing resources (e.g. processing, storage or network) to deploy any software. The consumer is neither responsible nor permitted to manage the underlying cloud infrastructure but has control over the operating system and deployed software.

The layered service models of Cloud Computing with examples can be seen in figure 2.2 [ZCB10, p. 9]:

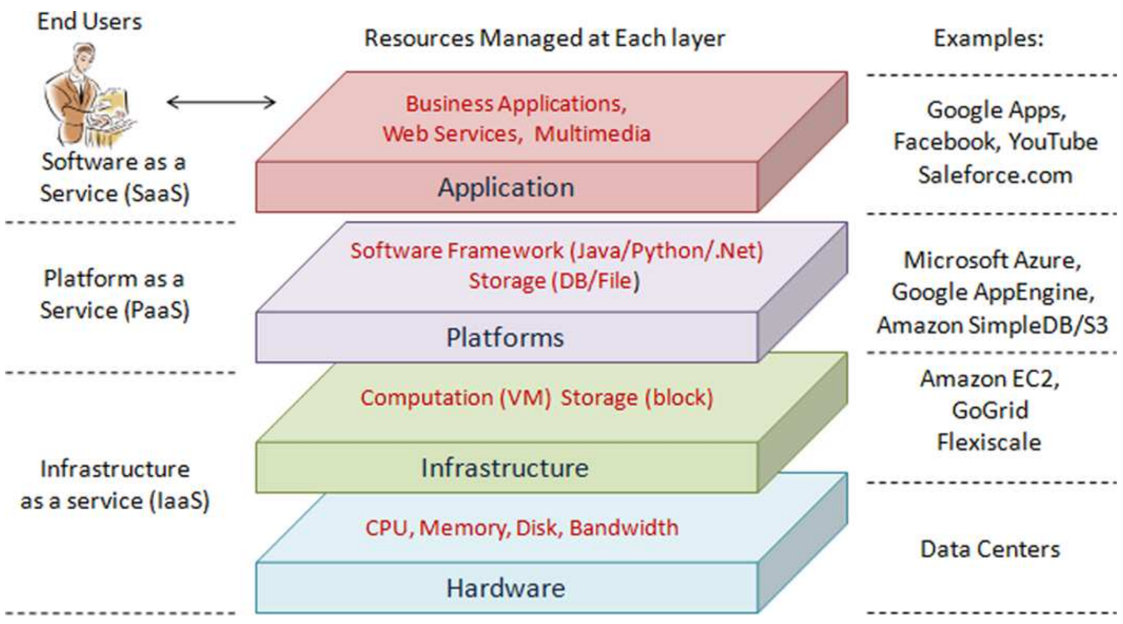
End Users
Resources Managed at Each layer
Examples:
Business Applications, Web Services, Multimedia
Application
Google Apps, Facebook, YouTube Saleforce.com
Software as a Service (SaaS)
Software Framework (Java/Python/.Net) Storage (DB/File)
Platforms
Platform as a Service (PaaS)
Microsoft Azure, Google AppEngine, Amazon SimpleDB/S3
Computation (VM) Storage (block)
Infrastructure
Amazon EC2, GoGrid Flexiscale
Infrastructure as a service (IaaS)
CPU, Memory, Disk, Bandwidth
Hardware
Data Centers

Figure 2.2: Cloud computing architecture

2.2 Edge Computing

This section of the diploma thesis deals with the topic of Edge Computing and its evolution. Additionally, it contains further information about the paradigm of Edge Computing.

2.2.1 Evolution

The concept of Edge Computing is a form of distributed computing, which has already been invented in the 1960s. But the main innovation for Edge Computing – as it is known today – was the launch of Akamai's Content Delivery Network (CDN) in the late 1990s. The CDN operates an edge network with servers to bring content closer to the consumers [Ham18]. Pang and Tan [PT04, p. 1] states that Edge Computing is just a natural extension of the CDN architecture.

Then in 1999, the concept of Internet of Things (IoT) in supply chain management was introduced for the first time by Kevin Ashton. He said that the Information Technology (IT) depends on data originated by people and that it is necessary to empower computers to gather information by themselves. This limitation has been overcome by using Radio Frequency Identification (RFID) and sensor technology [A^{+}09].

According to Shi et al. [SCZ+16, p. 1] IoT has been the starting point of the post-cloud era where things are generating a huge amount of data and applications are deployed at the edge to consume this data. In 2012, the term fog computing was introduced by Cisco to label dispersed cloud infrastructures. Cisco wanted to highlight IoT scalability to support handling of big data volumes for real-time applications [Akt19].

Cisco Systems Inc. [Cis16, p. 3] predicts that the total amount of data created by any device will reach 600 ZB by 2020. Compared to 145 ZB in 2015, the data generated would have been almost quadrupled. Another prediction made by IDC shows that at least 40% of IoT-created data will be stored, processed or analyzed close or at the edge of the network [MTC+16, p. 4].

2.2.2 Paradigm

Shi et al. [SCZ+16, p. 2] defines Edge Computing as the reference to "the enabling technologies allowing computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services". In literature Edge Computing is sometimes also referred to as "fog computing", "local cloud" or "cloudlets" [Wan16a]. According to Hamilton [Ham18] fog computing is a standard whereas Edge Computing is a concept based on that standard.

Moreover, the publication *The NIST Definition of Fog Computing* describes fog computing as "a horizontal, physical or virtual resource paradigm between end devices and cloud data centers" with the following characteristics, which can also be applied to the concept of Edge Computing [IFB+17, pp. 9-10]:

1. **Contextual location awareness and low latency:** As fog nodes are closer to the consumer the response times are reduced.
2. **Geographical distribution:** Fog nodes with deployed services must be widely distributed.
3. **Large-scale sensor networks:** Such networks are necessary to monitor the environment.
4. **Very large number of nodes:** Due to the wide geo-distribution fog computing requires a large number of nodes.
5. **Support for mobility:** The fog nodes must be capable of communicating directly with mobile devices, e.g. via the Locator/ID Separation Protocol (LISP).
6. **Real-time interactions:** Fog computing addresses real-time applications instead of applications with batch processing.
7. **Predominance of wireless access:** Fog nodes are well suited to communicate with wireless sensors in an IoT environment.

8. **Heterogeneity:** The concept of fog computing can be applied in a wide range of environments where the fog nodes cover different sectors. The runtime of each node can differ from each other.

9. **Interoperability and federation:** Fog nodes must be capable of supporting different data formats and exchanging it between different providers.

10. **Support for real-time analytics and interplay with the cloud:** On the one hand fog nodes must be able to communicate with end devices to provide low latency and on the other hand they have to be able to connect to the cloud in case the fog nodes must operate as middleware between end devices and cloud data centers.

Edge Computing (previously called fog computing) also provides the possibility to implement the three service models Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) as described in chapter 2.1.2 [IFB+17, p. 11].

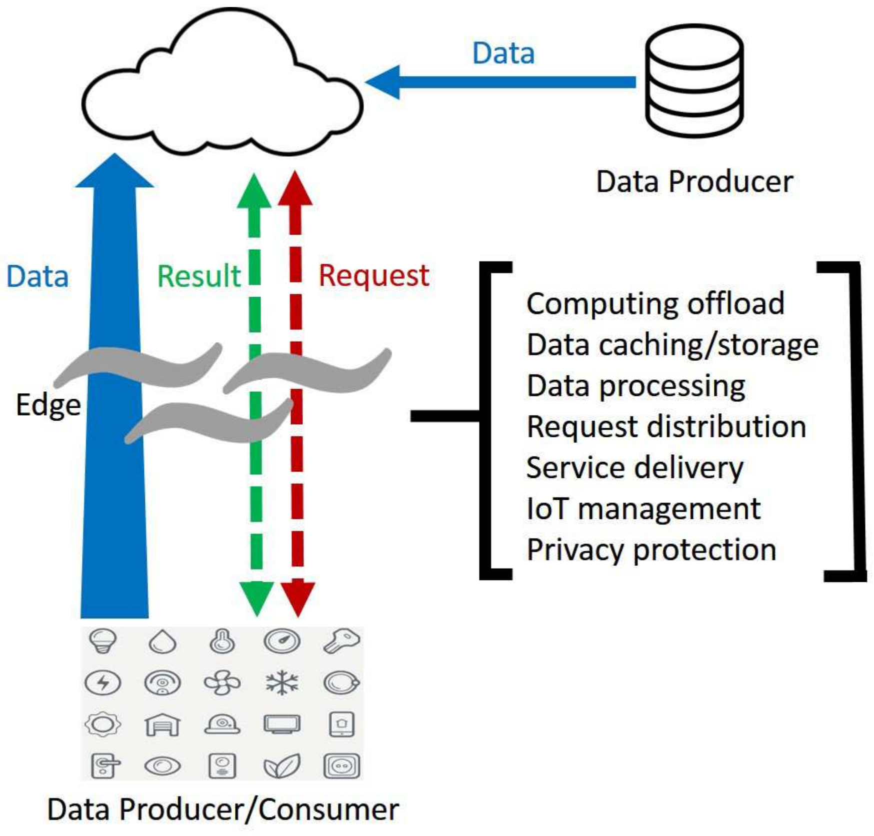

Figure 2.3: Edge Computing paradigm

Figure 2.3 shows that there are actors who are only producing data and pushing it into the cloud and on the other hand data consumers are requesting data based on their needs. In Edge Computing, data consumers can switch role and act as data producers too. Additionally, edge nodes do not only have the ability to request data from the cloud but can also perform computation offloading, data storage, data caching, data processing and distribute requests or response from or to the cloud [SCZ+16, p. 2]. This figure also underlines the statement of Iorga et al. [IFB+17, p. 9] that edge nodes are acting within a layer between the cloud and end devices.

It is also illustrated in figure 2.3 that Edge Computing tries to push business logic and processing power from the cloud out to proxy servers at the edge of the network [PT04, p. 1]. The edge itself (e.g. micro data center) is described by Shi et al.[SCZ+16, p. 2] as "any computing and network resources along the path between data sources and cloud data centers".

Wanner describes the following requirements for an Edge Computing environment [Wan16a]:

- The installation of the edge devices must be easy.
- The edge nodes must be easy to use.
- Little administration effort is needed.
- A remote access to the edge nodes is possible.
- Safety measures against system crash or data loss must be implemented.
- The connection to the edge nodes must be end-to-end encrypted [Rie18].
- The system has to be highly reliable [SCZ+16, p. 2].

To built a reliable system, Shi et al. outlines four basic features in terms of service management at the edge of the network namely Differentiation, Extensibility, Isolation and Reliability (DEIR) [SCZ+16, p. 7]:

1. **Differentiation:** The growing trend for IoT devices will result in multiple services that are deployed at edge nodes. These services must have different priorities that urgent tasks are executed first. E.g. health related services must be higher prioritized than services for entertainment.
2. **Extensibility:** The EdgeOS has to be capable of easily adding new things to an existing service or replacing nodes in case of defects or other reasons.
3. **Isolation:** There will be scenarios where multiple applications are deployed on the same edge node. These applications might share the same data resources. Therefore, the edge nodes must be well designed that crashing of one application

has no affect on another application. Additionally, a user's private data has to be protected from third party applications except the user actively grants permission to the application.

4. **Reliability:** Reliability must be guaranteed from different point of views. From the service point of view, to achieve reliability edge nodes should inform the user if a component has crashed are could crash in the near future. From the system point of view, it is necessary that all components in the network can communicate with each other to monitor the status. And from the data point view, it has to be made sure that data sensing and communication is reliable too.

An Edge Computing infrastructure is built by using the concept of peer-to-peer ad-hoc networking [Vel19]. A peer-to-peer network consists of devices that are all interconnected. Each device can take over tasks from the others because they share the same configuration and permissions [Rou05].

Within the edge network, edge devices act additionally as middleware and gateway. The nodes often provide several wired and wireless network interfaces to be able to communicate with different end devices. As sensors cannot store their generated data, it is immediately transferred to the edge nodes which further forward the data to other network components via protocols like Message Queuing Telemetry Transport (MQTT) or the Open Platform Communications Unified Architecture (OPC UA). During this process of transmitting data, edge nodes are also aggregating and pre-processing the generated data (e.g. remove redundant data). Edge nodes are not only able to process the data but are also capable to monitor the end devices and build analytics upon the monitored data [Rie18].

2.3 Mobile Computing

The final part of the introductory chapter covers necessary background information about the concept of Mobile Computing. This includes details about the evolution and provides a definition of this paradigm.

2.3.1 Evolution

The starting point of Mobile Computing can be traced back to the end of the 19th century. In 1894 Guglielmo Marconi (sometimes also called "the father of radio") was the first person who was able to produce and detect radio waves over long distances [JTZ$^+$14, p. 3]. This innovation made it possible to send the first distress signal using Morse Code in 1905 and enabled the start of consumer radio broadcasts for news and entertainment in 1919 [Liv13]. The next milestones in establishing a wireless mobile network for Mobile Computing was the introduction of the first wireless network called A-Netz and B-Netz, which was created in 1958 in Germany to make outgoing and receive incoming calls. But this network had still one limitation because the location of the mobile station must be known beforehand. Afterwards, the first commercial analog cellular mobile network called Advanced Mobile Phone Service (AMPS) has been established in Chicago in 1983, which allowed a maximum of approximately 800 simultaneous phone conversations per operator [Liv13] [Ano]. This technology allowed consumers to move around while using their mobile phones [Rou06].

While there have been made several improvements to the cellular mobile networks there are two important technologies that must be mentioned. On the one hand there was the invention of the Third Generation (3G) cellular radio transmission technology which focuses primary on high-data-rate communications with portable computers. And on the other hand the technology Wireless Application Protocol (WAP) has been developed. WAP is designed to reduce the number of bits of mobile applications transmitted through air by using compression techniques [Pie01, p. 13].

Beside the evolution of wireless mobile network technologies, the focus of the last five decades was also to develop faster, more reliable and cheaper electronic components. This trend helped to design smaller and more powerful mobile devices and thus changing the way of communication and computation [Sto02, p. 17]. The rapid expanding technologies in creating mobile devices lead to one major key event, which was the release of Apple's iPhone and its operating system iOS in 2007 [JTZ+14, p. 4]. With this event we entered the era of smartphones.

With the development of mobile devices like smartphones or Personal Digital Assistants (PDAs) and the emerging field of wireless technologies like cellular wireless networks, wireless LAN, Bluetooth, infrared, and wireless sensor networks, users have access to several IP-based applications anywhere and anytime [KL07, p. 413] [ZN10, p. 3]. The opportunities from these technologies lead to the rise of new paradigms called Mobile Computing or ubiquitous networking [Pie01, p. 1]. Since the release of the first iPhone there was an exponential growth in Mobile Computing and in October 2016 there were more users accessing the internet via mobile devices than desktop computers [JTZ+14, p. 4] [Lö17]. Mobile Computing is already used in the area of e-commerce, personal communications, telecommunications, monitoring remote or dangerous environments, national defense (monitoring troop movements), emergency and disaster operations, remote operations of appliances, and wireless Internet access [Sto02, p. XVII].

2.3.2 Paradigm

Livingston defines Mobile Computing – or sometimes also called Nomadic Computing – as "a technology that allows for the transmission of data, voice, and video via a computer or any other wireless enabled device without having to be connected to a fixed physical link" [Liv13]. The Information Systems Audit and Control Association (ISACA) has defined the following mobile devices in its white paper from 2010 [ISA10, p. 4]:

- Full-featured mobile phone with personal computer-like functionality, or "smart-phone"

- Laptop and netbook
- Personal Digital Assistant (PDA)
- Portable Universal Serial Bus (USB) device for storage and connectivity
- Digital camera
- Radio Frequency Identification (RFID) and mobile RFID (M-RFID) device for data storage identification and asset
- Infrared-enabled (IrDA) device such as printer and smart card

As it can be recognized from the list above, different types of mobile devices exist and many of them are used in everyday life. Summarizing all aspects of Mobile Computing and its devices, the following characteristics can be identified:

1. **Wireless communication:** All mobile devices are equipped with a wireless component to connect to existing wireless networks and communicate within them [IB93, p. 1].
2. **Mobility:** The devices used for Mobile Computing allow everybody to access data and information anywhere and anytime [Buc18].
3. **Small size:** Compared to desktop computers, mobile devices have to be much smaller in order to be portable and allow users to carry them around [Bra].
4. **Heterogeneity:** As it can be seen from the list of mobile devices above, Mobile Computing uses a variety of different devices. This means that the runtime or operating system of each device differs.
5. **Resource-constraint:** In order to build portable devices, they have to be smaller which can only be achieved by removing or shrinking computational components (e.g. CPU). Furthermore, the mobile devices have no permanent energy supply but rely on the power of their battery [Sto02, p. 173].

The concept of Mobile Computing can be split into two operational modes – connected and disconnected mode. The connected mode allows the mobile device to connect to a wireless or wired network to access online information from the network whereas in disconnected mode information is accessed locally like managing a schedule or address book [ZN10, p. 2]. Furthermore, there is another type of distinction based on mobile applications. On the on hand there exist horizontal applications, which are domain independent and where the device's location plays a significant role (e.g. local yellow pages with information of movies currently playing at the local theater). And on the other hand vertical applications have been developed which are written for a specific domain (e.g. field technicians who require access to manuals while on a repair assignment) [Pie01, p. 18].

In order to enable Mobile Computing the following generalized architecture has been established [IB93, p. 3].

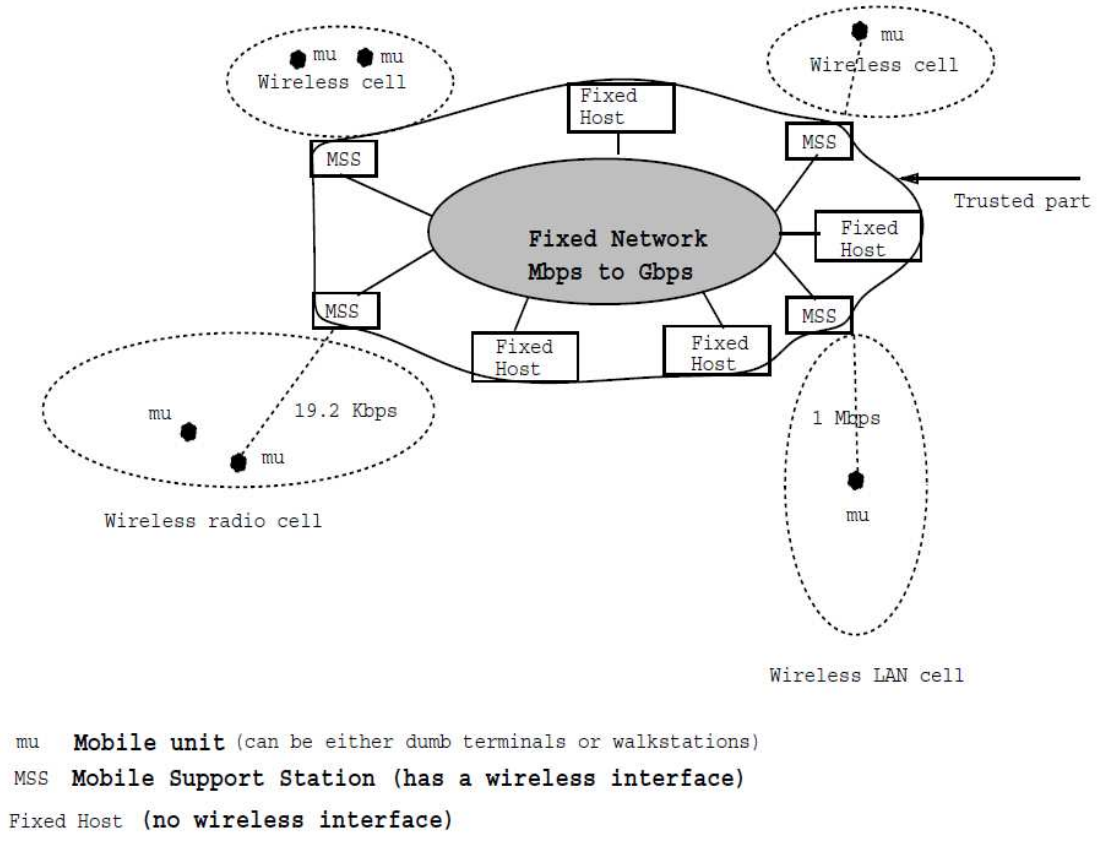

Figure 2.4: Mobile Computing architecture

As seen in figure 2.4 there exist mobile hosts as well as fixed hosts. The mobile hosts are represented by Mobile Units (MUs) operating in different wireless networks. Each of these networks is connected with a fixed host called Mobile Support Station (MSS). The Mobile Support Stations (MSSs) are equipped with a wireless interface to communicate with the MUs. Moreover, the MSSs have deployed application software which can be downloaded by mobile devices or used to execute operations remotely on the MSS. The architecture represented in figure 2.4 shows a two-tier structure. Firstly, there exists a powerful and reliable fixed network with MSSs and secondly, the two-tier structure consists of multiple often unreliable wireless networks containing MUs [IB93, pp. 3-4].

2.4 Task Offloading

2.4.1 Offloading Process

Task offloading in general – or often called computation offloading – is defined by Akherfi et al. as "the task of sending computation intensive application components to a remote server" [AGH18, p. 3]. During this process, mobile applications are partitioned into sub-components and these components are afterwards offloaded to remote target servers to get executed there. The main goal of task offloading is to overcome the hardware limitations of Smart Mobile Devices (SMDs) such as limited battery lifetime, limited processing power and limited storage capacity [AGH18, p. 1 + 3].

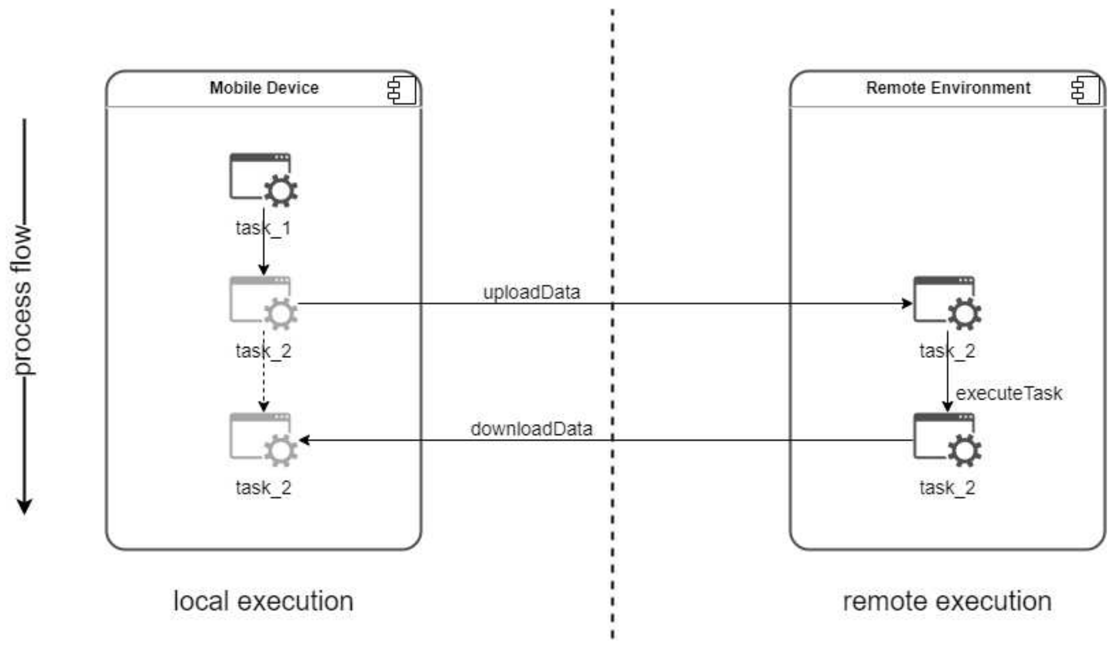

Figure 2.5: Task offloading process

Figure 2.5 illustrates the process flow of the task offloading procedure and where the execution takes place based on a very simple application, which consists of two tasks only. The process is initialized on the mobile device with the first task of a mobile application. If a task can be offloaded (like task_2) the task and all its data is sent to the remote environment to be executed there. The remote instance can either be hosted in the cloud or in an edge infrastructure. Once the execution has been finished on the remote instance the output data of the task is finally downloaded again to the mobile device and the offloading process terminates. The light-gray icons in 2.5 within the mobile device are used as placeholders to demonstrate that the task is currently executed but the execution is not happening locally.

2.4.2 Mobile Edge Cloud Offloading

The concept of Mobile Edge Cloud Offloading (MECO) has been introduced to overcome the limitations of Mobile Cloud Computing (MCC) – especially for latency sensitive applications. MECO makes use of aspects from Mobile Cloud Computing (MCC) and Mobile Edge Computing (MEC) and combines them for the offloading process.

The first concept – Mobile Cloud Computing – is defined by Alzahrani et al. as an infrastructure or platform where the storage and computing power used for mobile applications is transferred from mobile devices into centralized powerful computing platforms located in the cloud [AAS14, p. 2]. These centralized applications can be accessed via wireless connection based on a thin native client or web browser on the mobile devices. This concept has also gained attraction in recent years for companies to cut down development and operating costs of mobile applications [AAS14, p. 2]. Nowadays there exist already many applications that make use of Mobile Cloud Computing in the commerce, banking, learning or healthcare sector [AAS14, p. 2].

Secondly, the definition of Mobile Edge Computing (MEC) according to European Telecommunications Standards Institute (ETSI) is that "Mobile Edge Computing provides an IT service environment and cloud-computing capabilities at the edge of the mobile network, within the Radio Access Network (RAN) and in close proximity to mobile subscribers" [HPS+15, p. 4]. The concept of MEC is a big enabler for the evolution of 5G because it provides key aspects regarding technology and architecture. It helps to satisfy critical requirements in 5G networks like throughput, latency, scalability and automation. MEC is based on a virtualized platform which provides the possibility to run applications at the edge of the network [HPS+15, p. 4]. Hu et al. mentions several areas which can benefit from MEC which are augmented reality, intelligent video acceleration, connected cars and internet of things gateway [HPS+15, pp. 8-11].

As it can be seen from figure 2.6, MECO makes use of the three paradigms described in the chapters 2.1, 2.2 and 2.3 which are illustrated as three different layers. The bottom layer represents the concept of Mobile Computing. All different types of mobile devices like e.g. smartphones or tablets are located there. The middle or second layer makes use of the Edge Computing paradigm. It consists of several edge nodes or MEC servers which are deployed at Base Stations of the telecommunication network. Finally, the top layer in figure 2.6 represents the Cloud Computing concept. This layer is made up of central cloud data centers characterized by high computational power and large storage capacity.

In Mobile Edge Computing the mobile devices connect via the Radio Access Network with Base Stations (BSs) and in further consequence also with the MEC servers to offload tasks from mobile applications. Compared to MEC, Mobile Cloud Computing interacts directly with the cloud and offloads computations to centralized cloud data centers. The connection between the mobile devices and the cloud environment is established via the BSs of the RAN and afterwards via the core network (internet).

The three different layers of the MECO network topology shown in figure 2.6 also

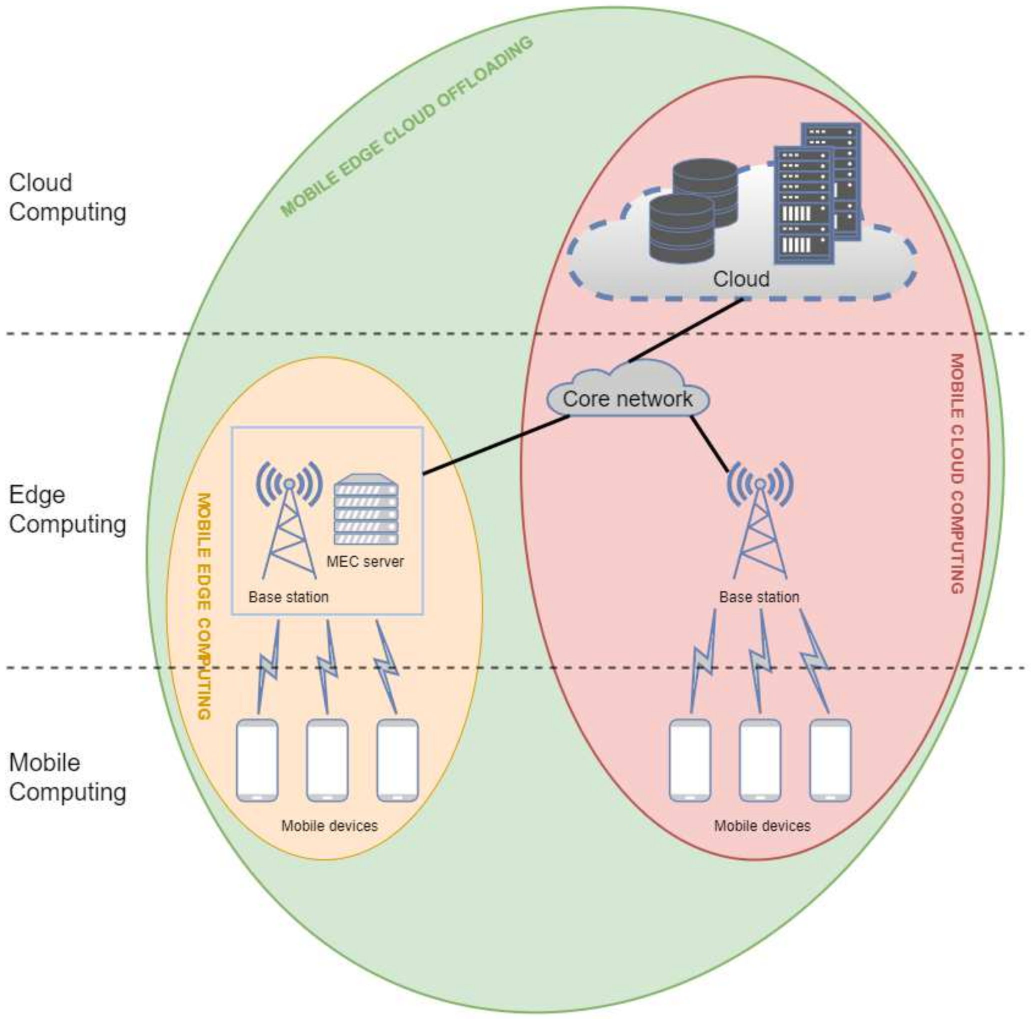

Figure 2.6: MECO network topology

illustrate how close each concept – Mobile, Edge or Cloud Computing – operates to mobile devices. The proximity to mobile devices is a critical factor regarding the reduction of latency times and to support near or real-time applications. MECO uses the concept of Mobile Computing where task offloading is not possible because the task relies on components of the device (e.g. camera). The concept of Edge Computing or Mobile Edge Computing is necessary to offload latency sensitive tasks, which is possible due to the short geographical distance between the edge nodes / MEC servers and the mobile node and higher bandwidth. Cloud Computing or Mobile Cloud Computing is mainly used as backup solution within the offloading process of latency sensitive applications because it cannot achieve acceptable latency times. The reason for this is on the one hand the long transmission distance and each packet has to bypass multiple network components within the core network.

CHAPTER 3

Related Work

This chapter covers related work for doing task offloading to either an edge computing environment only or in combination with the cloud. At the end of the chapter, the main aspects of the research papers are discussed and the lack of knowledge is pointed out.

3.1 Mobile Edge Cloud Offloading

Due to the emerging area of Internet of Things and the increasing demand in mobile applications, offloading tasks to the cloud has reached its limits. Therefore, the idea of offloading tasks to an Edge Computing environment has arisen and has attracted high attention in the past years in research. By now, there already exist a considerable number of scientific papers for doing mobile edge offloading.

For example, in the publication *Task Offloading for Mobile Edge Computing in Software Defined Ultra-Dense Network* by Chen and Hao [CH18], the two authors are presenting a Software Defined Task Offloading (SDTO) scheme for reducing the task duration (communication and computation time) but also considering the battery capacity on the mobile device within an ultra-dense network. A similar setup can be found in [TP18] but in addition these authors also take the prioritization of users into account, which is done by the providers based on the users' payments. In both publications, the network architecture consists of multiple small-cell Base Stations (BSs) – particularly with regard to the use of 5G network [GTM+16] – where each of them is equipped with edge nodes for performing remote computations. Therefore, both research papers focus on a scenario with a single user and multiple remote instances.

A different work published by You et al. is instead considering a mobile edge offloading system with multiple users and a single edge server [YHCK16]. In this paper, the authors are targeting the problem of energy-efficient resource allocation based on Time-Division Multiple Access (TDMA) and Orthogonal Frequency-Division Multiple Access (OFDMA).

Firstly, for TDMA they have studied the problem for minimizing the energy consumption on a mobile device by sharing a fixed offloading time-slot with multiple users. Secondly, for OFDMA You et al. have implemented an offloading algorithm in order to minimize the energy cost by dividing a fixed bandwidth into sub-channels and each sub-channel can only be assigned to one user. Their algorithm makes decisions based on an offloading priority function.

A similar problem is addressed in the publication by Chen et al. [CJLF15]. The authors are studying the challenge to achieve efficient wireless access coordination among multiple mobile devices within a mobile edge computing environment. The problem arises because Base Stations in the mobile network are configured to operate in a multi-channel setting and the data rates would be reduced if too many users are using the same wireless channel simultaneously. For solving this challenge, Chen et al. have used a game theoretic approach to model the distributed mechanisms and the have designed a distributed computation offloading algorithm. The algorithm is measuring wireless interference on different channels by sending a pilot signal to the Base Station. Afterwards, the different interference results are compared and the channel with the least power loss is chosen for offloading data.

The topic of minimizing the energy consumption of the mobile device based on TDMA is also covered in *Multiuser Resource Allocation for Mobile-Edge Computation Offloading* [YH16]. To solve this problem, You and Huang have designed an offloading algorithm which takes the channel gain and the local computing energy consumption into account. The proposed solution operates at the Base Station of the mobile network. The algorithm assigns priorities to each user and processes them in descending order until the resource capacities of the Base Station are fully occupied. Hereby, the BS also makes decisions about the size of data packages during the offloading process.

In comparison to [CH18], [TP18] and [YHCK16], Liu et al. proposes an offloading algorithm which uses queues on the mobile devices as well as on the edge nodes [LBP17]. Within their work, they have studied an Edge Computing architecture with multiple users and multiple servers, where they are on the one hand focusing on the trade-off between the UEs power consumption and the end-to-end delay. Moreover, on the other hand they are measuring the performance of the end-to-end delay based on the statistics of the queue length.

The paper *Mobility-Aware Caching and Computation Offloading in 5G Ultra-Dense Cellular Networks* published by Chen et al. deals with the impact of the user mobility on the performance of caching and computation offloading [CHQ+16]. They have identified that the most powerful caching mechanisms with respect to user mobility are Device-to-Device (D2D) and Small-cell Base Stations (SBS) caching because they are the least expensive ones. Regarding the offloading procedure, the authors are focusing on saving energy on the User Equipment (UE) but also considering user mobility.

A different topic is addressed within the publication *Latency Optimization for Resource Allocation in Mobile-Edge Computation Offloading* [RYCH18]. The authors' goal is to minimize the weighted-sum delay when offloading data from the mobile device to a base station at the edge of the network. Therefore, they have proposed three different models where data compression takes place: local compression, edge cloud compression and partial compression. They are also using a TDMA-based multi-user system like in the publication by You et al. [YHCK16]. Ren et al. have shown in their study that partial compression offloading can achieve the best performance because it jointly utilizes the communication and computation resources [RYCH18].

Zhao et al. [ZZGN17] are studying in their work the scheduling of tasks into a heterogeneous Edge Cloud Computing environment and how the computational resources of the edge nodes are allocated to the mobile users. Hereby, each task has bounded delay requirements. The task is successfully processed if the total offloading time lies within the boundaries, otherwise the task fails. The aim of their work is to maximize the probability that the defined delay bounds are satisfied. The simulations have shown that the offloading policy to edge nodes combined with cloud nodes can increase the success probability of satisfying the tasks' delay bounds by up to 40% compared to the usage of an Edge Computing environment only.

In the research [TOD+17], the authors are also dealing with fixed boundaries. Compared to [ZZGN17], Tao et al. are addressing the problem of minimizing the energy consumption with constraints on resource requirements and delay bounds [TOD+17]. The proposed offloading solution makes trade-offs between the energy consumption and completion performance, whereas the completion time takes precedence. For each offloading decision, the algorithm chooses whether to process the task locally or transmit it to an edge server

In many publications like the ones by [CH18], [TP18] and [LBP17], the proposed offloading solution is operating on task level, where each task can be either executed locally or remotely on an edge node. But there exist also approaches for offloading the whole application like presented in the scientific papers by Pan et al. [PMRT16] or Mach and Becvar [MB17]. Furthermore, the offloading algorithms *MAUI* by Cuervo et al. [CBC+10] and *CloneCloud* by Chun et al. [CIM+11] are instead operating on a more granular level, where each individual method can be offloaded to remote instances.

Another aspect where the individual solutions differ is the topic where the offloading decisions take place. In the work by Chen and Hao, the resource allocation and scheduling of tasks is done by a centralized network component which is also collecting and maintaining information of the mobile users and BSs [CH18]. A similar design decision has been made by You et al, where the Base Station itself is acting as the central point of coordination [YHCK16]. Also the authors in the publications *Latency Optimization for Resource Allocation in Mobile-Edge Computation Offloading* and *Multiuser Resource Allocation for Mobile-Edge Computation Offloading* are assuming that the edge environment has perfect knowledge about the network and its channel gains [RYCH18] [YH16]. In comparison to that, Tran and Pompili's solution makes the offloading decisions on the mobile device [TP18]. The decentralized approach for deciding locally at the User Equipment is also followed by Liu et al. where each UE has no information about other UEs [LBP17].

When talking about the offloading environment, [CH18] and [TP18] have built their offloading algorithm to be only performed between the end user and the edge environment but tasks are never transmitted to the public cloud or forwarded to other instances. In contrast to them, Chen et al. proposes an algorithm which allows computation offloading also to other mobile devices in addition to edge servers [CHQ+16] which is done by Device-to-Device communications among the users. In the publication *First Hop Mobile Offloading of DAG Computations* by De Maio and Brandic [DMB18, p. 1], the authors came up with a new approach called MECO where the offloading algorithm decides whether to offload the upcoming tasks to remote instances in either the Edge or Cloud Computing environment. Zhao et al. follows an identical approach, where the task can be processed on the one hand at the edge cloud or on the other hand on the remote cloud depending on the delay bounds [ZZGN17].

The experimental results presented in the publication by Chen and Hao [CH18] validate that the proposed Software Defined Task Offloading (SDTO) algorithm can reduce 20% of the task duration and save 30% of the energy cost as compared to random and uniform task computation offloading. Numerical results by Liu et al. have shown that it is possible to achieve a better power-delay trade-off for intense computation requirements and higher task arrival rates [LBP17]. Furthermore, it has been shown that the MECO algorithm in [DMB18] can reduce both application runtime and energy cost by up to 70%. Similar results have been presented in the publication by Chen et al. [CJLF15], where the experiments have shown that the presented distributed computation offloading algorithm can achieve a performance improvement by up to 68% compared to local computing.

3.2 Discussion

The given related work show that the offloading algorithms are designed for reducing energy cost reduction, application runtime or data transmission time for mobile applications. The provided solutions are developed to either support single or multiple target servers for offloading components on different levels of granularity – application, task or method. But according to the best of our knowledge, there exists no offloading algorithm which operates on the mobile device and makes decisions based on predefined priority levels in combination with the task's resource requirements, which is considered in this work. Furthermore, the mentioned solutions do not take task migrations between the remote instances into account to free-up resources and the fact of user mobility is often omitted. Our approach will also be designed to work within a multi-server edge cloud environment and the main aim will be the reduction of network and computation latency.

CHAPTER 4

Cloud and Edge Computing for Task Offloading

This part of the diploma thesis deals with benefits and limitations of Cloud and Edge Computing. The primary information about benefits and limitations has been extracted from existing research by conducting a Systematic Mapping Study. Afterwards, the relevant data has been correlated to the concept of task offloading to provide an overview about how one's benefits can help to overcome the other's limitations when doing task offloading from mobile devices to cloud or edge nodes.

4.1 The Systematic Mapping Process

To find relevant research about benefits and limitations of Cloud and Edge Computing in a structured and reproducible way, the following systematic mapping process by Peterson et al. has been applied in this thesis [PFMM08, p. 2].

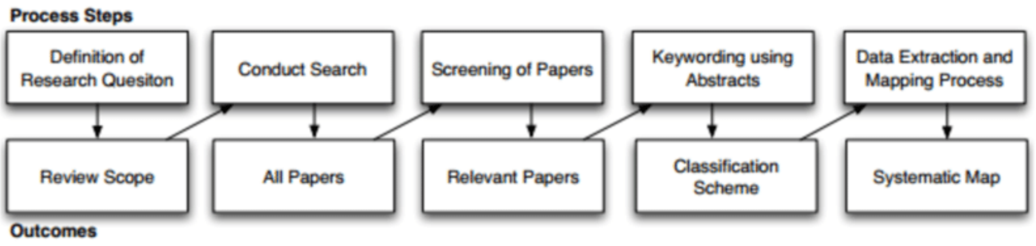

Figure 4.1: Systematic mapping process by Peterson et al.

Figure 4.1 shows all five process steps with their respective outcomes – starting with the definition of the research question and closing with the systematic map.

1. Definition of Research Question

This systematic mapping study is conducted to provide primary information about benefits and limitations of Edge and Cloud Computing for answering the following research question.

What benefits and limitations exist in Edge and Cloud Computing for doing task offloading?

2. Conduct Search

The search has been executed in the these online databases: ResearchGate, IEEE (IEEE Xplore), ACM (ACM Digital Library) and Google.
To find existing research, the following generalized search terms have been defined (the exact search terms for each database can be found in the Appendix A):

- **Cloud Computing:** ("cloud computing" AND ("benefits" OR "advantages" OR "pros" OR "limitations" OR "disadvantages" OR "cons" OR "drawbacks")) OR "Cloud- versus Edge-Computing" OR "Edge computing - Vision and Challenges"
- **Edge Computing:** ("edge computing" AND ("benefits" OR "advantages" OR "pros" OR "limitations" OR "disadvantages" OR "cons" OR "drawbacks")) OR "Cloud- versus Edge-Computing" OR "Edge computing - Vision and Challenges"

3. Screening of Papers

In this step, inclusion and exclusion criteria have been defined in order to eliminate primary search results that do not match, so that the relevant research remains. The exact remaining results after applying the inclusion and exclusion criteria can be found in the Appendix A.

- **Inclusion criteria**
  - books, papers, technical reports and grey literature ("Grey literature stands for manifold document types produced on all levels of government, academics, business and industry in print and electronic formats that are protected by intellectual property rights, of sufficient quality to be collected and preserved by libraries and institutional repositories, but not controlled by commercial publishers; i.e. where publishing is not the primary activity of the producing body." [Duk19]) describing empirical findings about Cloud/Edge Computing benefits/limitations in general or Mobile Cloud/Edge computing
  - research must be written in English or German
  - when there is the same research published in different journals, the most recent one is used
- **Exclusion criteria**
  - research that is not publicly available
  - research that is only available in the form of abstracts
  - studies that do not describe Cloud/Edge Computing benefits but only state that you can benefit from using it
  - studies that do not describe Cloud/Edge Computing limitations but only state that there exist some
  - no abstract available

4. Classification Scheme

By scanning the remaining research the following relevant keywords for the classification scheme have been identified:

- Bandwidth/latency
- Computational power
- Energy consumption
- Geographic context
- IT cost reduction
- Legal/rights
- Mobility
- Resiliency/availability
- Scalability

5. Data Extraction and Mapping Process

In the last step of the systematic mapping process, the remaining studies have been read and all relevant information has been extracted and categorized based on the predefined keywords from step 4. The exact mapping result can be found in the Appendix A.

4.2 Benefits and Limitations of Cloud and Edge Computing

The results of the systematic mapping study described in chapter 4.1 are listed in this part of the thesis focusing on how beneficial or limited both the paradigms Cloud and Edge Computing are with respect to task offloading. It starts with a detailed description of all benefits and limitations for both paradigms and closes with an overview about the findings.

4.2.1 Cloud Computing Benefits

This chapters gives detailed information about the identified benefits of Cloud Computing.

Computational power

The first advantage when using Cloud Computing is computational power. As the cloud consists of multiple data centers full of computers, the processing power of the cloud is almost limitless [Zor19]. This fact helps consumers of Cloud Computing services to reduce processing and waiting time because the cloud enables the automation of tasks and can handle tasks in parallel [MHS15, p. 13].

Energy consumption

Kumar and Lu [KL10, p. 3] have shown in their paper *CLOUD COMPUTING FOR MOBILE USERS: CAN OFFLOADING COMPUTATION SAVE ENERGY?* with experiments for two applications – chess game and image retrieval – that offloading the application's tasks to a cloud environment can save energy on mobile devices when enough bandwidth exists.

IT cost reduction

With conventional hosting it is needed to predict possible loads and to provide enough computing resources beforehand. Otherwise potential customers are forever gone because the service is not available due to overloading. This may lead to the problem that most of the time most of the computing resources are idle, which results in unnecessary operating costs [AFG^{+}10, p. 1]. By shifting the responsibilities of operating and maintaining the infrastructure to cloud providers, cloud computing enables organizations to reduce power, cooling, storage and space usage. It releases existing resources and budget which can be allocated to business strategic tasks

[CVdMK11, p. 3]. This economic benefit when preferring cloud computing over conventional hosting is often described as "converting capital expenses to operating expenses (CapEx to OpEx)" or "pay as you go" [AFG$^+$10, pp. 3-4]. Furthermore, Cloud Computing also helps to save costs for executing batch processes because the same operations can be done faster but at the same cost. This concept is called "cost associativity" [MHS15, p. 12].

Mobility

Mobility in this context is defined as the ability to have access to the cloud resources everywhere and at any time [AK15, p. 9]. The only requirement of the mobile device is the possibility to connect to the internet [Has13, p. 2].

Resiliency/availability

Furthermore, organization can become more competitive by providing highly reliable applications, because the cloud infrastructure is based on a very strong architecture and the vendors need to take care of the availability of their infrastructure [CVdMK11, p. 3] [AAS14, p. 3] [Mic15].

Scalability

With scalability, Cloud Computing offers consumers to easily add and remove Virtual Machines (VMs) with CPU, memory, storage, network and other resources [Set16, p. 8]. Hereby, Cloud Computing provides the possibility to scale-up and scale-out on demand. Scale-up or vertical scaling means adding more resources (e.g. memory or storage) to the existing hardware whereas scale-out or horizontal scaling describes the concept of adding additional hardware to the infrastructure [Sch18]. Therefore, it helps organizations to adapt their resources quickly to cover the current demand [AAS14, p. 3].

4.2.2 Cloud Computing Limitations

Cloud Computing does not only offers benefits but there also exist some limitations which are described below.

Bandwidth/latency

One problem that may arise is the network delay for offloading data to the cloud. NCTA – The Internet & Television Association – has published a study of Cisco which predicts that the number of connected devices will exceed 50 billion by 2020 [NCT]. According to Mohan and Kangasharju [MK16, p. 1] this leads to network congestion due to the large amount of data from IoT generators and limited bandwidth. Another aspect which is closely related to bandwidth is latency. As the cloud data centers are spread over the whole globe, the physical distance between the cloud and the end user increases and consequentially the transmission latency, which leads to the main problem for real-time applications [SNKS19, p. 2] [KQA14, p. 4].

Geographic context

As already mentioned above, the physical resources of the cloud are spread over multiple locations, which is per design the distributed nature of the cloud [KQA14, p. 4]. Due to this fact the central cloud data centers are often too far away from the end user and therefore, there is no possibility to provide geographic context for further data processing.

Legal/rights

With the launch of the General Data Protection Regulation (GDPR) in the European Union (EU) cloud consumers - especially for services outside the EU - have been facing legal challenges about the lack of knowledge where personal data is stored and who has access to the data. Especially questions like "Who is the owner of the data?" or "How can this data be used by third parties?" have been arisen [Wan16b]. Therefore, people are afraid of losing some rights and of being unable to protect user data because they do not know which jurisdiction has to be applied [AK15, p. 10] [BT15, p. 8].

4.2.3 Edge Computing Benefits

Within this part of the diploma thesis, all benefits of the Edge Computing paradigm are outlined in detail.

Bandwidth/latency

When using Edge Computing, a first benefit that arises is reduced network load. As the data processing and computing takes place close or at the edge of the network, the amount of data that must be transmitted to the cloud is reduced [MK16, p. 2]. By transferring data only within the edge environment it is possible to cut down latency times which is necessary for real-time applications [Wan16a]. The required low latency times can be achieved by high Local Area Network (LAN) bandwidth, decreased number of hops and short transmission distance [BOE17, p. 7] [SCZ$^+$16, p. 8]. In further consequence, the short transmission path also increases the transmission reliability [SCZ$^+$16, p. 8]. Another positive consequence of reducing the amount of data transmission from end device to the cloud is that the limited bandwidth of the internet connection and the core network is saved [SCZ$^+$16, p. 9] [BOE17, p. 7].

Energy consumption

Based on the research of Shi et al. it can be seen that the energy consumption on mobile devices can be reduced by $30 - 40\%$ by task offloading into an Edge

Computing environment [SCZ$^+$16, p. 3]. Additionally, in the publication *How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions* Baktir et al. states that offloading of tasks to edge servers consumes less energy compared to offloading of tasks to cloud servers [BOE17, p. 7].

Geographic context

Another advantage is that edge nodes provide contextual awareness. They can combine the generated data from sensors or other devices with location or application contexts to enrich the information [MK16, p. 2]. This is a big benefit for geographic based applications like transportation and utility management where data processing with geographic location information is possible without sending the data to the cloud [SCZ$^+$16, p. 4].

IT cost reduction

Edge Computing also helps to reduce costs for communication as the edge nodes do not necessarily need internet connection all the time [Wan16a].

Legal/rights

The concept of Edge Computing also supports enterprises to protect private and sensitive data against manipulation or theft because the data is not sent to the cloud and remains within the company's control [Wan16a]. This may increase the trust of customers into the companies and better fulfills the requirement for privacy protection [SCZ$^+$16, p. 2].

Scalability

Baktir et al. mentions that Edge Computing can enhance the scalability of Cloud Computing by replicating services and applications with the usage of VMs over several edge nodes [BOE17, p. 7].

4.2.4 Edge Computing Limitations

There are many benefits of Edge Computing but there can also be identified some drawbacks.

Computational power

As edge nodes in an Edge Computing environment are designed to be energy and cost saving, edge servers are not equipped with large storage or high computational power [Rie18].

Resiliency/availability

Edge servers are rarely high available [Rie18] and therefore Shi et al. states that reliability is a key challenge for Edge Computing. He mentions from the system point of view it might get very complex and challenging to maintain the network topology of the whole system because services are needed for failure protection and thing replacement in case of errors [SCZ+16, p. 2].

Scalability

Wanner argues that edge computing may be the wrong decision when a scalable solution is needed [Wan16a]. If there is a high fluctuation in demand for computing and storage, the ongoing maintenance and configuration effort may lead to high cost. He states too, that the architecture and specifications should be well designed beforehand because it does not provide the elasticity like cloud computing.

4.3 Overview of Cloud/Edge Computing Benefits and Limitations

This section provides an overview of all benefits and limitations of Cloud and Edge Computing mentioned in the chapters 4.2.1, 4.2.2, 4.2.3 and 4.2.4.

Topic	Cloud Computing	Edge Computing
Bandwidth/latency	–	+
Computational power	+	–
Energy consumption	∼	+
Geographic context	–	+
IT cost reduction	+	∼
Legal/rights	–	+
Mobility	+	–
Resiliency/availability	+	–
Scalability	+	∼

Table 4.1: Overview of how beneficial or limited Cloud and Edge Computing are for task offloading

Legend

+	... beneficial
∼	... partly beneficial (only under certain circumstances)
–	... limited

Table 4.1 shows that both concepts – Cloud as well as Edge Computing – have advantages for doing task offloading from mobile devices to cloud or edge nodes.

On the one hand, Cloud Computing benefits from its high computational power, IT cost reduction for consumers, mobility, resiliency/availability and scalability. With the usage of Cloud Computing and its high computational power, the possibility exists to offload compute-intensive tasks from the mobile device to a cloud data center to get lower execution times. Additionally, when combining the benefits of mobility, resiliency/availability and scalability, Cloud Computing can be used as a stable backbone or fallback mechanism for task offloading. Firstly, it can be accessed almost everywhere and at any time by just having an internet connection. Secondly, it is highly available and thirdly it enables easy and fast scalability when the whole demand cannot be processed on the existing Edge Computing architecture. Energy consumption in table 4.1 is only marked as partly beneficial because it can be reduced only under certain circumstances as mentioned in chapter 4.2.1.

On the other hand, the strengths of Edge Computing can be identified in lower latency and higher bandwidth, lower energy consumption of end devices, geographic context and controllability of legal/rights. The most important aspects when doing task offloading – especially for near or real-time applications – are low latency times and high bandwidth. Both benefits are achieved due to the proximity of edge servers to the end devices and are essential to keep the response times as low as possible. Another big focus of task offloading is the reduction of energy consumption on mobile devices in order to enhance the battery life. As shown in table 4.1 this goal can be reached by applying a task offloading concept in an Edge Computing environment. The proximity of edge nodes to end devices additionally provides the possibility to use location-aware information when processing offloaded tasks. Finally, the last big benefit can be identified in legal aspects and rights because the data on the edge servers is processed locally and is not potentially sent around the whole globe. Two more aspects are categorized as partly beneficial. The first one – IT cost reduction – can help to reduce communication costs via internet because the data is only sent to edge servers and not to the cloud. But there will rise again some hardware and maintenance costs even if they might be much lower than the costs for operating on-premise data centers. The second partial benefit is scalability because it can help to enhance the scalability of Cloud Computing due to literature but Cloud Computing is the preferred way when dealing with fluctuating demands.

Summarizing all mentioned benefits and limitations listed in table 4.1 it can be clearly identified that Cloud and Edge Computing are complementary paradigms when task offloading is required. For each limitation of one paradigm it can be seen that the other paradigm can help to overcome the problem. Therefore, both concepts can be perfectly combined to implement a task offloading concept to fulfill the requirement of low latency times for latency sensitive tasks but also have a stable fallback technology which can step in if necessary.

CHAPTER 5

Implementation of the PBMECO Algorithm

In this chapter, detailed information about the implementation of the Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm is covered. The chapter starts by describing the task prioritization concept, which is used by our offloading algorithm to differentiate the tasks by priorities. The next part includes information about the concept for handling task dependencies which may exist in mobile applications. The last section of this chapter includes a detailed technical description about the working procedure for selecting nodes and offloading tasks.

5.1 Concept for Task Prioritization

The PBMECO algorithm is a priority based algorithm, which means it makes decisions based on different latency priorities. Those priorities have influence on how the individual tasks of an application are treated during the offloading process.

The prioritization technique is based on a technique from requirements management, which is called "Numerical Assignment (Grouping)". This prioritization technique is most common and suggested in RFC 2119 and IEEE Standard 830-1998 for grouping requirements [BA05, p. 76]. The idea of this method is to assign each object a pre-defined priority group. For priority groups it can either be used labels (e.g. "high", "medium" or "low") or just numbers from 1 to 3 [Ves10, p. 2]. To avoid misunderstandings of what the different priority groups mean, a clear definition must be set. The number of distinct priority groups can be individually set, but in practice, the selection of three groups is most common [BA05, p. 76]. As this technique is widely adopted in requirements management and the tasks of an application can be treated in a similar way like requirements of a project, this method is also applied in the context of prioritizing tasks of applications for the offloading process of the PBMECO algorithm.

Due to Tsiatsis et al. the total latency time can be split into computation and network latency. [TKS05, p. 7]. Additionally, Toczé et al. defines six workload characteristics in edge computing: computation demand, communication demand, storage demand, deadline, arrival type and interarrival time [TSB+19, p. 2]. As there exists an overlap between network/computation latency and communication/computation demand, the algorithm takes these two workload characteristics into account to allow prioritization of network and computation latency for each task. This two types are further referenced as "communication latency" and "computation latency".

The prioritization concept has defined the three labeled ordinal priority groups "HIGH", "MEDIUM" and "LOW" for both types of latency. These three latency priority groups are described in detail below:

- **HIGH**
  The priority group "HIGH" represents the highest level. This level has to be used for very latency sensitive tasks because this tasks take precedence over all other latency priorities. In case this latency priority is applied as communication latency, the PBMECO algorithm tries to find the closest edge node or otherwise – which means it is set as computation latency – it looks for the most powerful node in the Edge Computing environment. In both scenarios, the best suitable edge node is chosen that has enough resources available to execute the task. Hereby, the algorithm also searches for already offloaded tasks of lower priorities and checks if a migration of those tasks is possible. The details on how the migration process is implemented can be found in chapter 5.4.6. Therefore, the latency priority HIGH should be chosen deliberately. Toczé et al. mentions, that reasons for setting the communication latency to HIGH are if the demand requires high bandwidth or the service needs to interact with other services. The computation latency instead should be set to HIGH if long execution runtime will probably result in a degradation or an error [TSB+19, p. 2].

- **MEDIUM**
  "MEDIUM" prioritized tasks are offloaded by the PBMECO algorithm to an edge node with enough storage and computational power. This edge node is probably not the closest or most power full one – depending on the latency type – because it maybe reserved for tasks with latency priority HIGH. Thus, this priority group should be applied to tasks which can be offloaded to edge nodes but higher latency times are acceptable and not mission critical.

- **LOW**
  The latency priority "LOW" specifies the lowest or least important level. When the PBMECO algorithm is applied, all tasks with this priority level are treated with least priority regarding communication or computation latency during the offloading process. Therefore, such tasks may be offloaded either to edge nodes or the cloud, depending on the resources reserved on edge nodes for higher priority

levels. If it does not matter where the task is offloaded and latency times are not important, this priority group should be used. This prioritization level is also defined as the standard level because it should not block resources if no conscious assignment has been made.

When the PBMECO algorithm is applied for task offloading on a mobile device, each task has to be applied one of these priority groups to both communication and computation latency. If no priority group is set, LOW is taken by the algorithm as default setting. The task prioritization concept of the PBMECO algorithm is designed this way because it is assumed that the developers of applications know their implementations and the environment best. Thus the developers should have the full control on how to prioritize the different tasks of an application within a given environment.

The technical implementation of the handling of the latency priorities is explained in further detail in the upcoming chapter 5.4.3.

5.2 Concept for Task Dependencies

In practice, each application on a mobile device can be split into multiple tasks, where each task has different execution responsibilities. For example, a face recognition application can be divided into the tasks "Image Acquisition, Face Detection, Pre-processing, Feature Extraction and Classification" [MYZ$^+$17, p. 2]. Usually, the individual tasks of an application have inter-dependencies and cannot be executed autonomously because upcoming tasks rely on output data from previous tasks. Therefore, it is necessary to apply a model in the PBMECO algorithm which can deal with inter-dependencies of tasks. This model is called task-call graph, which is defined as a Directed Acyclic Graph (DAG) [MYZ$^+$17, p. 6]. "In computer science and mathematics, a Directed Acyclic Graph (DAG) is a graph that is directed and without cycles connecting the other edges. This means that it is impossible to traverse the entire graph starting at one edge. The edges of the directed graph only go one way. The graph is a topological sorting, where each node is in a certain order" [Tec].

This definition perfectly matches the characteristics of an application because there exist one starting point and the individual tasks are executed in logical order. Therefore, each application is defined as a graph $G(V, E)$ [MYZ$^+$17, p. 6]. The vertices V define the set of tasks and the edges E represent the inter-dependencies between them. Each edge E is specified as pair (V_i, V_j), which means that there is a directed path from vertex V_i to the vertex V_j. According to Mao et al., there exist three common dependency models between tasks. The model can either include sequential, parallel or general dependencies [MYZ$^+$17, p. 6]. An example for each model can be seen in figure 5.1.

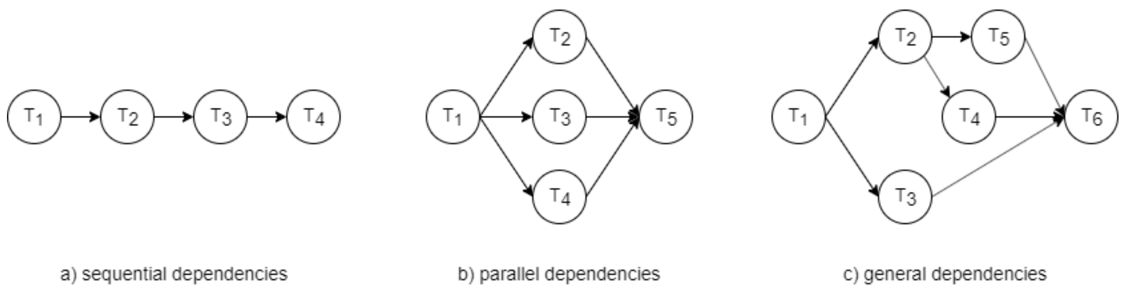

Figure 5.1: Task dependency models (adapted from Mao et al. [MYZ+17, p. 7]

Depending on the mobile application, one of these three dependency models can be applied. The PBMECO algorithm supports all three models and is therefore not limited to specific applications. Further information on how the dependencies are technically handled during the offloading process are described step by step in chapter 5.4.4.

5.3 Algorithmic Procedure

This section describes the procedure of the PBMECO algorithm on a high level to get an overview how the algorithm executes the task offloading process.

The algorithm is mainly split into two parts – one passive and one active part. On the one hand there is a procedure which is executed periodically in the background for scanning and tracking edge nodes in the close surroundings of the mobile node. On the other hand the algorithm performs the offloading procedure of tasks of an application on-demand, which means when there is something to offload. Both parts of the PBMECO algorithm are described below as pseudo code in general and on a high level perspective.

Algorithm 5.1: PBMECO generic edge nodes scanning task

```
Input: Edge nodes in environment
Output: Available edge nodes in scanning range sorted by distance to mobile
        node and computational power

1 PeriodicFunction begin
2     availableEdgeNodes ← scanEdgeNodes(edgeNodes)
      storeLocally(availableEdgeNodes)
3 end
```

The algorithm 5.1 shows the first part of the PBMECO algorithm which consists of only one task. This single task is executed periodically in the background and is responsible for scanning the surroundings of the current position of the mobile node to find available edge nodes, which are in predefined range. The periodic execution of the task is necessary due to possible movements and consequent position changes of the mobile node. Hereby, all available edge nodes are on the one hand sorted by distance between mobile node and edge node in ascending order and on the other hand by computational power in descending order. Afterwards, the sorted data is stored locally to enable quick access to the collected information. The details how the sorting and storage is performed are described in the upcoming chapters 5.4.1 and 5.4.2.

Algorithm 5.2: PBMECO generic offloading procedure "startDeployment"

Input: A task T, where each task is defined as $T_i(id, name, ram, cpu, storage, isOffloadable, mioInstructionsCount, communicationPriority, computationPriority, sequence)$

Output: A suitable node N where the task T_i is executed, either mobile N_{mobile} or edge N_{edge_j} or cloud N_{cloud} node

```
Function startDeployment($T_i$, $N_{mobile}$, $N_{cloud}$, $N_{edge}$) begin
    if preDependencyTasks $T_{i_{pre}}$ exist && $T_{i_{pre}}$ are offloaded then
        undeployTasks($T_{i_{pre}}$);
    end
    if $T_i$ is offloadable then
        $N_{edge_j} \leftarrow$ getAvailableEdgeNode($T_i$);
        suitableNode $\leftarrow$ decideOnNodes($T_i$, $N_{mobile}$, $N_{edge_j}$, $N_{cloud}$);
    else
        suitableNode $\leftarrow N_{mobile}$;
    end
    if suitableNode is found then
        deployTask($T_i$, suitableNode);
        return true;
    else
        return false;
    end
end
```

Algorithm 5.3: PBMECO generic offloading procedure "getAvailableEdgeNode"

```
Input: A task $T_i$
Output: A suitable edge node $N_{edge_j}$

Function getAvailableEdgeNode($T_i$) begin
    suitableEdgeNode ← null;
    searchPolicy ← evaluateSearchPolicy($T_i$);

    foreach getScannedEdgeNodes(searchPolicy) do
        if latencyPriority($T_i$) == HIGH then
            if hasEnoughResourcesWithMigration($N_{edge_j}$) then
                suitableEdgeNode ← $N_{edge_j}$;
                break;
            end
        else if latencyPriority($T_i$) == MEDIUM then
            if hasEnoughResources($N_{edge_j}$) && expectedRemainingRessources($N_{edge_j}$) ≥ knowledgebase.expectedRessources(latencyPriority HIGH) then
                suitableEdgeNode ← $N_{edge_j}$;
                break;
            end
        else
            if hasEnoughResources($N_{edge_j}$) && expectedRemainingRessources($N_{edge_j}$) ≥ knowledgebase.expectedRessources(latencyPriority HIGH AND MEDIUM) then
                suitableEdgeNode ← $N_{edge_j}$;
                break;
            end
        end
    end

    return suitableEdgeNode;
end
```

As shown in the pseudo code 5.2, the PBMECO offloading procedure receives as input a task T_i, which is characterized by the following parameters:

- id: is needed for technical differentiation of the tasks
- name: a human readable representation of the task
- RAM: represents the amount of RAM needed in MB

- CPU: specifies the number of processing units needed
- storage: defines the amount of storage needed in MB
- isOffloadable: boolean flag to specify if the task is offloadable to cloud or edge nodes
- mioInstructionsCount: represents the number of million instructions needed for task execution
- communicationPriority: is used to differentiate between HIGH, MEDIUM or LOW prioritized tasks for communication latency (see chapter 5.1 for more details)
- computationPriority: specifies the computation latency of the task, either HIGH, MEDIUM or LOW (see chapter 5.1 for more details)
- sequence: describes the execution order of the tasks

The offloading procedure is initialized by the function startDeployment(T_i, N_{mobile}, N_{cloud}, N_{edge}) and starts by checking if the task T_i has some pre-dependency-tasks $T_{i_{pre}}$. Pre-dependency-tasks represent tasks on which the execution of the current task T_i depends on and therefore the pre-dependency-task must be finished beforehand. If there exist one and they are currently offloaded to some cloud or edge node, they must be firstly undeployed from their nodes before the offloading procedure for the task T_i continues. In the next step a check regarding offloadability is performed. If the outcome of the check is negative – this means the task cannot be offloaded – the mobile node N_{mobile} is selected as suitable node for task deployment. Otherwise, the algorithm searches for an available edge node N_{edge_j} which matches the resource requirements of the task T_i by taking the latency priority – communication or computation latency – into account. To know whether to use the sorted list by distance or by computational power, the PBMECO algorithm has implemented a simple heuristic to determine the appropriate search policy. The search process is executed by the function getAvailableEdgeNode(T_i) described in general in 5.3. Afterwards, the function decideOnNodes(T_i, N_{mobile}, N_{edge_j}, N_{cloud}) checks if an edge node has been found beforehand. In case no edge node could be found, this function decides whether the task T_i should be deployed on the mobile N_{mobile} or cloud N_{cloud} node by calculating the total latency (network + computation latency). These two steps, represented in line 6 and 7 in the pseudo code 5.2 and in 5.3, are described in detail in the chapter 5.4.3. Finally, the PBMECO algorithm checks if a suitable node has been found. If yes, the task T_i is deployed on the selected node, which happens in line 12 in 5.2. If no compatible node could be found by the algorithm, the PBMECO algorithm returns false to provide the possibility for error handling. The detailed procedure about the function deployTask(T_i, suitableNode) is explained in chapter 5.4.4.

Once the PBMECO offloading procedure for the task T_i is finished, the task has been deployed on the best suitable node, which can be either the mobile N_{mobile}, edge N_{edge_j}

or cloud N_{cloud} node. Hereby, the main offloading target nodes are the edge nodes and the mobile node is chosen if the task is not offloadable. Finally, the cloud node instead acts as fallback instance, if no edge node is available or has not enough resources and the mobile node cannot handle the execution of the task T_i due to resource limitations.

The PBMECO algorithm is designed as an adapted First Fit (FF) decreasing heuristic, which is known from the bin packing problem. According to Skiena, analytical and empirical results suggest this method as the best and most intuitive heuristic. The general idea is to sort the objects in decreasing order of size and insert each object one by one into the first bin that fits. The implementation of a FF decreasing heuristic requires $O(n\ lgn)$ time [Ski98, p. 596]. In case of the PBMECO algorithm, the objects represent the tasks that have to be offloaded and the bins are the edge nodes. Hereby, not the objects are sorted but the edge nodes are on the one hand sorted by distance in ascending order, where the closest edge node N_{edge_j} is first and the farthest last, and on the other hand by computational power in descending order, which means the node with the highest computational power is first. Furthermore, the algorithm always stores the remaining resources of each N_{edge_j} locally, which is technically described in chapter 5.4.2. The FF heuristic is applied for finding a suitable edge node N_{edge_j} – an edge node which has enough resources available – where a task T_i can be offloaded to. The technical details on how this procedure is implemented can be found in chapter 5.4.3.

5.4 Technical Components of the Algorithm

The whole procedure of the Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm is split into the following six technical components, where each of the them has different responsibilities within the offloading process:

- Scanning component
- Storage component
- Offloading component
- Node Evaluation component
- Knowledge Base component
- Migration component

5.4.1 Scanning component

As described in the chapter 5.3 the PBMECO algorithm consists of two main parts. One main part is initialized by the function `scanEdgeNodes(edgeNodes)` of the scanning component, which is described in the pseudo code 5.1. The aim of this function is to discover the environment of the mobile node and to find potential edge nodes for offloading tasks. Once available edge nodes have been found, the scanning component calculates the geographical distance between the mobile node and each edge node. The distance calculation is done by applying the Haversine formula. The Haversine formula enables the calculation of the distance on the surface in kilometers based on the coordinates (latitude and longitude) between two different nodes [CN13, p. 2]. It is also an appropriate approach for numerical calculations at small distances [Ven19]. The Haversine formula is defined as follows [CN13, p. 2]:

$$d = 2r \sin^{-1}(\sqrt{\sin^2(\frac{\phi_2 - \phi_1}{2}) + \cos(\phi_1)\cos(\phi_2)\sin^2(\frac{\psi_2 - \psi_1}{2})}) \tag{5.1}$$

The formula uses the following variables:

- r ... radius of the earth in km, which is 6,371 km
- ϕ_i ... latitude of the coordinates of node N_i
- ψ_i ... longitude of the coordinates of node N_i
- d ... is the result representing the distance between the two points in kilometers

After the calculation of the distances between the mobile node and each edge node has finished, the PBMECO algorithm stores the information about the edge devices locally. This functionality enables the offloading algorithm to access the information quickly and it has not to scan the environment to find a suitable edge node every time on-demand once the offloading process is started. Finally, the scanning node initializes also the sorting of the edge nodes based on distance and computational power to find the closest or most powerful node during the offloading procedure first. The storage and the sorting itself is done in the storage component described in chapter 5.4.2.

The technical implementation of the main function scanEdgeNodes(edgeNodes) for initializing the scan of the mobile node's environment is defined in the listing 5.1 below.

```java
public void scanEdgeNodes(@NotNull List<EdgeNode> edgeNodes) {
  // get distance between mobile node and each edge node
  edgeNodes.forEach(edge -> {
    edge.setDistance(getDistanceToMobile(edge));
    storageComponent.updateAvailableNodes(edge);
  });

  // sort edge nodes
  storageComponent.sortNodesByDistance();
  storageComponent.sortNodesByPower();
}
```

Listing 5.1: Scanning component: function for scanning, storing and sorting of edge nodes

As mobile devices may change position, this action must be called periodically to keep the local information about available edge nodes up-to-date. Furthermore, not just the resource information must be updated but also the distances between the mobile node and the edge nodes have to be recalculated to always have quick access to the closest and most powerful node available.

5.4.2 Storage component

The storage component of the PBMECO algorithm handles the local storage of information regarding available edge nodes, which have been found by the scanning component described in chapter 5.4.1. The storing procedure is executed by the function updateAvailableNodes(edgeNode) described below.

```java
public void updateAvailableNodes(@NotNull EdgeNode node) {
  // check if distance exceeds max range and node already stored
      and no tasks are deployed on node -> remove node
  if (node.getDistance() > MAX_RANGE &&
      arrayIndicesOfNodesDistance.containsKey(node.getId()) && node
      .getTasksDeployed() == 0) {
    availableNodesDistance.remove(arrayIndicesOfNodesDistance.get(
        node.getId()).intValue());
  }
  // check if distance is in range and node already stored ->
      update distance
  else if (node.getDistance() <= MAX_RANGE &&
      arrayIndicesOfNodesDistance.containsKey(node.getId())) {
    availableNodesDistance.get(arrayIndicesOfNodesDistance.get(
        node.getId())).setDistance(node.getDistance());
  }
  // check if distance is in range and node is NOT already stored
  else if (node.getDistance() <= MAX_RANGE && !
      arrayIndicesOfNodesDistance.containsKey(node.getId())) {
    // create clone of original edge node because it is stored
        locally in algorithm, the original node object represents
        the real edge node
    availableNodesDistance.add(node.cloneCurrentNode());
  }

  availableNodesPower = new ArrayList<>(availableNodesDistance);
}
```

Listing 5.2: Storage component: local storage procedure

The local storage is built by using a combination of a HashMap and an Array for either the sorted list by distance and the sorted list by computational power. The HashMap is used to store a key-value pair consisting of the unique identifier of the edge node and the array index whereas the Array is needed to save the edge nodes. This structure is chosen to enable quick access to the objects. The retrieval of values from an Array by index and the retrieval of an object from the HashMap requires constant time $O(1)$ [Ano12]. Therefore, the PBMECO algorithm can achieve a time complexity of $O(n)$ when selecting any edge node N_{edge_j} from local storage.

The procedure implemented in 5.2 has to check three different scenarios. In scenario one starting in line 3, it checks if the edge node N_{edge_j} is out of range and it is stored locally without any tasks deployed. If this condition matches, N_{edge_j} is removed from local storage. The second condition starting in line 7 handles the scenario where N_{edge_j} is still in range and available in the local storage. In this case, the information about the distance between the mobile node and edge node is updated. Scenario three – the last scenario – which begins in line 11 covers the case where the edge node N_{edge_j} is in range but missing in local storage, therefore N_{edge_j} must be stored locally. For identifying the individual nodes, the PBMECO algorithm makes use of the concept Universally Unique Identifier (UUID). UUID has been established as a standard for generating unique identifiers by the Open Software Foundation. Based on the defined standard, they are considered to be unique worldwide [Aug19]. Therefore, they are best suited to technically identify edge nodes.

The two additional important functions of the storage component are sortNodesByDistance() and sortNodesByPower(), which actually execute either the sorting by distance or by computational power of the locally stored edge nodes.

```java
public void sortNodesByDistance() {
  // sort values
  availableNodes.sort(Comparator.comparingDouble(Node::getDistance
      ));

  // map Array indices to node IDs
  IntStream.range(0, availableNodes.size()).forEach(index ->
      arrayIndicesOfNodes.put(availableNodes.get(index).getId(),
      index));
}
```

Listing 5.3: Storage component: sorting procedure by distance

```java
public void sortNodesByPower() {
  // sort values
  Comparator<EdgeNode> edgeNodeComparator = Comparator.comparing(
      node -> node.getFreeCpu() * node.getFrequency());
  availableNodesPower.sort(edgeNodeComparator.reversed());

6    // map Array indices to node IDs
7    IntStream.range(0, availableNodesPower.size()).forEach(index ->
         arrayIndicesOfNodesPower.put(availableNodesPower.get(index).
         getId(), index));
8  }
```

Listing 5.4: Storage component: sorting procedure by computational power

The sorting of the edge nodes N_{edge} described in line 3 in 5.3 and line 4 in 5.4 is done by applying a stable, adaptive and iterative Mergesort, which can achieve far fewer time complexity than $O(nlog(n))$ when the array is partially sorted [Ora19]. The Mergesort algorithm implementation uses techniques from Peter McIlroy's *Optimistic Sorting and Information Theoretic Complexity* and is adapted from TimSort for Python by Tim Peter [Ora19] [McI93]. Finally, the HashMap of the local storage is updated to include the new mapping between the edge node identifier and the array index. This procedure is done in line 6 in 5.3 and line 7 in 5.4.

5.4.3 Node Evaluation component

The next technical component of the PBMECO algorithm, namely the node evaluation component, includes the functionality for finding a suitable target node. This node is used for offloading the task of the mobile application. The node evaluation component consists of two main functions, getAvailableEdgeNode(task) and decideOnNodes(task, mobileNode, edgeNode, cloudNode). The evaluation procedure for finding the best node – mobile N_{mobile}, edge N_{edge_j} or cloud N_{cloud} node – is initialized by the function getAvailableEdgeNode(task) defined in 5.5.

```
public EdgeNode getAvailableEdgeNode(Task task) {
  EdgeNode possibleNode = null;
  this.evaluatedSearchPolicy = evaluateSearchPolicy(task);

  switch (evaluatedSearchPolicy) {
    case DISTANCE_FIRST:
      for (int i = 0; i < storageComponent.
          getNumberOfAvailableNodesDistance(); i++) {
        if (i == 0) {
          possibleNode = storageComponent.getClosestNode();
        } else {
          possibleNode = storageComponent.getCeilingNodeDistance(
              possibleNode.getId());
        }

        // check if selected node has enough resources for task OR
            task has communication prio HIGH and migration of
            other task is possible
        boolean enoughResourcesAvail = hasNodeEnoughResources(task
            , possibleNode);
        if (enoughResourcesAvail || (task.getCommunicationPriority
            () == LatencyPriority.HIGH && migrationComponent.
            migrateTask(task, possibleNode, evaluatedSearchPolicy))
            ) {
          LOG.fine(String.format("Edge␣node␣found.␣Distance␣to␣
              node␣%s:␣%.2f␣km\n", possibleNode.getId(),
              possibleNode.getDistance()));
          return possibleNode;
        }
      }
      break;

    case POWER_FIRST:
      for (int i = 0; i < storageComponent.
          getNumberOfAvailableNodesPower(); i++) {
        if (i == 0) {
          possibleNode = storageComponent.getMostPowerfulNode();
        } else {
          possibleNode = storageComponent.getCeilingNodePower(
              possibleNode.getId());
        }

        // check if selected node has enough resources for task OR
            task has computation prio HIGH and migration of other
            task is possible
        boolean enoughResourcesAvail = hasNodeEnoughResources(task
            , possibleNode);
        if (enoughResourcesAvail || (task.getComputationPriority()
             == LatencyPriority.HIGH &&  migrationComponent.
            migrateTask(task, possibleNode, evaluatedSearchPolicy))
            ) {
          LOG.fine(String.format("Edge␣node␣found.␣Distance␣to␣
              node␣%s:␣%.2f␣km\n", possibleNode.getId(),
              possibleNode.getDistance()));
          return possibleNode;
        }
      }
      break;
  }
  return null;
}
```

Listing 5.5: Node evaluation component: initializing function to find best edge node for offloading

The function getAvailableEdgeNode(task) starts by evaluating the search policy for finding a suitable edge node N_{edge_j}. The search policy can either be "DISTANCE_FIRST" or "POWER_FIRST" and is identified by applying the Latency of Communication and Computation Heuristic (LACCH). "DISTANCE_FIRST" means that the algorithm starts looking for a suitable edge node based on the list sorted by distance. In case "POWER_FIRST" has been evaluated, the sorted list based on computational power is considered for the search process. The LACCH heuristic takes the task's specified priority for communication and computation latency into account and evaluates the matching search policy. This leads to the following three configuration setups:

1. **Communication latency = Computation latency**
   In this case, the PBMECO algorithm makes the decision for the search policy by identifying which latency type – communication or computation – based on historic values better fits the current task T_i. Therefore, it compares the characteristics RAM, CPU, storage and millions of instructions of T_i with the historic values from the knowledge base component (described in chapter 5.4.5) for both latency types. To make it comparable, it calculates the difference between the needed resources of the current task T_i and the historic value for each characteristic and each latency type (e.g. if T_i has set communication and computation latency to HIGH, the following calculations are done for RAM: $RAM_{communication} = RAM_i - RAM_{hist_{communicationHIGH}}$ and $RAM_{computation} = RAM_i - RAM_{hist_{computationHIGH}}$). If the total difference of all characteristics of type communication latency is smaller the algorithm chooses "DISTANCE_FIRST" as search policy, otherwise "POWER_FIRST" will be used.

2. **Communication latency > Computation latency**
   As the latency priorities defined in chapter 5.1 are comparable due its characteristic of being ordinal ($HIGH > MEDIUM > LOW$), we can evaluate this equation. If the communication latency of the task T_i is higher, the algorithm will use the search policy "DISTANCE_FIRST".

3. **Communication latency < Computation latency**
   The last possibility represents the reverse configuration of option two, where the communication latency is lower in comparison to the computation latency. Hereby, the evaluation will result in using "POWER_FIRST" as search policy.

After the evaluation of the search policy is done, the function getAvailableEdgeNode(task) iterates all available edge nodes – starting either with the closest or most powerful one depending on the search policy – stored in the storage component to find a suitable edge node N_{edge_j}. Hereby, each N_{edge_j} is evaluated if it has enough resources to handle the task T_i. The resources to check are RAM, CPU and disk storage of both N_{edge_j} represented by the variable $RESOURCE_{edge_j}$ and T_i represented by the variable $RESOURCE_i$. The resource evaluation procedure takes the latency priority groups described in chapter 5.1 into account and uses a cascading approach. Depending on the defined latency priority in T_i, the following three scenarios are possible:

- **Scenario 1: task priority HIGH**
  When the task T_i is of priority HIGH, the edge node N_{edge_j} is checked
  - $CPU_{edge_j} \geq CPU_i$,
  - $RAM_{edge_j} \geq RAM_i$ and
  - $STORAGE_{edge_j} \geq STORAGE_i$.

  If all three equations evaluate to `true`, the current edge node N_{edge_j} can be used for offloading the task.

- **Scenario 2: task priority MEDIUM**
  In case T_i is MEDIUM prioritized and the current edge node N_{edge_j} is the closest or most powerful node, it is initially calculated how much free resources on N_{edge_j} are available after an eventual deployment. The expected value is defined as $E(RESOURCE_{edge_j})$. Afterwards, the algorithm retrieves the average values of resources needed of historical HIGH priority tasks $RESOURCE_{hist_{HIGH}}$ from the knowledge base component. The historical data is used to predict how much resources may be needed for upcoming HIGH prioritized tasks and therefore the PBMECO algorithm reserves them on edge nodes. Then, the following equations are evaluated:
  - $E(CPU_{edge_j}) \geq CPU_{hist_{HIGH}}$
  - $E(RAM_{edge_j}) \geq RAM_{hist_{HIGH}}$
  - $E(STORAGE_{edge_j}) \geq STORAGE_{hist_{HIGH}}$

  These three equations must be `true` in order to choose N_{edge_j} as possible offloading target. If the current edge node N_{edge_j} does not represent the closest or most powerful node, the procedure is equal to the one described in "Scenario 1".

- **Scenario 3: task priority LOW**
  Similar to task priority MEDIUM, the algorithm initially calculates the expected resources $E(RESOURCE_{edge_j})$ and retrieves the historical data of HIGH priority tasks $RESOURCE_{hist_{HIGH}}$. In addition, it also retrieves historical values of MEDIUM prioritized tasks defined as $RESOURCE_{hist_{MEDIUM}}$ and it evaluates these equations:
  - $E(CPU_{edge_j}) \geq CPU_{hist_{HIGH}}$
  - $E(RAM_{edge_j}) \geq RAM_{hist_{HIGH}}$
  - $E(STORAGE_{edge_j}) \geq STORAGE_{hist_{HIGH}}$
  - $E(CPU_{edge_j}) \geq CPU_{hist_{MEDIUM}}$
  - $E(RAM_{edge_j}) \geq RAM_{hist_{MEDIUM}}$
  - $E(STORAGE_{edge_j}) \geq STORAGE_{hist_{MEDIUM}}$

If all six equations are true, the current edge node N_{edge_j} can be used for offloading the task T_i. If the current edge node N_{edge_j} does not represent the closest or most powerful node, the procedure is equal to the one described in "Scenario 1".

Once a possible edge node is found as offloading target, no further evaluations are done. Otherwise the node evaluation component is also responsible for deciding if the best alternative option is the mobile or cloud node. In this case, this component calculates the total latency time for both types of nodes and selects the node with the lowest time. The total latency time is obtained by summing up the the network and computation latency [TKS05, p. 7]. The network latency consists of two parts, namely propagation and serialization delay. Both delay times are defined in equations 5.2 and 5.3 [RF 12]:

$$Propagation\ delay = distance\ /\ speed \tag{5.2}$$

$$Serialization\ delay = packet\ size\ (bits)\ /\ transmission\ rate\ (bps) \tag{5.3}$$

The whole node decision procedure is implemented within the function decideOnNodes(task, mobileNode, edgeNode, cloudNode) and can be seen below.

```java
public Node decideOnNodes(Task task, MobileNode mobileNode,
    EdgeNode edgeNode, CloudNode cloudNode) {

  // find preferred node
  Node preferredNode = edgeNode;
  if (edgeNode == null && mobileNode != null && cloudNode != null)
      {
    // mobile node has no latency time because no network transfer
        needed
    double totalTimeMobileNode = Utils.calculateComputationLatency
        (task, mobileNode);
    double totalTimeCloudNode = Utils.calculateComputationLatency(
        task, cloudNode) + Utils.calculateCommunicationLatency(task
        , cloudNode, cloudNode.getDistance(), false, true);

    // check if mobile node has minimum total time and enough
        resources, otherwise deploy on cloud node
    if (totalTimeMobileNode < totalTimeCloudNode &&
        hasNodeEnoughResources(task, mobileNode)) {
      preferredNode = mobileNode;
    } else {
      preferredNode = cloudNode;
    }
  }

  ...

20    return preferredNode;
21  }
```

Listing 5.6: Node evaluation component: node decision procedure

The conditional block starting in line 5 in listing 5.6 is applied when no edge node has been found. In this case, it is checked if the mobile or cloud node has enough resources available (CPU, RAM and storage). If yes, the decision procedure takes the node – mobile or cloud node – with the lowest total latency time as preferred node $N_{preferred}$ as offloading target for the task T_i. This node is then used for initializing the deployment, which is handled by the offloading component described in the upcoming chapter 5.4.4.

5.4.4 Offloading component

Another technical component of the PBMECO algorithm is the offloading component. It is responsible for the whole deployment and undeployment process of a task T_i and handling task dependencies based on the concept described in chapter 5.2. This functionality is initialized by the function startDeployment(@NotNull Task task, Node mobileNode, Node cloudNode, List<Node> edgeNodes). The implementation of the previously mentioned function is described in the following listing.

```
public boolean startDeployment(@NotNull Task task, MobileNode
    mobileNode, List<CloudNode> cloudNodes, List<EdgeNode>
    edgeNodes) {
  // check for pre-dependency tasks that are already deployed on
      nodes
  task.getPreDependencyTasks().stream().filter((preDependentTask)
      -> preDependentTask.getDeployedNode() != null).forEach((
      preDependentTask) -> {
    LOG.fine(String.format("%s␣depends␣on␣deployed␣%s\n", task.
        getId(), preDependentTask.getId()));

    undeployTask(preDependentTask, preDependentTask.
        getDeployedNode(), false);
  });

  // set mobile node as default for deployment
  Node suitableNode = mobileNode;

  // check edge nodes only if task is offloadable
  if (task.isOffloadable()) {
    EdgeNode edgeNode = nodeEvaluationComponent.
        getAvailableEdgeNode(task);
    // use first cloud node because their exist only 1
    suitableNode = nodeEvaluationComponent.decideOnNodes(task,
        mobileNode, edgeNode, cloudNodes.get(0));
  }
18
19      // deploy task to a suitable node
20      if (suitableNode != null) {
21        deployTask(task, suitableNode, false);
22        LOG.fine(String.format("%s␣deployed␣on␣%s\n", task.getId(),
              suitableNode.getId()));
23        return true;
24      } else {
25        return false;
26      }
27    }
```

Listing 5.7: Offloading component: deployment initializing function

As defined in 5.7, the code block starting in line 3 until line 7 is responsible for handling task dependencies. Hereby, each task T_i is checked if there exist already deployed pre-dependency-tasks and the algorithm undeploys them if necessary, which is specified in line 6. Afterwards if the task is offloadable, the offloading component initializes the search process to find a suitable node described in line 14 and 16. Finally, the PBMECO algorithm performs the actual deployment on the selected node, which can be either the mobile, edge or cloud node.

In case T_i has been successfully deployed to an edge node, the local information of the specific edge node stored in the storage component is updated. Thereby, the algorithm refreshes the data about the remaining resources like CPU, RAM and storage. This step is necessary in order to have the correct information available when finding a suitable node for the next task T_{i+1}. As final steps, the offloading process also includes firstly the update of historical values in the knowledge base component and secondly the re-ordering of the locally stored edge nodes based on remaining computational power. Additionally, once the remote execution is finished, the offloading component is also responsible for undeploying the task's data from the remote node back to the mobile node. Hereby, it again updates the local resource information of the storage component. The technical functions used for both the deployment and undeployment process are deployTask(@NotNull Task task, @NotNull Node node, boolean isMigration) and undeployTask(@NotNull Task task, @NotNull Node node, boolean isMigration) in 5.8 and 5.9.

```
public void deployTask(@NotNull Task task, @NotNull Node node,
    boolean isMigration) {
  ...
  node.setFreeRam(node.getFreeRam() - task.getRamNeeded());
  node.setFreeCpu(node.getFreeCpu() - task.getCpuNeeded());
  node.setFreeStorage(node.getFreeStorage() - task.
      getStorageNeeded());
  task.setDeployedNode(node);
  ...

```

```java
  // only update knowledge base if task is offloadable and an edge
      node has been selected
  // re-sort locally stored edge nodes due to change in available
     resources
  if (task.isOffloadable() && node instanceof EdgeNode) {
    knowledgeBaseComponent.updateAvgValues(task);
    storageComponent.sortNodesByPower();
  }
}
```

Listing 5.8: Offloading component: task deployment

```java
public void undeployTask(@NotNull Task task, @NotNull Node node,
    boolean isMigration) {
  ...
  node.setFreeRam(node.getFreeRam() + task.getRamNeeded());
  node.setFreeCpu(node.getFreeCpu() + task.getCpuNeeded());
  node.setFreeStorage(node.getFreeStorage() + task.
     getStorageNeeded());
  task.setDeployedNode(null);
  ...

  // re-sort locally stored edge nodes due to change in available
     resources
  if (task.isOffloadable() && node instanceof EdgeNode) {
    storageComponent.sortNodesByPower();
  }
}
```

Listing 5.9: Offloading component: task undeployment

5.4.5 Knowledge base component

The knowledge base component of the PBMECO algorithm holds the historical information about CPU, RAM, storage and millions of instructions of already deployed tasks categorized by latency type and priority. The technical implementation on how the data is handled and aggregated can be seen in listing 5.10.

```java
public void updateAvgValues(@NotNull Task task) {
  LatencyPriority latencyPrioCommunication = task.
     getCommunicationPriority();
  LatencyPriority latencyPrioComputation = task.
     getComputationPriority();

  // calculate moving average
  // RAM
```

```
    priorityTaskMapAvgRam.get(communication).put(
        latencyPrioCommunication, calculateMovingAverage(
        latencyPrioCommunication, mapValuesRam.get(communication),
        mapSumRam.get(communication), mapIndexRam.get(communication),
          task.getRamNeeded()));
    priorityTaskMapAvgRam.get(computation).put(
        latencyPrioComputation, calculateMovingAverage(
        latencyPrioComputation, mapValuesRam.get(computation),
        mapSumRam.get(computation), mapIndexRam.get(computation),
        task.getRamNeeded()));
    // CPU
    priorityTaskMapAvgCpu.get(communication).put(
        latencyPrioCommunication, calculateMovingAverage(
        latencyPrioCommunication, mapValuesCpu.get(communication),
        mapSumCpu.get(communication), mapIndexCpu.get(communication),
          task.getCpuNeeded()));
    priorityTaskMapAvgCpu.get(computation).put(
        latencyPrioComputation, calculateMovingAverage(
        latencyPrioComputation, mapValuesCpu.get(computation),
        mapSumCpu.get(computation), mapIndexCpu.get(computation),
        task.getCpuNeeded()));
    // Storage
    priorityTaskMapAvgStorage.get(communication).put(
        latencyPrioCommunication, calculateMovingAverage(
        latencyPrioCommunication, mapValuesStorage.get(communication)
        , mapSumStorage.get(communication), mapIndexStorage.get(
        communication), task.getStorageNeeded()));
    priorityTaskMapAvgStorage.get(computation).put(
        latencyPrioComputation, calculateMovingAverage(
        latencyPrioComputation, mapValuesStorage.get(computation),
        mapSumStorage.get(computation), mapIndexStorage.get(
        computation), task.getStorageNeeded()));
    // Million instructions
    priorityTaskMapAvgMioInstructions.get(communication).put(
        latencyPrioCommunication, calculateMovingAverage(
        latencyPrioCommunication, mapValuesMioInstructions.get(
        communication), mapSumMioInstructions.get(communication),
        mapIndexMioInstructions.get(communication), task.
        getMioInstructionsCount()));
    priorityTaskMapAvgMioInstructions.get(computation).put(
        latencyPrioComputation, calculateMovingAverage(
        latencyPrioComputation, mapValuesMioInstructions.get(
        computation), mapSumMioInstructions.get(computation),
        mapIndexMioInstructions.get(computation), task.
        getMioInstructionsCount()));
}
```

Listing 5.10: Knowledge base component: historical task data aggregation functionality

The method updateAvgValues(@NotNull Task task) is responsible for calculating predictive values for each resource (CPU, RAM, storage or millions of instructions). Everytime a task T_i is deployed, this method is called to compute new forecast values based on historic data. It differentiates always between the specified latency priorities for each type – communication and computation latency – described in detail in chapter 5.1. The algorithm uses the forecast method "moving averages of order five". The formula of a moving average of order m is written as

$$\hat{T}_t = \frac{1}{m} \sum_{j=-k}^{k} y_{t+j} \tag{5.4}$$

where $m = 2k + 1$. Moving averages is one method for doing time series decomposition where the trend and cycle are combined into a single so-called trend-cycle component [HA18, p. 197]. This trend-cycle at time t is calculated by averaging observations of the time series within k periods of t [HA18, p. 203]. By applying the forecasting method moving averages, the PBMECO algorithm can follow trends of resource usages of the tasks of a mobile application. Therefore, it covers scenarios like one-time outliers or behavioral changes in application usage (e.g. the end user uses different mobile applications over time). The order of value "five" was chosen due to two reasons: 1) "In general, a larger order means a smoother curve" and 2) "Simple moving averages such as these are usually of an odd order that they are symmetric. In a moving average of order $m = 2k + 1$, the middle observation and the same amount of k values on either side are used for calculating the average" [HA18, p. 207]. Thus, the method moving averages of order five – also written as 5-MA – can help to accurately forecast the needed resources for an upcoming task T_{i+1} of a specific priority class.

5.4.6 Migration component

Finally, the PBMECO algorithm has implemented a migration component. This technical component kicks in once the algorithm has to find an edge node N_{edge_j} for offloading a task T_i of priority HIGH – either communication or computation latency – and there are currently not enough free resources (CPU, RAM or storage) on N_{edge_j}. The migration procedure is initialized by the function migrateTask(Task conflictingTask, @NotNull Node sourceNode, SearchPolicy searchPolicy) which is implemented as shown below.

```
public boolean migrateTask(Task conflictingTask, @NotNull Node
    sourceNode, SearchPolicy searchPolicy) {

  ...

  // find possible tasks to migrate based on the migration policy
  switch (migrationPolicy) {
    case LOW_ONLY:
        possibleTasksToMigrate = deployedTasks.stream().filter(
            deployedTask -> deployedTask.getCommunicationPriority()
            == LatencyPriority.LOW && deployedTask.
            getComputationPriority() == LatencyPriority.LOW).collect(
            Collectors.toList());
        break;
      case LOW_AND_MEDIUM:
        possibleTasksToMigrate = deployedTasks.stream().filter(
            deployedTask ->
          {
            return Arrays.asList(LatencyPriority.LOW,
                LatencyPriority.MEDIUM).contains(deployedTask.
                getCommunicationPriority()) ||
                Arrays.asList(LatencyPriority.LOW, LatencyPriority.
                    MEDIUM).contains(deployedTask.
                    getComputationPriority());
          }
        ).collect(Collectors.toList());
        break;
      default:
        throw new IllegalStateException("Unexpected␣value:␣" +
            migrationPolicy);
    }

    ...

    // get sum of deployed resources
    double ramToMigrate = possibleTasksToMigrate.stream().
        mapToDouble(Task::getRamNeeded).sum();
    double cpuToMigrate = possibleTasksToMigrate.stream().
        mapToDouble(Task::getCpuNeeded).sum();
    double storageToMigrate = possibleTasksToMigrate.stream().
        mapToDouble(Task::getStorageNeeded).sum();

    // check how much resources are free after possible migrations
    if (ramToMigrate + sourceNode.getFreeRam() < conflictingTask.
        getRamNeeded() || cpuToMigrate + sourceNode.getFreeCpu() <
        conflictingTask.getCpuNeeded() || storageToMigrate +
        sourceNode.getFreeStorage() < conflictingTask.
        getStorageNeeded()) {
      return false;
    }

    for (Task taskToMigrate : possibleTasksToMigrate) {
      // stop migration of additional tasks if there are enough
          resources available
      if (nodeEvaluationComponent.hasNodeEnoughResources(
          conflictingTask, sourceNode)) break;

```

```
38      // find suitable alternative edge node
39      Node currentNode = storageComponent.getCeilingNode(sourceNode.
            getId(), searchPolicy);
40      while (currentNode != null) {
41        if (nodeEvaluationComponent.hasNodeEnoughResources(
              taskToMigrate, currentNode)) {
42          deployTaskOnOtherNode(taskToMigrate, sourceNode,
                currentNode);
43          break;
44        }
45        currentNode = storageComponent.getCeilingNode(currentNode.
              getId(), searchPolicy);
46      }
47
48      // use cloud for tasks that could not be migrated to other
            edge nodes
49      if (currentNode == null) {
50        tasksToMigrateForCloud.add(taskToMigrate);
51      }
52    }
53
54    // migrate tasks to cloud node
55    for (Task taskToMigrate : tasksToMigrateForCloud) {
56      // stop migration of additional tasks if there are enough
            resources available
57      if (nodeEvaluationComponent.hasNodeEnoughResources(
            conflictingTask, sourceNode)) break;
58
59      deployTaskOnOtherNode(taskToMigrate, sourceNode, cloudNode);
60    }
61
62    return true;
63  }
```

Listing 5.11: Migration component: task migration procedure

As shown in listing 5.11 between line 6 and 20, the PBMECO algorithm has implemented two different migration policies. Depending on the configuration, the algorithm executes either

1. migration policy "LOW_ONLY": Hereby, the algorithm is looking for tasks with communication and computation latency priority LOW that are already deployed on the current node N_{edge_j} or

2. migration policy "LOW_AND_MEDIUM": Once this configuration is selected, the algorithm searches for already deployed tasks on N_{edge_j} of priority LOW or MEDIUM for both communication and computation latency.

If the algorithm finds tasks $T_{migrate}$ that can be migrated to other nodes, it calculates the amount of resources that can be made available after a potential migration. In case the computed free resources exceed the required resources of task T_i, the actual migration of the tasks is started. At the beginning of the migration process, the algorithm tries to find a new edge node, where the task $T_{migrate_k}$ can be transferred to. This procedure can be seen between line 39 and 46. Hereby, the algorithm iterates through the locally stored edge nodes of the storage component. If no alternative edge node can be found, the PBMECO algorithm uses the cloud instance as fallback mechanism. Migrations of tasks $T_{migrate}$ are done until enough resources are available on the current edge node N_{edge_j} to make the deployment of the new task T_i possible. Finally, the function `migrateTask(Task conflictingTask, @NotNull Node sourceNode, SearchPolicy searchPolicy)` returns `true` if the tasks have been successfully migrated to other nodes, which can be either an edge or a cloud node.

CHAPTER 6

Evaluation

The purpose of the experimental simulation is to evaluate the performance of the implemented Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm, which has been described in detail in chapter 5. Thus, this chapter contains information about the evaluation setup and the simulation process that is executed. It starts by explaining the method of Monte Carlo Simulation, continuing with the description of the simulation environment and process and finalizing with a discussion on the simulation results based on predefined scenarios.

6.1 Event-driven Monte Carlo Simulation

Before the term "Event-driven Monte Carlo Simulation" is explained in further detail, this section starts with a short introduction what simulation means in general. Banks et al. defines simulation as "the imitation of the operation of a real-world process or system over time" [BCINN14, p. 3]. This definition matches the one from Waldmann and Helm [WH16, p. 4], who additionally state that this method is needed to get insights on interesting metrics, where no analytical or efficient solution exists for the specified problem. Advantages of doing simulations are that they are easier to apply compared to analytical methods and provide higher flexibility in modeling and analyzing real-world systems [WH16, p. 4].

Based on this definition, we are now focusing on the method Monte Carlo Simulation. In general, the idea is to estimate the parameter μ based on the expected value $E(X)$ of a random variable X [WH16, p. 6]. The execution of a single simulation run leads to the problem that the deviation of $E(X)$ and hence μ is too high and therefore not a convincing result. But this dispersion can be reduced by doing repetitive simulation runs using independent random samples and calculating the arithmetic mean afterwards [WH16, p. 6]. Based on the theorem "Law of Large Numbers", which states that the arithmetic mean $\bar{X}$ for $\mu \to \infty$ converges to the expected value $E(X)$ and therefore also to the parameter μ. When doing a simulation, the arithmetic mean is called "Monte Carlo Estimator" and is defined in equation 6.1 [WH16, p. 8]:

$$I_{MC}(n) = \frac{1}{n} \sum_{k=1}^{n} X_k \tag{6.1}$$

To understand the concept of Monte Carlo Simulation there are three more terms that are explained below [WH16, p. 77]:

- System: is represented by a set of objects that have relations to each other
- State: includes the total amount of variables that are needed to describe the system and its relation at a specific point in time
- Model: a simplified projection of the real-world system or process

As there exist multiple variations of Monte Carlo Simulations, this thesis focuses only on applying an Event-driven Monte Carlo Simulation. This type of Monte Carlo Simulation is used if the state of the model is only changed when an event occurs. In this thesis, the method helps to simulate the expected communication and computation latency times for the different priority groups described in chapter 5.1. These times are calculated on the one hand by applying the Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm and on the other hand by using a random offloading procedure, which means the target edge nodes are selected randomly. Furthermore, the random offloading procedure does not differentiate between task priorities, makes no migrations and it takes the first node that matches the resource requirements. The detailed description about the simulation model is given in the upcoming chapter 6.2.

6.2 Simulation Model/Process

The simulation framework has been implemented in order to evaluate the performance of the Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm with respect to latency times as already mentioned before. Therefore, the simulation process illustrated in figure 6.1 has been designed in this thesis and is applied to all simulation scenarios specified in chapter 6.4.2.

The simulation process starts by sampling the position of the mobile node N_{mobile}. Hereby, the simulation framework calculates new coordinates of N_{mobile} on a random basis within a specified corridor. This step has been implemented in order to simulate possible movements of the mobile node because the mobile end devices usually not stick to one specific location. Afterwards, random samples of the resources CPU, RAM, storage and millions of instructions of all tasks of the mobile application are computed. This helps to cover volatile resource requirements due to different usages of the mobile application. Therefore, we can simulate the real-world behavior more accurately and can achieve more

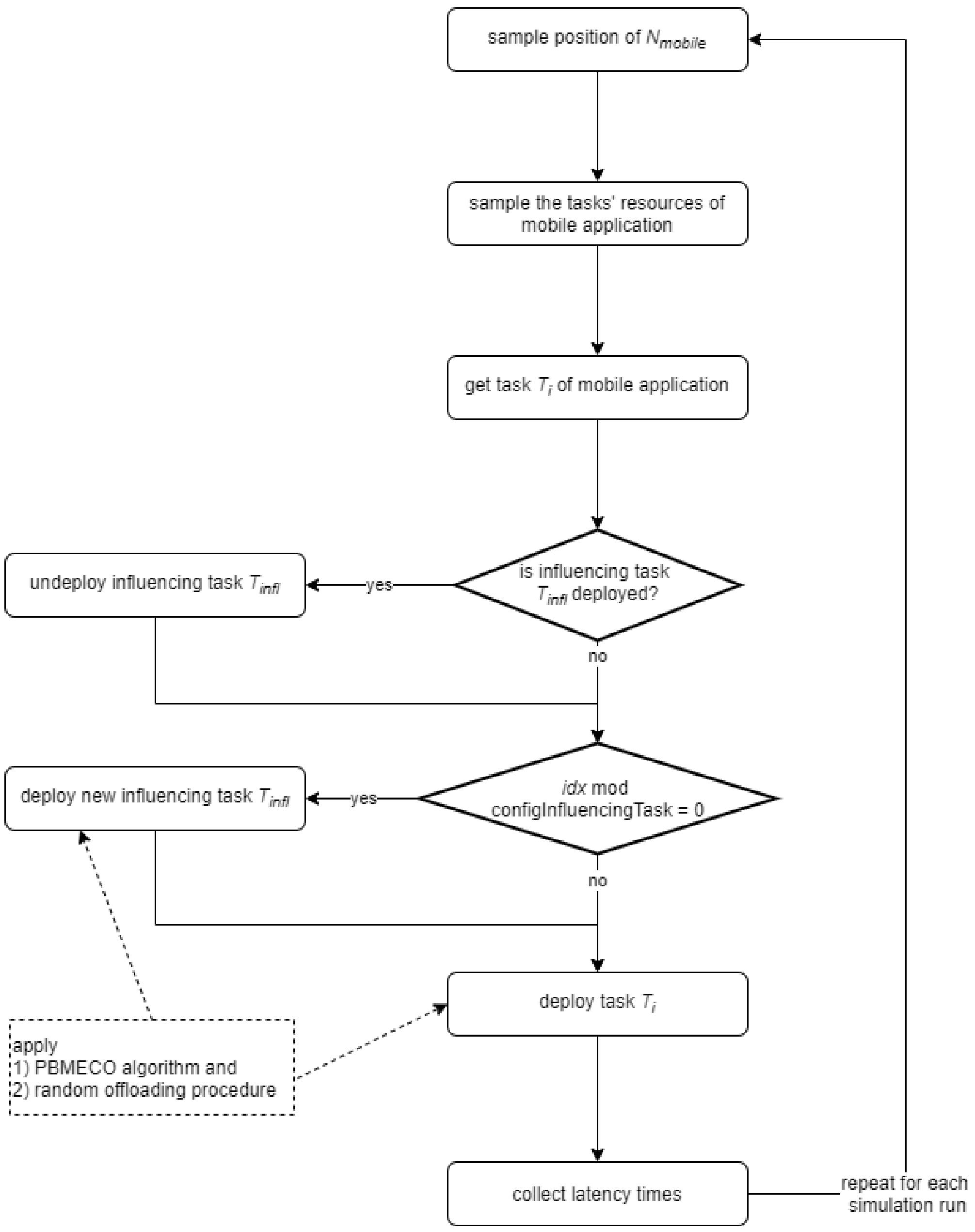
sample position of N_{mobile}
sample the tasks' resources of mobile application
get task T_i of mobile application
is influencing task T_{infl} deployed?
yes
no
undeploy influencing task T_{infl}
idx mod configInfluencingTask = 0
yes
no
deploy new influencing task T_{infl}
deploy task T_i
apply
1) PBMECO algorithm and
2) random offloading procedure
collect latency times
repeat for each simulation run

Figure 6.1: Flowchart of simulation process

confident simulation results. The process is continued by retrieving each task T_i of the mobile application, that has to be offloaded to remote instances like edge or cloud nodes. As there may arise unforeseen tasks during a regular offloading process, the simulation framework has covered this scenario by instantiating so called influencing tasks T_{infl}. Hereby, it is initially checked if there exist T_{infl} that are already deployed to remote nodes. If yes, they are undeployed before new influencing tasks are created and passed to the offloading process. The moment when the creation of T_{infl} is initialized can be configured via a modulo operator $config_{infl}$. During the simulation run, there exist an index count idx which is increased after each deployment of T_i. Everytime the equation

$$idx \mod config_{infl} = 0 \tag{6.2}$$

evaluates to `true`, a new influencing task with communication and computation latency HIGH is created and deployed to a suitable node. Once the potential deployment of an influencing task is done, the task T_i is sent to the offloading process. This step is on the one hand executed by the implemented PBMECO algorithm and on the other hand by using the random offloading procedure. Finally, the computed communication and computation latency times of each task T_i are collected and categorized into each priority group.

The calculations of the communication and computation latency are done based on the formulas 6.3 and 6.4 [RF 12] [Inc20].

$$Communication\ latency = Propagation\ delay + Serialization\ delay + Distance\ delay \tag{6.3}$$

$$Computation\ latency = \frac{MioInstructions \times CPI}{CPU \times Frequency \times 1000} \tag{6.4}$$

The propagation and serialization delay are defined in 5.2 and 5.3. For the calculation of the distance delay, it is assumed that the latency increases over distance due to environmental or networking influences (e.g. buildings or network hops). The formula 6.3 is only applied on calculations for edge nodes because we do not know the exact coordinates of the cloud data centers and thus, we cannot calculate the distance between the mobile and cloud node. Therefore, we have evaluated the propagation delay by using CloudHarmony [Clo20] which measures the Round Trip Time (RTT) between the current position and the cloud data center (see chapter 6.3.3 for more details). Within the equation 6.4, the Cycles per Instruction (CPI) are set to 1 due to simplifications and the frequency is specified in GHz.

These steps of the simulation process are executed for each simulation run. As proposed in the paper *OPTIMAL NUMBER OF TRIALS FOR MONTE CARLO SIMULATION* by Liu, we can achieve a high confidence level by performing anywhere between 100,000 to 500,000 iterations [Liu17, p. 1]. Therefore, the number of simulation runs has been set to the upper bound, which is 500,000 runs for each scenario described in chapter 6.4.2 and each offloading procedure – PBMECO algorithm or random offloading.

Based on the process illustrated in 6.1 it can be seen that there exist three different events, that are used within the Event-driven Monte Carlo Simulation.

1. Movement of mobile node N_{mobile}
2. Deployment of a task (T_i or T_{infl}) to a node (N_{mobile}, N_{edge_j} or N_{cloud})
3. Undeployment of a task (T_i or T_{infl}) from node (N_{mobile}, N_{edge_j} or N_{cloud})

As described in the chapter 6.1, an event changes the state of the system. Hereby, the first event – movement of N_{mobile} – changes the position and therefore the coordinates of it. Additionally, the second and third event result in a change of remaining resources, that are available on either the mobile, edge or cloud node.

6.3 Simulation Environment

The simulation process described in the previous chapter is executed in a predefined environment. This setup remains the same during all simulation runs to make all results comparable with each other. The environment, or also called system in the area of simulations, consists of three types of nodes: 1) one mobile node; 2) four edge nodes and 3) one cloud node. Details about these three components are given in the upcoming chapters 6.3.1, 6.3.2 and 6.3.3. This setup leads to the following network topology, that is used for all simulation runs.

Figure 6.2 illustrates the previously mentioned node types and how they are connected with each other. Firstly, the mobile node – which is the starting point of the offloading procedure – has a connection to both the edge and cloud computing environment. The edge computing environment is represented by the dashed circle including edge servers whereas the cloud infrastructure is symbolized by the cloud sign. Secondly, the edge computing environment consists of four edge nodes. All four nodes can communicate with each other via a high-speed connection, which is assumed to have a bandwidth of 1 Gbps. Furthermore, all edge nodes also have the possibility to connect to the cloud environment in order to perform potential migrations. Finally, the cloud includes only one instantiated virtual machine. For simplification reasons, it is assumed within the simulation that the cloud has unlimited resources available due to scaling even if there must be created additional VMs in reality.

As network connection, the mobile node communicates on the one hand with the edge environment either via 4G or 5G. Details about both network types are given in the chapters 6.3.4 and 6.3.5. On the other hand, it is assumed that the bandwidth between the mobile and cloud node is more limited because it has to be shared by multiple users. Therefore, the bandwidth is set to 150 Mbps as up and download speed.

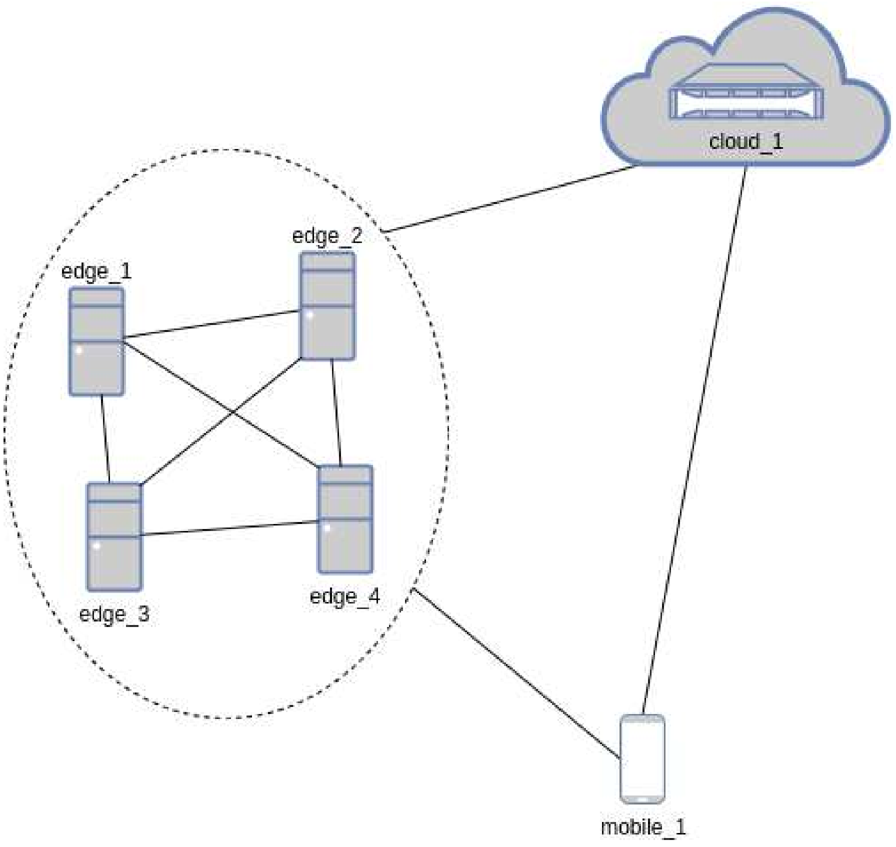

Figure 6.2: Simulation network topology

6.3.1 Mobile node

Within the simulation setup, the mobile node N_{mobile} represents a smartphone which has installed mobile applications that are compatible for doing offloading. It has the following specifications [Fin20]:

Characteristics	Values
Model	iPhone XS
Name	mobile_1
RAM	4 GB
CPUs	4
Storage	64 GB
Coordinates (lat/lon)	sampled randomly
Frequency	2.49 GHz

Table 6.1: Mobile node specification

Within the simulation model, the mobile node is initialized with the parameters described in table 6.1. Additionally, N_{mobile} can hold multiple applications where each has several tasks. Each task is characterized as $T_i(id, name, ram, cpu, storage, isOffloadable,$ $mioInstructionsCount, communicationPriority, computationPriority, sequence)$ and multiple pre-dependency-tasks can be specified for each task T_i.

As movement area of the mobile node, the area of and around the city of Vienna has been specified. Therefore, a rectangle above Vienna has been spanned with the following range starting at the lower left corner and ending in the upper right one: from coordinates(48.13, 16.20) until coordinates(48.31, 16.57).

6.3.2 Edge nodes

Edge nodes are used within the offloading algorithm as main target in order to reduce the latency times. Hereby, we use two different types of state-of-the-art edge nodes from OnLogic and they are initialized within the simulation based on the characteristics described below [OnL19].

Characteristics	Values Standard	Values High Performance
Model	MC850-54 (low-config)	MC850-54 (high-config)
Name	edge_1 / edge_2	edge_3 / edge_4
RAM	16 GB	32 GB
CPUs	6	6
Storage	100 GB	100 GB
Coordinates (lat/lon)	48.19,16.30 / 48.22,16.50	48.28,16.41 / 48.15,16.40
Frequency	2.10 GHz	3.40 GHz

Table 6.2: Edge nodes specifications

As specified in table 6.2, there exist two different types of edge nodes. Firstly, two standard edge nodes with low performance configuration are used and secondly, there are two edge nodes configured with high computational power. The main difference between those nodes can be seen in the amount of RAM available and the frequency per CPU.

The edge locations of the simulation environment are illustrated in figure 6.3 which represents a map of Vienna. This map is built based on D-maps.com [dm20] and the marker icons are used from Feather [Fea20] and have been modified.

The red markers within the map 6.3 represent the standard or low performance edge nodes where the high computational power instances are represented by green pins. All four edge nodes are almost equally distributed via the city of Vienna, so that each node covers almost the same size.

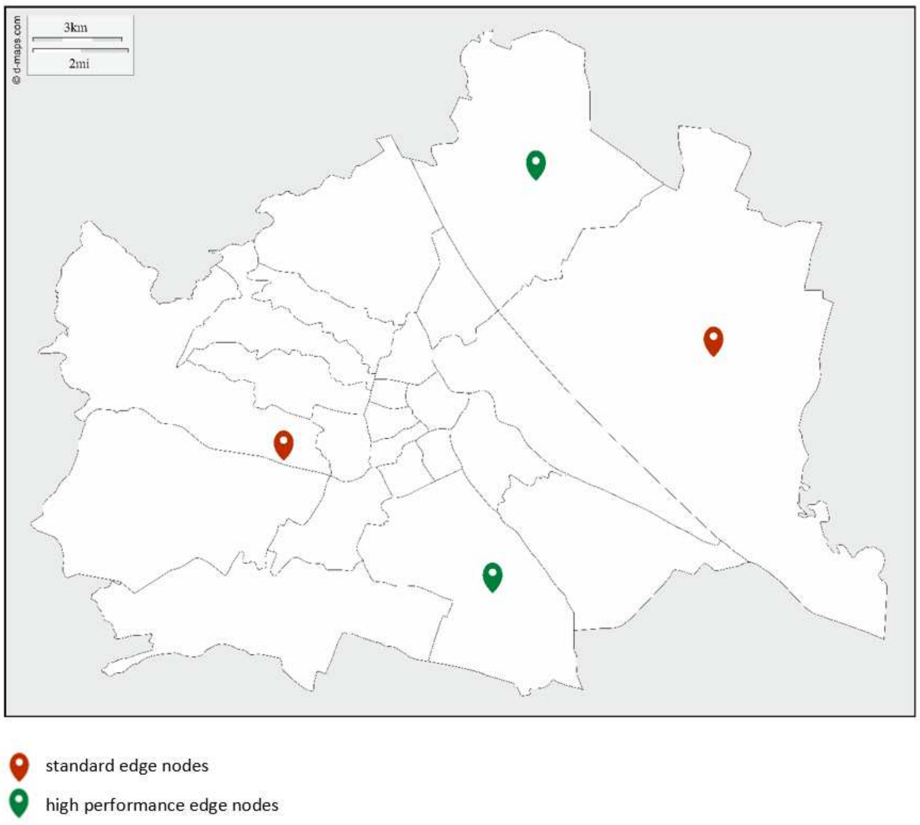

Figure 6.3: Edge environment

6.3.3 Cloud node

Beside the usage of mobile and edge nodes, there is also one cloud node configured within the simulation environment. As cloud node, we use one c5.2xlarge instance from AWS and its configuration is listed below [Ama19].

Characteristics	Values
Model	c5.2xlarge
Name	cloud_1
RAM	16 GB
CPUs	8
Storage	500 GB
Coordinates (lat/lon)	not available
Frequency	3.60 GHz

Table 6.3: Cloud node specification

As already mentioned in chapter 6.2, we are missing information for calculating the distance between the mobile and cloud node and thus cannot calculate the propagation delay. Therefore, we have done four network speed tests using CloudHarmony on different points of time to get the RTT for all existing AWS regions. We have then decided to only use European regions because the simulation environment is based in Vienna. Finally, the results of all EU regions indicated by the prefix "eu-" of all four network speed tests have been averaged, which resulted in a RTT of 90.92 ms. This value is used within the implemented simulation environment to calculate the propagation delay once a task is offloaded to the cloud node.

6.3.4 4G mobile network

Starting with the networking components of the simulation environment, we are on the one hand using the Fourth Generation (4G) standard. Today's 4G - also known as Long Term Evolution (LTE) Advanced - mobile network is the successor of the Third Generation (3G) network. The 3G mobile network has used the Universal Mobile Telecommunication System (UMTS) air interface for transferring data. The need for the new 4G technology came up around 2010. Measurements by Ericsson has shown an increase of the amount of data traffic between 2007 and 2011 by a factor of 100 [Cox12, p. 8]. Cox [Cox12, p. 11] points out three options to increase the capacity of the mobile network: Reduce the size of the cells, increase the bandwidth and improve the communication technology.

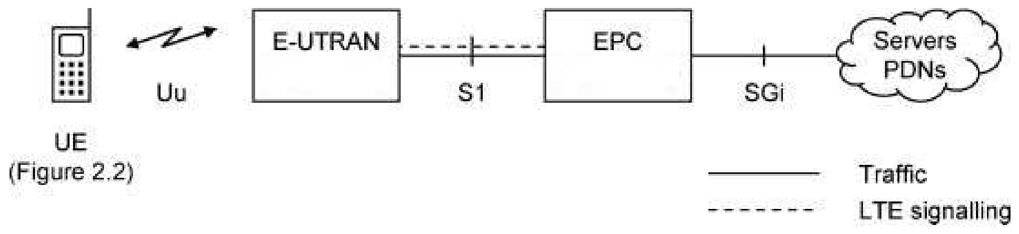

Figure 6.4: 4G architecture

In figure 6.4, the high level architecture of the 4G network can be seen. The Evolved UMTS Terrestrial Radio Access Network (E-UTRAN) represents a base station. It is responsible for handling the radio communication - download and upload of data and handover to other cells - between the User Equipment (UE) (the user's mobile phone) and the Evolved Packet Core (EPC). Then the EPC transmits the received data to the outside world, e.g. the internet and is also the first access point when receiving the data from outside [Cox12, pp. 21-24].

The peak data rates that can be achieved in lab environments by the 4G mobile network are 3000 Mbps downlink and 1500 Mbps uplink [Cox12, p. 303]. In comparison to the real world, the actual download rate lies at about 300 Mbps and the upload rate is about 50 Mbps [LA20].

Therefore, we have configured those two values as downlink and uplink speed within our simulation framework to achieve faithful results. The upload rate is used for calculating the communication latency for the deployment whereas the download rate is applied during the undeployment process. As the 4G mobile network is already well established – especially in urban areas like the city of Vienna, where the simulation takes place – this type of network is chosen as the standard type of communication between the mobile and edge nodes.

6.3.5 5G mobile network

As the amount of produced data on mobile devices is still growing there is a need to improve the possibilities of the 4G mobile network described in chapter 6.3.4. Therefore, the development of the Fifth Generation (5G) mobile network has already begun to overcome the limitations of its predecessor in cellular technology [RSM+13, p. 2]. Rappaport et al. [RSM+13, p. 2] describes the main differences compared to 4G as the use of millimeter waves, longer battery life, lower outage probability, higher bit rates and capacity for simultaneous connections.

When talking about the performance, the new generation of mobile network technology can achieve in theory up to 10 Gbps download speed [Lo18]. In 2018, Telekom has executed tests at IFA where they measured 3 Gbps when downloading data [lte18]. 5G-Anbieter.info states, that the ratio between upload and download will be about 1:5 [GA20].

Based on the real-world experiments of Telekom and the ratio of 1:5 we have set the download speed to 3 Gbps and the upload speed to 600 Mbps. Additionally, it is assumed that the 5G network will be initially built around edge nodes in order to achieve high speed communication and low latency times. Therefore, the mobile node can benefit from using the 5G network once the target edge node is within a range of 5 kilometers.

6.3.6 Random Number Generator

When doing a Monte Carlo Simulation, one essential point is the creation of random numbers. In computer-aided systems, these values are not real random numbers but they are called "Pseudo Random Numbers" because machines are always computing a deterministic numerical sequence $z_0, z_1, ...$ with values of the interval $[0, 1]$ [WH16, p. 21]. The best known pseudo random number generators are "Linear Congruential Generators" and they have been very popular for a long time [WH16, p. 21]. But according to Waldmann and Helm, the state-of-the-art pseudo random number generator in Monte Carlo Simulation is the Mersenne Twister [WH16, p. 27]. The Mersenne Twister has been implemented by Matsumoto and Nishimura in 1998. This random number generator provides a period of $2^{19937} - 1$ and a 623-dimensional equidistribution [MN98].

Therefore, the generation of (pseudo) random numbers in this simulation framework is done by using the implementation of the Mersenne Twister, which is available in Java's

Apache Commons library [Fou16]. Hereby, we are using the concrete version Apache Commons Math 3.6.1.

6.4 Simulation Runs

In order to evaluate the performance of the implemented PBMECO algorithm compared to a random offloading procedure, several simulation runs are executed. Each simulation run is configured based on a scenario description, which are given below in 6.4.2.

The simulations are executed using the following hardware and software setup:

- Desktop PC using Windows 10 Pro, Intel(R) Core(TM) i7-4790K @ 4.00 GHz and 16 GB RAM
- Java v1.8.0_152
- Java library Apache Commons Math v3.6.1
- IntelliJ IDEA v2019.3 (Ultimate Edition)

6.4.1 Metrics

Metrics within the evaluation process are needed to make the results comparable regarding performance of the algorithm. As this thesis focuses on the improvement of latency times of highly sensitive applications, the following evaluation metrics are computed:

- **Average communication latency per priority group**
  This metric specifies the calculation of the average of the communication latency for all offloadable tasks. The latency times are collected for each priority group during each simulation run and aggregated at the end of the simulation. It results in three computed numbers namely $CommunicationLatency_{HIGH}$, $CommunicationLatency_{MEDIUM}$ and $CommunicationLatency_{LOW}$.
- **Average computation latency per priority group**
  Similar to the metric "average communication latency per priority group" specified above, this metric calculates average values for all offloadable tasks. But hereby, we are calculating the averages using the computation latency. This leads to the outputs $ComputationLatency_{HIGH}$, $ComputationLatency_{MEDIUM}$ and $ComputationLatency_{LOW}$.

6.4.2 Scenarios

As already mentioned in the previous chapter, the simulation process takes several scenario configurations as input. The detailed descriptions of the defined scenarios can be found in the upcoming sections. We are using three different real-world scenarios within the evaluation process.

Scenario 1: Percentage of tasks have communication priority HIGH

Scenario 1 is used to simulate the performance of the PBMECO algorithm regarding communication latency. Hereby, a specific percentage of all tasks of the mobile application are configured with communication latency HIGH. This leads to the following three sub-scenarios:

- 1a: 25% of all tasks have communication priority HIGH
- 1b: 50% of all tasks have communication priority HIGH
- 1c: 75% of all tasks have communication priority HIGH

The selection of the tasks, that are configured with priority HIGH, is done on a random base. The remaining tasks are randomly configured with communication latency MEDIUM or LOW. As this scenario focuses only on the performance regarding communication latency, the computation latency for all tasks is set to MEDIUM or LOW.

Setup
This scenario uses a navigator application modeled as Directed Acyclic Graph (DAG), which is shown below:

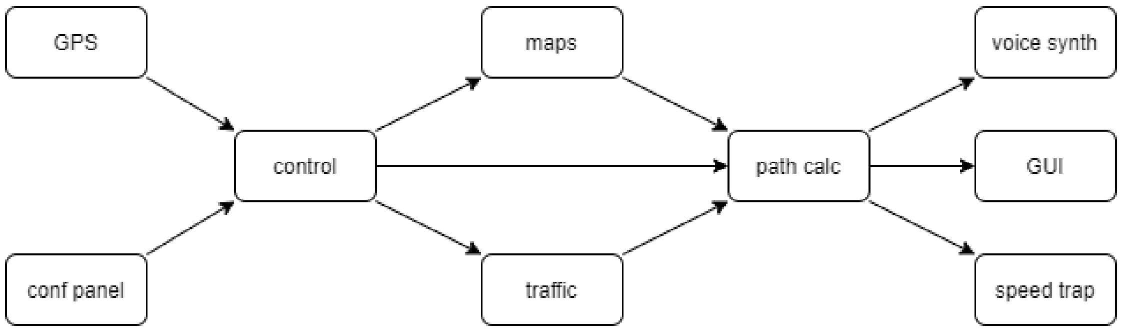

Figure 6.5: DAG of navigator application (adapted from De Maio and Brandic [DMB18])

As illustrated in figure 6.5, this navigator application consists of nine tasks where almost each of them depends on a previous task (e.g. maps depends on control). The resource configuration of each individual task is given in the table 6.4 (adapted from De Maio and Brandic [DMB18] because million of instructions have been adapted to match milliseconds).

Name	RAM (MB)	CPUs	Storage (MB)	Offloadable	Mio Instructions
conf panel	1000	1	1005	false	$\frac{1}{\lambda} = 1000$
GPS	3000	1	5005	false	$\frac{1}{\lambda} = 1000$
control	3000	2	1005	true	$\frac{1}{\lambda} = 2000$
maps	5000	2	$5000 + (\frac{1}{\lambda} = mapSize)$	true	$\frac{1}{\lambda} = 3000$
traffic	1000	1	$5000 + (\frac{1}{\lambda} = mapSize)$	true	$\frac{1}{\lambda} = 5000$
path calc	2000	1	$5000 + (\frac{1}{\lambda} = mapSize)$	true	$\frac{1}{\lambda} = 5000$
voice synth	1000	1	5005	false	$\frac{1}{\lambda} = 2000$
GUI	1000	1	5001	false	$\frac{1}{\lambda} = 2000$
speed trap	1000	1	5010	false	$\frac{1}{\lambda} = 2000$

Table 6.4: Resource configuration: navigator application

When applying this scenario, the mapSize is varied between 25 and 500 MB and are set as $\frac{1}{\lambda}$ for the exponential distribution [DMB18, p. 6]. Furthermore, also the number of Million Instructions is randomly calculated as exponential distribution like stated by De Maio and Brandic [DMB18, p. 6].

Results

The simulation applying the first scenario evaluates the performance of the PBMECO algorithm with respect to communication latency of HIGH priority tasks. After 500,000 simulation runs with randomized task resources, the actual results are presented in the bar chart 6.6. The fraction of HIGH priority tasks is displayed on the x-axis whereas the y-axis shows the communication latency time in Milli Seconds (ms). Moreover, the dark blue bars within the diagram represent the average $CommunicationLatency_{HIGH}$ of the PBMECO algorithm whereas the light blue bars are displaying the results from the random offloading procedure.

By analyzing figure 6.6, one can see that the algorithm can achieve constant times regardless of whether 25%, 50% or 75% of the application's tasks have set the communication latency to HIGH. In absolute values, the algorithm has steadily achieved 28 ms on average throughout all offloading procedures. Compared to the random selection of target nodes, the PBMECO algorithm can achieve nearly half the time (41% to be exact) for a fraction of 25%, 50% and 75% HIGH priority tasks.

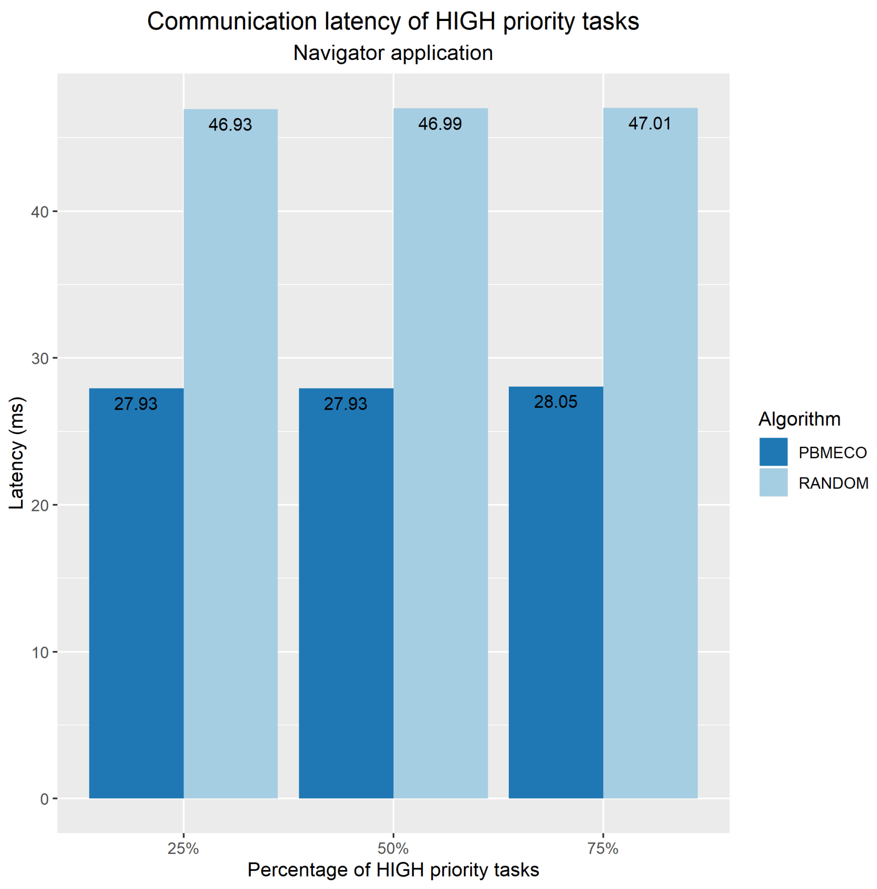

Figure 6.6: Scenario 1: results for communication latency of HIGH priority tasks

Scenario 2: Percentage of tasks have computation priority HIGH

This scenario has been defined in order to evaluate the performance of the algorithm with respect to computation latency. Similar to scenario 1, we have here also specified three sub-scenarios with different percentages of HIGH priority tasks:

- 1a: 25% of all tasks have computation priority HIGH
- 1b: 50% of all tasks have computation priority HIGH
- 1c: 75% of all tasks have computation priority HIGH

The HIGH priority tasks are chosen randomly and the computation latency of all remaining tasks is randomly set to either MEDIUM or LOW. Furthermore, none of the tasks has configured the communication latency with priority HIGH because this scenario is only used to measure the efficiency of the PBMECO algorithm regarding to computation latency.

Setup

The setup is identical to the one described in scenario 1. We are using the same navigator application, which is illustrated in figure 6.5 and this navigator application makes use of the resource configuration described in the table 6.4.

Results

Similar to the first scenario, the evaluation scenario 2 measures the performance of the algorithm for different percentages of HIGH priority tasks. But this scenario instead focuses on the computation latency and the collected results are illustrated in figure 6.7. The axis are defined the same way like in the bar chart 6.6 and represent the number of HIGH priority tasks and the achieved latency time in ms. The measure $ComputationLatency_{HIGH}$ of the PBMECO algorithm is displayed by the dark blue bars and for the random procedure, light blue bars are used.

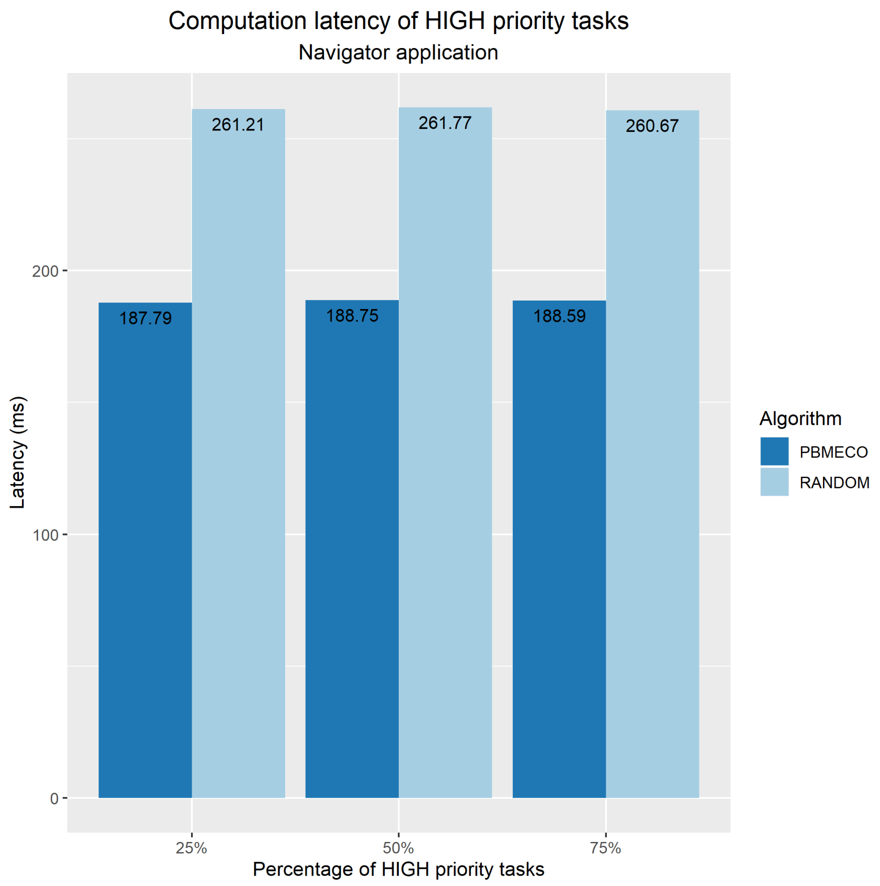

Figure 6.7: Scenario 2: results for computation latency of HIGH priority tasks

The concrete results in figure 6.7 show that the average computation latency during the offloading process of the PBMECO algorithm is constant for each percentage of HIGH priority tasks – 25%, 50% or 75% of the application's tasks. Hereby, the overall performance of the algorithm is ∼188 ms compared to ∼261 ms when the target offloading node is selected randomly and no priority differentiation is made. Therefore, in relation to the random offloading process, the algorithm can reduce the computation latency by ∼30% no matter of how many tasks have set its priority to HIGH.

Scenario 3: Antivirus application

In scenario 3 we are using the real-world antivirus application ClamAv v0.102.1. An antivirus application has been chosen, because it potentially has to transfer large files and therefore has to achieve low communication latency times.

Setup
Within this scenario we have modeled the antivirus application based on the following DAG illustration. The model has been adapted from De Maio and Brandic [DMB18].

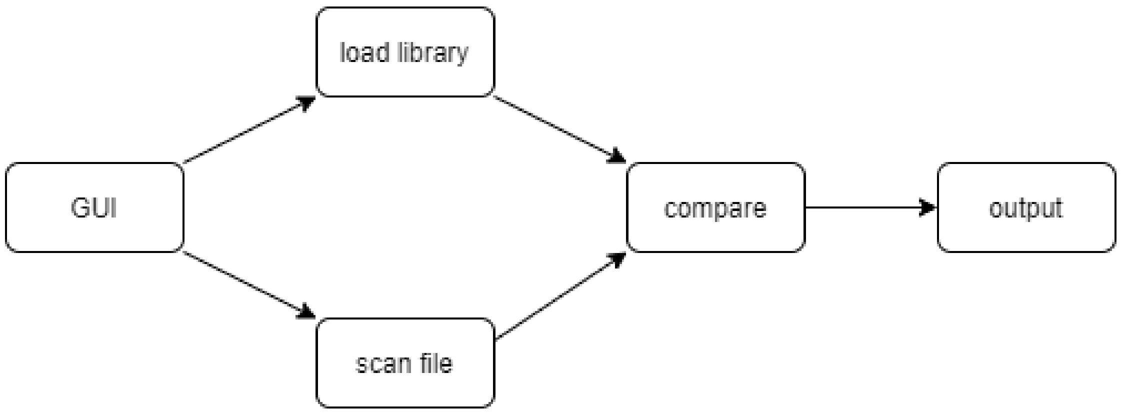

Figure 6.8: DAG of antivirus application

As shown in figure 6.8 the antivirus application is split into five individual tasks called GUI, load library, scan file, compare and output. The configuration of each individual task can be seen in 6.5.

Name	RAM (MB)	CPUs	Storage (MB)	Offloadable	Mio Instructions
GUI	$\mathcal{N}(10, 1)$	1	20	false	$\mathcal{N}(1,000, 10)$
load library	$\mathcal{N}(10, 1)$	1	170	true	$\mathcal{N}(5,000, 50)$
scan file	$\mathcal{N}(15, 1.5)$	2	file size	true	$\mathcal{N}(32,500, 325)$
compare	$\mathcal{N}(15, 1.5)$	2	170 (library) + file size	true	$\mathcal{N}(28,000, 280)$
output	$\mathcal{N}(5, 0.5)$	1	10	false	$\mathcal{N}(500, 5)$

Table 6.5: Resource configuration: antivirus application

The total values of RAM and millions of instructions of the antivirus application have been measured on the one hand with the linux tool valgrind –tool=massif and on the other hand with the linux tool perf stat. For retrieving the values, we have applied different file sizes and types: 1) text file (70 KB), 2) PDF file (1.8 MB), 3) ZIP file (1.4 GB) and 4) ZIP file (15 GB). As there exist no correlation between file size and RAM and millions of instructions needed, we have specified within our simulation that those two resource specification are normally distributed with a variance of 1% of the expected value ($X \sim \mathcal{N}(\mu, \mu \times 1\%)$). Furthermore, the size of the input file is randomly configured between 70 KB and 1 GB.

From table 6.5 it can be seen that the application has three offloadable tasks – namely load library, scan file and compare. The communication and computation latency of those three tasks are defined below in table 6.6:

Name	Communication latency	Computation latency
load library	MEDIUM	LOW
scan file	HIGH	MEDIUM
compare	HIGH	MEDIUM

Table 6.6: Communication and computation latency priority configuration: antivirus application

The two tasks scan file and compare are configured with communication latency HIGH because they need to transfer large amount of data and therefore rely on short distances between the mobile and target node and high bandwidths.

Results

The third scenario of the experimental simulation deals with a real-world antivirus application and analyzes the performance of the algorithm with respect to communication and computation latency. The concrete results of the simulation runs have been illustrated as bar charts in 6.9 and 6.10. Both charts only differ in the fact, that figure 6.9 displays the communication latency times whereas figure 6.10 illustrates the computation latency for the different priority groups. In both diagrams, the latency priority is displayed on the x-axis and the achieved latency time in ms is represented by the y-axis. Like in the bar charts in scenario 1 and 2, again the dark blue bars outline the results of the PBMECO algorithm whereas the light blue bars demonstrate the measured latency times using the random offloading procedure.

As it can be seen in figure 6.9, the communication latency of HIGH priority tasks can be considerably reduced by about 40%. This means an absolute decrease in time from ∼97 ms to ∼58 ms, which results in a difference of ∼39 ms. When analyzing the average values of MEDIUM prioritized tasks, the diagram illustrates that the algorithm has a worse performance compared to the randomly selecting target nodes for remote execution. Hereby, the algorithm needs ∼3 ms more time. This result can be explained by the implemented behavior of the algorithm, which tries to predict the needed resources of future HIGH priority tasks and thus reserves the predicted resources on the closest edge nodes. Therefore, the task is offloaded to a more distant edge node, which results in a slight increase for the measure $CommunicationLatency_{MEDIUM}$. The value -1 for LOW priority task means, that no task of the application has configured this priority class.

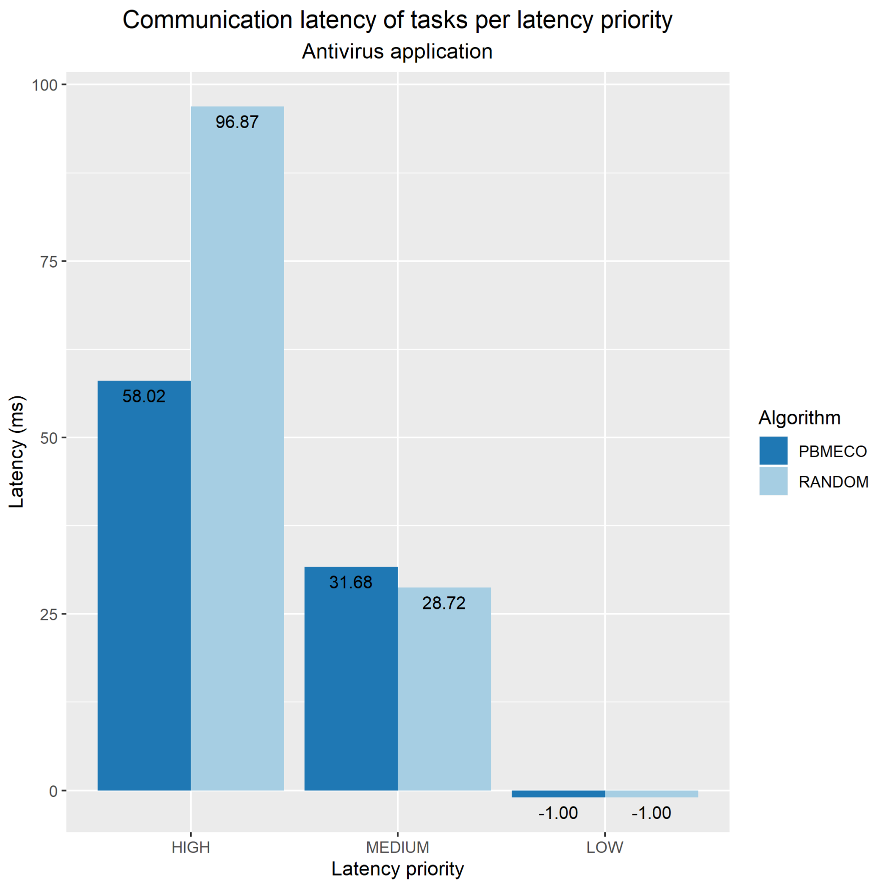

Figure 6.9: Scenario 3: results for communication latency of antivirus application per priority group

When taking a closer look at the bar chart 6.10, one can immediately see the value -1 for tasks with priority group HIGH. As already mentioned above, the value -1 specifies that no tasks of this class exist in the application setup. Continuing with the average times for tasks with computation latency MEDIUM. Hereby, it is clearly shown that the PBMECO algorithm as well as the random offloading procedure have achieved almost the same result. The absolute difference is only ∼47 ms, which leads to a relative gap of 1%. The same behavior can be observed for LOW priority tasks. Due to the slight increase or decrease of 1%, it cannot be concluded that the PBMECO algorithm can improve the performance for MEDIUM or LOW prioritized tasks for this type of application.

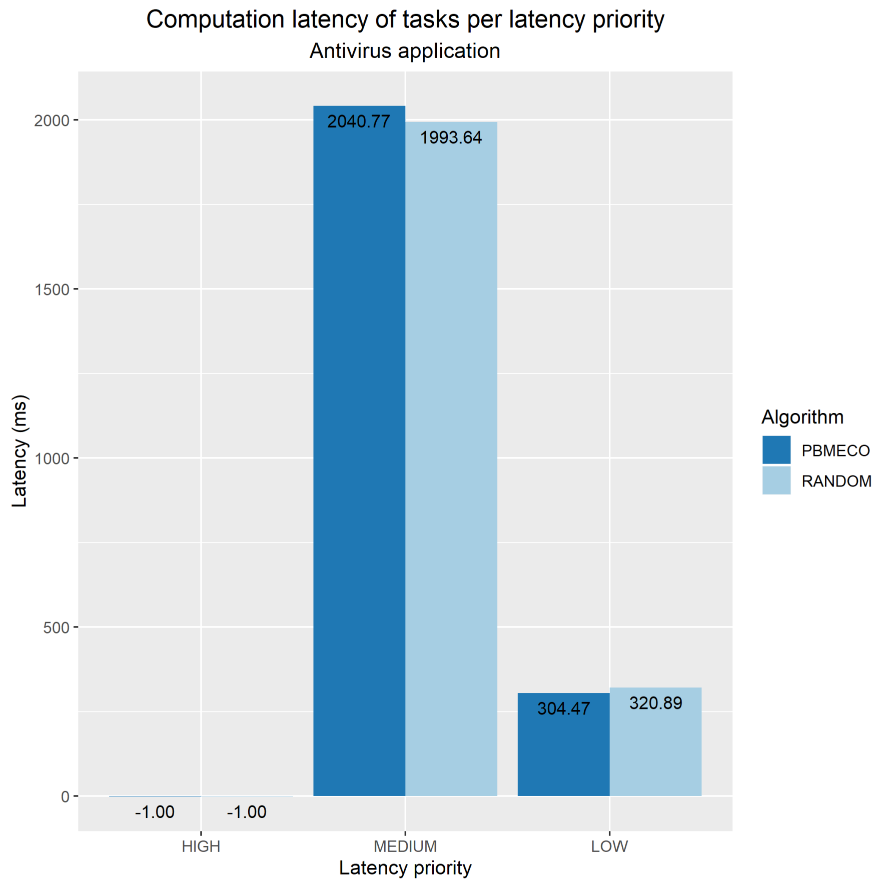

Figure 6.10: Scenario 3: results for computation latency of antivirus application per priority group

In summary, it can be stated that the PBMECO algorithm is an appropriate choice for doing task offloading, when the application needs to transfer large amounts of data. As it has been shown in the simulation, it is very important to accurately configure the right tasks with the right priority class in order to achieve the best performance.

Scenario 4: Face Recognition application

Scenario 4 applies the face recognition application v1.2.3 from `https://github.com/ageitgey/face_recognition`. Compared to the antivirus application, a face recognition system includes compute intensive tasks. Therefore, this type of application has been chosen in order to simulate the performance of the PBMECO algorithm for a real-world application, which focuses on low computation latency times.

Setup

The face recognition application consists of five tasks. How the individual tasks are connected to each other is shown in figure 6.11.

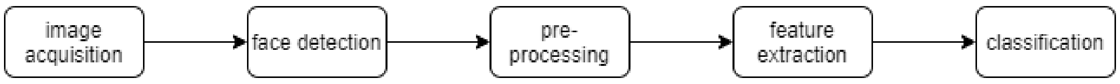

Figure 6.11: DAG of face recognition application (modified from Mao et al. [MYZ+17])

As illustrated in figure 6.11, all tasks of the application are executed in sequential order. The individual tasks and their resource configurations can be extracted from table 6.7.

Name	RAM (MB)	CPUs	Storage (MB)	Offloadable	Mio Instructions
image acquisition	$RAM_{calc} \times 0.1$	1	image size	false	$\mathcal{N}(10,000,200)$
face detection	$RAM_{calc} \times 0.5$	2	image size	true	$\mathcal{N}(30,000,600)$
pre-processing	RAM_{calc}	2	image size $\times$ 1.1	true	$\mathcal{N}(50,000,1,000)$
feature extraction	RAM_{calc}	2	image size $\times$ 2	true	$\mathcal{N}(75,000,1,500)$
classification	RAM_{calc}	3	image size $\times$ 2	true	$\mathcal{N}(85,000,1,700)$

Table 6.7: Resource configuration: face recognition application

The resource usages of the application for RAM and millions of instructions have been measured by using the linux tools valgrind –tool=massif and perf stat like in scenario 3. The measures have been collected by executing the face recognition application on a sample image library of 100 images from the original dataset by Liu et al. [LLWT15]. Hereby, the input file that has to be processed by the application had different file sizes starting from 10 KB until 700 KB. After analyzing the measures, we have identified a correlation between image size of the input file and RAM needed. Therefore, we have formulated the following equation after computing the coefficients with the function lm() of R:

$$RAM_{calc} = 171.3723 + image\ size \times 0.1459 \tag{6.5}$$

Additionally, there exists no correlation between millions of instructions needed and the processed image size. Therefore, in our simulation setup it is configured that this type of resource is normally distributed with a variance of 2% of the expected value ($X \sim \mathcal{N}(\mu, \mu \times 2\%)$). Finally, the file size of the image that has to be processed is computed on a random base and lies within the range 100 KB – 5 MB.

As specified in the table 6.7, four tasks of the application can be offloaded to and executed on remote nodes. These four tasks are configured with the following communication and computation latency priorities:

Name	Communication latency	Computation latency
face detection	HIGH	LOW
pre-processing	LOW	MEDIUM
feature extraction	LOW	HIGH
classification	MEDIUM	HIGH

Table 6.8: Communication and computation latency priority configuration: face recognition application

In this scenario, we have configured the task face detection with communication latency HIGH, because the image should be transferred as fast as possible for further processing. Additionally, the reason why we have set the computation latency of the two tasks feature extraction and classification to HIGH is, that those two tasks require the most number of CPUs and millions of instructions and therefore need the most computational power.

Results

In scenario 4 we have setup a face recognition application and executed the 500,000 simulation runs like we did for the other scenarios too. All the collected results have been summarized in figure 6.12 and 6.13. Those two diagrams represent on the one hand the measured communication latency times and on the other hand the computation latency times. As the diagrams used in scenario 3, here they are also built using the latency priority groups on the x-axis and the achieved latency times in ms on the y-axis. The results of the PBMECO algorithm are represented by the dark blue bars and the ones from the random offloading procedure are outlined using light blue bars.

When looking at the first bar chart 6.12, it shows the results with respect to communication latency of the face recognition application. As illustrated there, the algorithm can improve the performance for all three priority groups HIGH, MEDIUM and LOW. Starting with HIGH prioritized tasks, the value for $CommunicationLatency_{HIGH}$ is there three times higher when the offloading target is selected randomly without differentiating task priorities compared to the measured results by applying the PBMECO algorithm as offloading procedure. Furthermore, also the latency time for tasks with communication latency MEDIUM and LOW can be reduced by ∼25% and ∼35% respectively.

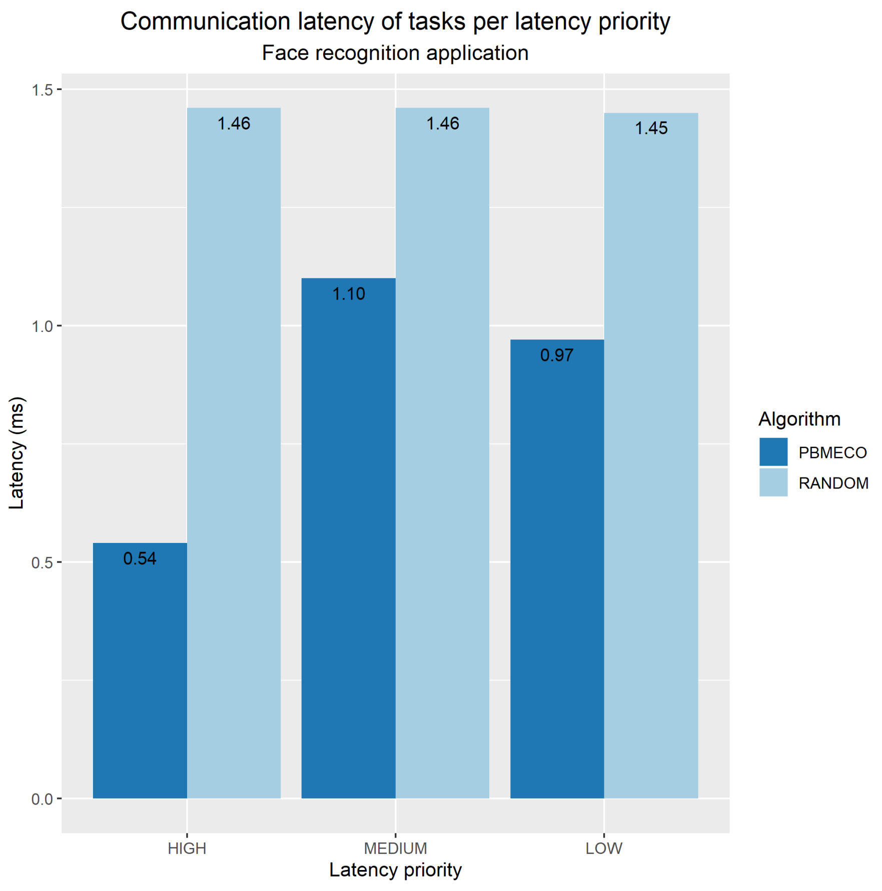

Figure 6.12: Scenario 4: results for communication latency of face recognition application per priority group

By analyzing the second bar chart 6.13, one can see the measured computation latency times for the PBMECO algorithm and random offloading procedure. Firstly, the computation latency of HIGH priority tasks of the face recognition application can be reduced from ∼5.1 seconds to ∼3.9 seconds, which is a reduction of almost 25%. Secondly, the results illustrated in figure 6.13 show that also the latency of MEDIUM prioritized tasks can be reduced by applying the PBMECO algorithm. Hereby, we can achieve a decrease of 25% compared to the simulated latency times of the random offloading procedure. Finally, for tasks that have set their computation latency to LOW, we have to record a slight increase of 5% in computation latency when offloading the tasks to remote nodes when using PBMECO.

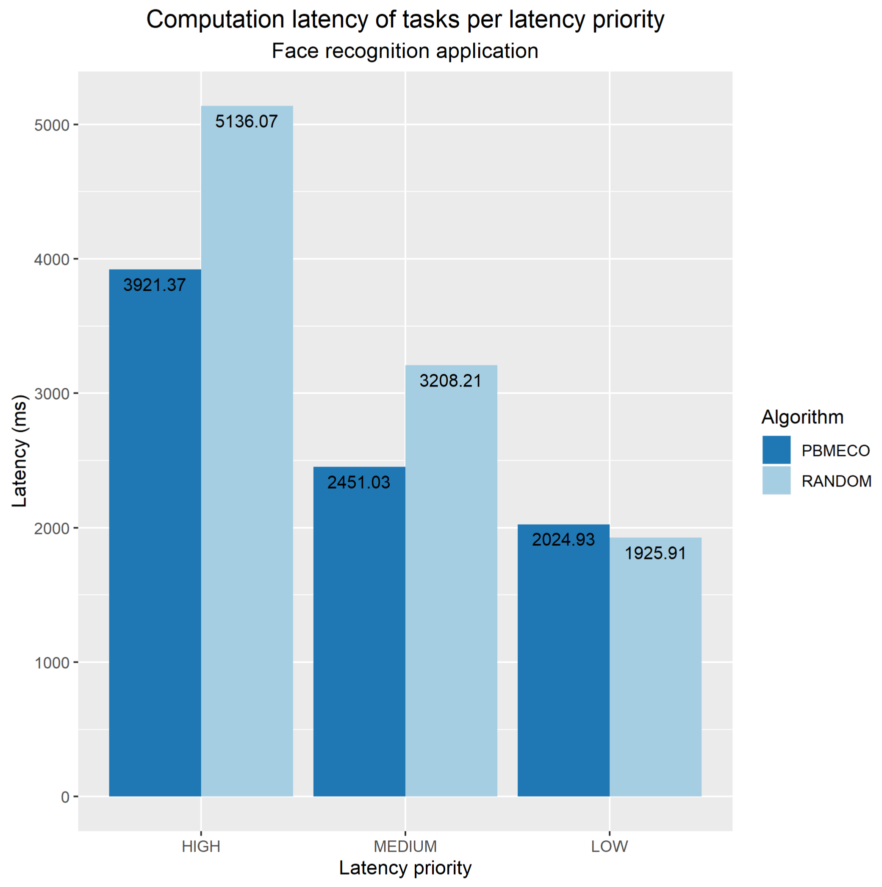

Figure 6.13: Scenario 4: results for computation latency of face recognition application per priority group

Summarizing the results for the face recognition application, one can clearly identify that the Priority Based Mobile Edge Cloud Offloading (PBMECO) can improve the performance for nearly every measurement. Only for LOW prioritized tasks in computation latency, we have to record a slight degradation in performance when the PBMECO is applied as offloading procedure. Therefore, we can conclude that the implemented algorithm is very suitable for such types of applications that require higher computational power and still rely on short latency times like shown with the chosen face recognition application.

CHAPTER 7

Conclusion

The final chapter starts with a discussion on the findings and results of this diploma thesis. It continues by describing the limitations of the provided offloading approach in this work and it ends with suggestions for future research.

7.1 Discussion

In this diploma thesis we have been dealing with the topic of task offloading into an Edge and Cloud Computing environment. We have seen that especially Edge Computing is an emerging area and it has become more interesting in the recent years due to the rising demand of IoT and wearable devices and the need for offering near real-time applications to end users.

Therefore, we have initially conducted a systematic mapping study to analyze existing research on the topics Edge and Cloud Computing. The primary goal was to identify what benefits and limitations both paradigms have to understand how they can be integrated into a task offloading algorithm. On the one hand we have identified that the main advantages of Cloud Computing with respect to task offloading are its high availability and flexible scalability. By making use of this characteristics, it is possible to provide a stable and reliable backbone to meet fluctuating demands. On the other hand, the concept of Edge Computing is the main enabler for providing and supporting near real-time applications. Due to its proximity to end users, Edge Computing can provide low latency and high bandwidth in order to achieve near real-time end-to-end communication.

Based on these insights, we have designed a Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm to reduce the total latency time which consists of network and computation latency. Due to this fact we have decided to use Edge Computing as the main offloading target where cloud instances only serve as backup solution in case no edge server is available. The algorithm itself consists of two main parts – one passive and one active. Firstly, the passive component of the PBMECO is responsible for scanning the surroundings of the environment to keep track of nearby edge servers and their available resource capacities (e.g. CPU or RAM). Secondly, the active part is the real offloading process that kicks in once a task is ready for remote execution. Hereby, the algorithm is operating and making offloading decisions based on predefined priorities and the tasks' resource requirements. This design decision was essential because there must be some background job which tracks position changes on a regular basis whereas the offloading process needs only be started on-demand. In this work we have defined three priorities – HIGH, MEDIUM, LOW – and the two latency categories 1) communication and 2) computation latency. When applying our PBMECO algorithm, one of these priorities has to be assigned to each offloadable task and to each latency category. By taking this design choice, we can enable developers to prioritize the tasks by themselves because those are the people who know the application best. The algorithm uses this information for identifying, which task should take precedence over the other. During each offloading process, the algorithm also tries to make predictions for possible future task requirements for each priority group in order to potentially reserve resources on nearby edge servers for HIGH prioritized tasks. As forecast method, we have decided to use the method "moving averages of order five" to follow resource trends over time and to overcome one-time outliers.

Finally, we have done an event-driven Monte Carlo simulation in order to evaluate the performance of our PBMECO algorithm. For all scenarios we have computed the metrics "average communication latency per priority group" or "average computation latency per priority group". The scenarios cover three real-life applications and are setup as a single user and multiple server Edge Cloud Computing environment. Within Scenario 1, we have evaluated the performance with respect to communication latency based on a navigator application. This scenario has been set up to analyze how the algorithm performs when the number of tasks with communication priority HIGH is increasing. Numerical results of our simulation runs have shown, that the communication latency of HIGH prioritized tasks remains constant and it can be reduced by up to 41%. Scenario 2 is using the same navigator application but focusing on the computation latency and how the performance of the PBMECO changes when the number of tasks with computation priority HIGH is growing. The concrete results have demonstrated, that the algorithm can provide a constant performance for this setup too and it is possible to cut down the computation latency of HIGH priority tasks by up to 30%. The results of our Monte Carlo simulation in Scenario 3 and Scenario 4 based on two other real-life applications have shown that our offloading approach can achieve a reduction of up to 65% for tasks with communication latency priority HIGH and up to 35% for the priorities MEDIUM and LOW. Moreover, the metric average computation latency per priority group demonstrates that applying the PBMECO helps to reduce the latency time for HIGH and MEDIUM prioritized tasks by up to 25%. Summarizing the results of the Monte Carlo simulation, it can be clearly seen that the proposed offloading solution can provide a performance boost for latency sensitive tasks for both communication and computation latency, which are declared with priority MEDIUM or HIGH.

7.2 Limitations

One limitation of our implementation of the Priority Based Mobile Edge Cloud Offloading (PBMECO) algorithm is that it is designed to only support near real-time applications. Based on the definition of "near real-time" we do not cover scenarios where the offloading procedure must fulfill specified deadlines [OnL11]. The main focus of our offloading algorithm is to reduce the network and computation latency of high prioritized tasks. This means it can be applied to mobile applications, that are not mission critical, with the goal of increasing the user experience.

Furthermore, the PBMECO algorithm supports the mobility of users only to a certain extent. Our solution tracks position changes and updates the information about edge servers in the close surroundings of the user, but our implementation does not evaluate if it would be better to migrate the task to a closer edge node when there was a significant position change.

When talking about mobility of User Equipments, this work and the implemented offloading algorithm also do not cover situations where the mobile device leaves the area of the Edge Computing environment. We assume in this work that the user has always the possibility to connect to edge devices and therefore, it is not analyzed what needs to be done when the offloading process has already started and some tasks have already been sent to remote instances for execution.

Moreover, the proposed offloading approach is currently not designed to handle task execution failures. There is no fallback mechanism implemented, like for example re-deployment of a task or rollback of the offloading process, which kicks in when an error occurs. In this diploma thesis, we assume that all remote task executions are successful and they do not break the regular offloading behavior.

Our implementation of the PBMECO algorithm also does not consider the topic of energy consumption. In our work we do not evaluate if our algorithmic computations have a positive, negative or any impact on the energy usage at the mobile device.

7.3 Future Work

As Mobile Edge Cloud Offloading is an emerging research area, there are still remaining challenges that can be tackled in future work. Based on the limitations described in chapter 7.2, there are some possible extensions that can be made on our presented offloading algorithm.

One challenging research topic for future work is user mobility. Hereby, it would be necessary to propose a model which takes already offloaded tasks and position changes of the mobile device into account. Within the model it could be evaluated whether a potential migration to another remote instance can reduce the total latency time. One essential point to focus here is also to avoid ping-pong migrations, which means the task is always migrated between instances due to fast and frequent position changes. The model for user mobility could be also integrated into our proposed offloading algorithm by extending the migration component.

Another interesting area for further research would be the extension of our approach by making a joint offloading decision which considers on the one hand the reduction of latency based on task priorities and resource requirements, and on the other hand the energy consumption. This topic would be important too beside achieving low latency times, because battery lifetime is an additional limitation on mobile devices. For tackling this problem, it would be needed to implement a heuristic for making trade-offs between latency times and energy usage.

Both the topics user mobility including task migrations and joint task offloading considering latency and energy consumption could potentially supported by using already existing Machine Learning (ML) algorithms. One idea would be to implement a ML tool within the cloud to benefit from its high computational power. The mobile devices can then request the results of the ML algorithm and use them for making and improving offloading decisions over time.

One additional technically challenging topic is the implementation of fault tolerance mechanisms. In each mobile application there exist some tasks whose execution is crucial for everything to be working. This aspect is also very important when doing task offloading because the offloading algorithm has to cover such scenarios by for example doing a rollback or re-deploying the task. Hereby, it would be required to identify what information is essential for making decisions within the fault tolerance mechanism and how it cloud be integrated into the existing solution.

As our proposed work only focuses on a single user and multiple server Edge Cloud Computing environment, it would also be interesting in future work, how the PBMECO algorithm performs with multiple users. Initially, this could be evaluated using a simulation and afterwards, a real field study could potentially be conducted to evaluate the performance in real-life.

List of Figures

List of Tables

List of Algorithms

List of Listings

Acronyms

GB	Gigabyte. 66–68, 71, 77
Gbps	Gigabits per Second. 65, 70
GDPR	General Data Protection Regulation. 32
GHz	Gigahertz. 64, 66–68, 71
IaaS	Infrastructure-as-a-Service. 9, 12
IFA	Internationale Funkausstellung. 70
IoT	Internet of Things. 1, 2, 10, 11, 13, 23, 31, 85
IP	Internet Protocol. 15
IrDA	Infrared-enabled. 16
ISACA	Information Systems Audit and Control Association. 15
IT	Information Technology. 10
KB	Kilobyte. 77, 80
LACCH	Latency of Communication and Computation Heuristic. 49, 50
LAN	Local Area Network. 15, 32
LISP	Locator/ID Separation Protocol. 11
LTE	Long Term Evolution. 69
MB	Megabyte. 42, 43, 73, 77, 80
Mbps	Megabits per Second. 65, 69, 70
MCC	Mobile Cloud Computing. 19, 21
MDC	Micro Data Center. 2
MEC	Mobile Edge Computing. 19, 20
MECO	Mobile Edge Cloud Offloading. 6, 7, 19, 20, 26, 87, 89
MI	Million Instructions. 73
ML	Machine Learning. 88
MQTT	Message Queuing Telemetry Transport. 14
ms	Milli Second. 3, 69, 73, 75–78, 81

MSS	Mobile Support Station. 17
MU	Mobile Unit. 17
NIST	National Institute of Standards and Technology. 8, 9
OFDMA	Orthogonal Frequency-Division Multiple Access. 23, 24
OPC UA	Open Platform Communications Unified Architecture. 14
PaaS	Platform-as-a-Service. 9, 12
PBMECO	Priority Based Mobile Edge Cloud Offloading. xi, xiii, 5, 6, 37–48, 50, 51, 53–55, 57, 59–62, 64, 71–75, 77–79, 81–83, 85–88, 93
PDA	Personal Digital Assistant. 15, 16
RAM	Random-Access Memory. 42, 50, 53–55, 57, 62, 66–68, 71, 76, 77, 80, 86
RAN	Radio Access Network. 19
RFID	Radio Frequency Identification. 10, 16
RTT	Round Trip Time. 64, 69
SaaS	Software-as-a-Service. 8, 9, 12
SBS	Small-cell Base Stations. 24
SDTO	Software Defined Task Offloading. 23, 26
SMD	Smart Mobile Device. 18
TDMA	Time-Division Multiple Access. 23–25
UE	User Equipment. 24, 25, 69, 87
UMTS	Universal Mobile Telecommunication System. 69
USB	Universal Serial Bus. 16
UUID	Universally Unique Identifier. 47
VM	Virtual Machine. 31, 33, 65
WAP	Wireless Application Protocol. 15
ZB	Zeta Byte. 8, 11

Bibliography

[A$^+$09] Kevin Ashton et al. That 'Internet of Things' thing. *RFID Journal*, 22(7):97–114, 2009.

[AAS14] Ahmed Alzahrani, Nasser Alalwan, and Mohamed Sarrab. Mobile cloud computing: advantage, disadvantage and open challenge. In *Proceedings of the 7th Euro American Conference on Telematics and Information Systems*, page 21. ACM, 2014.

[AFG$^+$10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, April 2010.

[AGH18] Khadija Akherfi, Micheal Gerndt, and Hamid Harroud. Mobile cloud computing for computation offloading: Issues and challenges. *Applied computing and informatics*, 14(1):1–16, 2018.

[AK15] Mohammad Oqail Ahmad and Rafiqul Zaman Khan. The Cloud Computing: A Systematic Review. *International Journal of Innovative Research in Computer and Communication Engineering*, 3(5):4066–4075, May 2015.

[Akt19] Ismet Aktas. Cloud and edge computing for IoT: a short history. `https://blog.bosch-si.com/bosch-iot-suite/cloud-and-edge-computing-for-iot-a-short-history/`, 2019. Accessed on 25.04.2019.

[Ama19] Amazon Web Services. Amazon EC2 Instance Types. `https://aws.amazon.com/ec2/instance-types/`, 2019. Accessed on 28.01.2020.

[Ano] Anonymous. What is Advanced Mobile Phone Service (AMPS)? `https://networkencyclopedia.com/advanced-mobile-phone-service-amps/`. Accessed on 03.12.2019.

[Ano12] Anonymous. Big-O Cheat Sheet. `https://www.bigocheatsheet.com/`, 2012. Accessed on 19.12.2019.

[Aug19] Stephan Augsten. Was ist eine UUID? https://www.dev-insider.de/was-ist-eine-uuid-a-788491/, 2019. Accessed on 19.12.2019.

[BA05] Patrik Berander and Anneliese Andrews. Requirements prioritization. In *Engineering and managing software requirements*, pages 69–94. Springer, 2005.

[BCINN14] Jerry Banks, John S. Carson II, Barry L. Nelson, and David M. Nicol. *Discrete-Event System Simulation.* Pearson Education Limited, 2014.

[Bea] Vangie Beal. Latency. https://www.webopedia.com/TERM/L/latency.html. Accessed on 24.03.2019.

[BLRK11] Markus Böhm, Stefanie Leimeister, Christoph Riedl, and Helmut Krcmar. Cloud computing and computing evolution. *Cloud Computing Technologies, Business Models, Opportunities and Challenges*, 01 2011.

[BOE17] Ahmet Cihat Baktir, Atay Ozgovde, and Cem Ersoy. How can edge computing benefit from software-defined networking: A survey, use cases, and future directions. *IEEE Communications Surveys & Tutorials*, 19(4):2359–2391, 2017.

[Bra] BrainKart. Characteristics of Mobile Computing. https://www.brainkart.com/article/Characteristics-of-Mobile-Computing_9875/. Accessed on 03.12.2019.

[BT15] Nils Backhaus and Manfred Thüring. Trust in cloud computing: pro and contra from the user's point of view. *i-com*, 14(3):231–243, 2015.

[Buc18] James Bucki. The Benefits of Mobile Computing. https://www.thebalancesmb.com/definition-of-mobile-computing-2533640, 2018. Accessed on 03.12.2019.

[CBC+10] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. In *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pages 49–62, 2010.

[CH18] Min Chen and Yixue Hao. Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, 36(3):587–597, 2018.

[CHQ+16] Min Chen, Yixue Hao, Meikang Qiu, Jeungeun Song, Di Wu, and Iztok Humar. Mobility-aware caching and computation offloading in 5g ultra-dense cellular networks. *Sensors*, 16(7):974, 2016.

[CIM+11] Byung-Gon Chun, Sunghwan Ihm, Petros Maniatis, Mayur Naik, and Ashwin Patti. Clonecloud: elastic execution between mobile device and cloud. In *Proceedings of the sixth conference on Computer systems*, pages 301–314, 2011.

[Cis16] Cisco Systems Inc. Cisco global cloud index: Forecast and methodology 2015–2020. *White paper*, 2016.

[CJLF15] Xu Chen, Lei Jiao, Wenzhong Li, and Xiaoming Fu. Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, 24(5):2795–2808, 2015.

[Clo20] CloudHarmony. Amazon Web Services Network Test. `http://cloudharmony.com/speedtest-latency-for-aws:ec2`, 2020. Accessed on 28.01.2020.

[CN13] Nitin R Chopde and M Nichat. Landmark based shortest path detection by using a* and haversine formula. *International Journal of Innovative Research in Computer and Communication Engineering*, 1(2):298–302, 2013.

[Cox12] Christopher Cox. *An introduction to LTE: LTE, LTE-advanced, SAE and 4G mobile communications.* John Wiley & Sons, 2012.

[CVdMK11] Mariana Carroll, Alta Van der Merwe, and Paula Kotzé. Secure cloud computing: Benefits, risks and controls. In *Information Security for South Africa*, pages 1–9, Aug 2011.

[dm20] d maps.com. Map Vienna (Austria). `https://d-maps.com/carte.php?num_car=34276&lang=en`, 2020. Accessed on 22.01.2020.

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

[Duk19] Duke University Medical Center. Systematic Reviews: the process: Grey Literature. `https://guides.mclibrary.duke.edu/sysreview/greylit`, 2019. Accessed on 27.11.2019.

[Fea20] Feather. Simply beautiful open source icons. `https://feathericons.com/?query=marker`, 2020. Accessed on 22.01.2020.

[Fin20] Roger Fingas. iPhone XS has 4GB of RAM, 2.49 GHz A12 chip according to benchmarks. `https://appleinsider.com/articles/18/09/13/iphone-xs-has-4gb-of-ram-249-ghz-a12-chip-according-to-benchmarks`, 2020. Accessed on 23.01.2020.

[Fou16] Apache Software Foundation. Class MersenneTwister. https://commons.apache.org/proper/commons-math/javadocs/api-3.6/org/apache/commons/math3/random/MersenneTwister.html, 2016. Accessed on 31.01.2020.

[GA20] 5G-Anbieter.info. 5G Upload. https://www.5g-anbieter.info/speed/5g-upload.html, 2020. Accessed on 28.01.2020.

[GHMP08] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[God18] William Goddard. The Evolution of Cloud Computing – Where's It Going Next? https://www.itchronicles.com/cloud/the-evolution-of-cloud-computing-wheres-it-going-next/, 2018. Accessed on 24.04.2019.

[GTM+16] Xiaohu Ge, Song Tu, Guoqiang Mao, Cheng-Xiang Wang, and Tao Han. 5g ultra-dense cellular networks. *IEEE Wireless Communications*, 23(1):72–79, 2016.

[HA18] Rob J. Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[Ham18] Eric Hamilton. What is Edge Computing: The Network Edge Explained. https://www.cloudwards.net/what-is-edge-computing/, 2018. Accessed on 24.04.2019.

[Has13] Samah Ahmed Zaki Hassan. Sona: A service oriented nodes architecture for developing cloud computing applications. In *2013 International Conference on Advanced Computing and Communication Systems*, pages 1–6. IEEE, 2013.

[HPS+15] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing – a key technology towards 5g. *ETSI white paper*, 11(11):1–16, 2015.

[IB93] Tomasz Imieliński and B. R. Badrinath. Mobile wireless computing: Solutions and challenges in data management. *Computer Science*, 1993.

[IFB+17] Michaela Iorga, Larry Feldman, Robert Barton, Michael Martin, Nedim Goren, and Charif Mahmoudi. The NIST definition of fog computing. Technical report, National Institute of Standards and Technology, 2017.

[Inc20] Chegg Inc. We have solutions for your book! https://www.chegg.com/homework-help/computer-organization-and-architecture-10th-edition-chapter-2-sol 2020. Accessed on 24.01.2020.

[ISA10] ISACA. Securing mobile devices. Technical report, ISACA, 2010.

[JTZ+14] Sam Johnson, Nick Twilley, Tianyi Zhang, Zhanni Zhou, and Suijun Wu. Mobile computing - a look at concepts, problems, and solutions, 2014.

[KL07] Yu-Kwong Ricky Kwok and Vincent KN Lau. *Wireless Internet and mobile computing: interoperability and performance*, volume 89. John Wiley & Sons, 2007.

[KL10] K. Kumar and Y. Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, 43(04):51–56, apr 2010.

[KQA14] Aamir Khan, Nauman A Qureshi, and Umair Abdullah. Software engineering challenges: A cloud based architecture for earthquake forecasting. *Science International*, 26(5), 2014.

[Lö17] Sylvia Lösel. Was ist Mobile Computing? `https://www.it-business.de/was-ist-mobile-computing-a-634341/`, 2017. Accessed on 03.12.2019.

[LA20] LTE-Anbieter.info. Wie schnell ist eigentlich LTE | 4G? `https://www.lte-anbieter.info/ratgeber/schnelligkeit-lte.php`, 2020. Accessed on 23.01.2020.

[LBP17] Chen-Feng Liu, Mehdi Bennis, and H Vincent Poor. Latency and reliability-aware task offloading and resource allocation for mobile edge computing. In *2017 IEEE Globecom Workshops (GC Wkshps)*, pages 1–7. IEEE, 2017.

[Liu17] Marco Liu. Optimal number of trials for Monte Carlo simulation. *VRC–Valuation Research Report*, 2017.

[Liv13] David J. Livingston. Introduction & history of mobile computing. `https://de.slideshare.net/davidjlivi/introduction-history-of-mobile-computing`, 2013. Accessed on 03.12.2019.

[LLWT15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[Lo18] Ken Lo. Download Speeds: What Do 2G, 3G, 4G & 5G Actually Mean? `https://kenstechtips.com/index.php/download-speeds-2g-3g-and-4g-actual-meaning`, 2018. Accessed on 26.04.2019.

[lte18] lte-anbieter.info. 5G: Alles zum LTE-Nachfolger der Zukunft. `https://www.lte-anbieter.info/5g/`, 2018. Accessed on 26.04.2019.

[Mac18] Meghan MacDonald. More buying, less building in The Age of Consumption. `https://451research.com/blog/1933-more-buying-less-building-in-the-age-of-consumption`, 2018. Accessed on 24.04.2019.

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

[McI93] Peter McIlroy. Optimistic sorting and information theoretic complexity. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 467–474. Society for Industrial and Applied Mathematics, 1993.

[MG+11] Peter Mell, Tim Grance, et al. The NIST definition of cloud computing. Technical report, National Institute of Standards and Technology, 2011.

[MHS15] Sune Dueholm Müller, Stefan Rubæk Holm, and Jens Søndergaard. Benefits of cloud computing: Literature review in a maturity model perspective. *Communications of the Association for Information Systems*, 37(1):42, 2015.

[Mic15] Microsoft. What is Infrastructure as a Service? `https://social.technet.microsoft.com/wiki/contents/articles/4633.what-is-infrastructure-as-a-service.aspx`, 2015. Accessed on 22.11.2019.

[MK16] N. Mohan and J. Kangasharju. Edge-fog cloud: A distributed cloud for internet of things computations. In *2016 Cloudification of the Internet of Things (CIoT)*, pages 1–6, Nov 2016.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[MTC+16] C MacGillivray, Vernon Turner, RY Clarke, J Feblowitz, K Knickle, L Lamy, M Xiang, A Siviero, and M Cansfield. Idc future scape: Worldwide internet of things 2017 predictions. In *IDC Web Conference*, 2016.

[MYZ+17] Yuyi Mao, Changsheng You, Jun Zhang, Kaibin Huang, and Khaled B. Letaief. A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys & Tutorials*, 19(4):2322–2358, 2017.

[NCT] NCTA. Growth in the Internet of Things. `https://www.ncta.com/broadband-by-the-numbers`. Accessed on 24.04.2019.

[OnL11] OnLogic. Near Real Time. `https://wiki.c2.com/?NearRealTime`, 2011. Accessed on 26.03.2020.

[OnL19] OnLogic. High-Performance Industrie-Edge-Mini-Server mit Lüfter und Xeon. `https://www.onlogic.com/de-de/mc850-54/`, 2019. Accessed on 24.01.2020.

[Ora19] Oracle. Class Arrays. `https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html#sort-T:A-java.util.Comparator-`, 2019. Accessed on 19.12.2019.

[PFMM08] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. Systematic mapping studies in software engineering. In *Ease*, volume 8, pages 68–77, 2008.

[Pie01] Samuel Pierre. Mobile computing and ubiquitous networking: concepts, technologies and challenges. *Telematics and Informatics*, 18(2-3):109–131, 2001.

[PKKB09] Nicholas Palmer, Roelof Kemp, Thilo Kielmann, and Henri Bal. Ibis for mobility: solving challenges of mobile computing using grid techniques. In *Proceedings of the 10th workshop on Mobile Computing Systems and Applications*, pages 1–6, 2009.

[PMRT16] Jianli Pan, Lin Ma, Ravishankar Ravindran, and Peyman TalebiFard. Homecloud: An edge cloud framework and testbed for new application delivery. In *2016 23rd International Conference on Telecommunications (ICT)*, pages 1–6. IEEE, 2016.

[Pre] Cambridge University Press. Meaning of on demand in English. `https://dictionary.cambridge.org/dictionary/english/on-demand`. Accessed on 16.10.2019.

[PT04] H. H. Pang and K. L. Tan. Authenticating query results in edge computing. In *Proceedings. 20th International Conference on Data Engineering*, pages 560–571. IEEE, March 2004.

[RF 12] RF Wireless World. Network Latency Calculator | Network Latency Formula. `https://www.rfwireless-world.com/calculators/Network-Latency-Calculator.html`, 2012. Accessed on 25.12.2019.

[Rie18] Stefan Ried. IoT Edge – Von Gateway bis Machine Learning. `https://www.computerwoche.de/a/iot-edge-von-gateway-bis-machine-learning,3545872`, 2018. Accessed on 25.04.2019.

[Roe17] Matthias Roese. IT für das Internet der Dinge: Industrie 4.0 braucht Edge Computing. https://www.digital-engineering-magazin.de/it-fuer-das-internet-der-dinge-industrie-40-braucht-edge-computi 2017. Accessed on 24.03.2019.

[Rou05] Margaret Rouse. Peer-to-Peer (P2P). https://www.computerweekly.com/de/definition/Peer-to-Peer-P2P, 2005. Accessed on 25.04.2019.

[Rou06] Margaret Rouse. Advanced Mobile Phone Service (AMPS). https://searchmobilecomputing.techtarget.com/definition/Advanced-Mobile-Phone-Service, 2006. Accessed on 03.12.2019.

[RSM+13] T. S. Rappaport, S. Sun, R. Mayzus, H. Zhao, Y. Azar, K. Wang, G. N. Wong, J. K. Schulz, M. Samimi, and F. Gutierrez. Millimeter wave mobile communications for 5g cellular: It will work! *IEEE Access*, 1:335–349, 2013.

[RYCH18] Jinke Ren, Guanding Yu, Yunlong Cai, and Yinghui He. Latency optimization for resource allocation in mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 17(8):5506–5519, 2018.

[SBCD09] Mahadev Satyanarayanan, Victor Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 2009.

[Sch18] Leah Schoeb. Cloud Scalability: Scale Up vs Scale Out. https://blog.turbonomic.com/blog/on-technology/cloud-scalability-scale-vs-scale, 2018. Accessed on 24.04.2019.

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

[Sem16] Frank Sempert. Edge computing für das internet der dinge. https://www.computerwoche.de/a/edge-computing-fuer-das-internet-der-dinge,3226335, 2016. Accessed on 13.07.2018.

[Set16] Ayob Sether. Cloud Computing Benefits, June 2016.

[SGFW10] Harald Sundmaeker, Patrick Guillemin, Peter Friess, and Sylvie Woelfflé. Vision and challenges for realising the Internet of Things. *Cluster of European Research Projects on the Internet of Things, European Commision*, 3(3):34–36, 2010.

[Ski98] Steven S. Skiena. *The algorithm design manual*, volume 1. Springer Science & Business Media, 1998.

[SNKS19] Simar Preet Singh, Anand Nayyar, Rajesh Kumar, and Anju Sharma. Fog computing: from architecture to edge computing and big data processing. *The Journal of Supercomputing*, 75(4):2070–2105, 2019.

[Sto02] Ivan Stojmenović, editor. *Handbook of Wireless Networks and Mobile Computing*. John Wiley & Sons, Inc, 2002.

[Tec] Techopedia. Directed Acyclic Graph (DAG). `https://www.techopedia.com/definition/5739/directed-acyclic-graph-dag`. Accessed on 17.12.2019.

[TKS05] Vlasios Tsiatsis, Ram Kumar, and Mani B. Srivastava. Computation Hierarchy for In-Network Processing. *Mobile Networks and Applications*, 10(4):505–518, Aug 2005.

[TOD+17] Xiaoyi Tao, Kaoru Ota, Mianxiong Dong, Heng Qi, and Keqiu Li. Performance guaranteed computation offloading for mobile-edge cloud computing. *IEEE Wireless Communications Letters*, 6(6):774–777, 2017.

[TP18] Tuyen X Tran and Dario Pompili. Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, 68(1):856–868, 2018.

[TSB+19] Klervie Toczé, Norbert Schmitt, Ivona Brandic, Atakan Aral, and Simin Nadjm-Tehrani. Towards Edge Benchmarking: A Methodology for Characterizing Edge Workloads. In *2019 IEEE 4th International Workshops on Foundations and Applications of Self* Systems (FAS* W)*, pages 70–71. IEEE, 2019.

[Vel19] Velotio Technologies. A Beginner's Guide to Edge Computing. `https://medium.com/velotio-perspectives/a-beginners-guide-to-edge-computing-6cfea853aa11`, 2019. Accessed on 25.04.2019.

[Ven19] Chris Veness. Calculate distance, bearing and more between Latitude/-Longitude points. `https://www.movable-type.co.uk/scripts/latlong.html`, 2019. Accessed on 18.12.2019.

[Ves10] Mikko Vestola. A comparison of nine basic techniques for requirements prioritization. *Helsinki University of Technology*, pages 1–8, 2010.

[Wan16a] Wolfgang Wanner. Cloud- versus Edge-Computing Part 1. `http://www.funkschau.de/telekommunikation/artikel/130928/`, 2016. Accessed on 24.04.2019.

[Wan16b] Wolfgang Wanner. Cloud- versus Edge-Computing Part 2. `https://www.funkschau.de/telekommunikation/artikel/130928/1/`, 2016. Accessed on 24.04.2019.

[WH16] Karl-Heinz Waldmann and Werner E. Helm. *Simulation stochastischer Systeme*. Springer, 2016.

[YH16] Changsheng You and Kaibin Huang. Multiuser resource allocation for mobile-edge computation offloading. In *2016 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2016.

[YHCK16] Changsheng You, Kaibin Huang, Hyukjin Chae, and Byoung-Hoon Kim. Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, 16(3):1397–1411, 2016.

[ZCB10] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

[ZN10] Pei Zheng and Lionel Ni. *Smart phone and next generation mobile computing*. Elsevier, 2010.

[Zor19] Peter Zornio. Cloud Versus Edge Computing – What's The Difference? `https://www.forbes.com/sites/forbestechcouncil/2019/02/22/cloud-versus-edge-computing-whats-the-difference/#605158ff8dc5`, 2019. Accessed on 25.11.2019.

[ZZGN17] Tianchu Zhao, Sheng Zhou, Xueying Guo, and Zhisheng Niu. Tasks scheduling and resource allocation in heterogeneous cloud for delay-bounded mobile edge computing. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2017.

Appendix A

Conduct Search and Screening of Papers

Cloud Computing Benefits

ResearchGate

(https://www.researchgate.net/)

Search string: "cloud computing benefits" - results from 21.11.2019

- https://www.researchgate.net/publication/314530281_Cloud_Computing_Benefits
- https://www.researchgate.net/publication/254028415_Cloud_Computing_Benefits_and_Architecture_in_E-Learning -> is kept because it discusses access of cloud services via mobile devices
- https://www.researchgate.net/publication/310952332_Information_Systems_Security_on_Cloud_Computing_Benefits_Risks_Security_Considerations_Recommended_Model
- https://www.researchgate.net/publication/319302104_Using_cloud_computing_services_in_e-learning_process_Benefits_and_challenges -> is kept because there is one part that describes cloud computing benefits in general
- https://www.researchgate.net/publication/292946638_A_Unified_Forensic_Framework_for_Data_Identification_and_Collection_in_Mobile_Cloud_Social_Network_Applications
- https://www.researchgate.net/publication/284593673_Access_control_and_user_authentication_concerns_in_cloud_computing_environments -> is kept because there is one part that describes cloud computing benefits in general
- https://www.researchgate.net/publication/280302501_The_Cloud_Computing_A_Systematic_Review

- https://www.researchgate.net/publication/290459511_Cloud_computing_audit -> is kept because there is one part that describes cloud computing benefits in general

- https://www.researchgate.net/publication/279992596_Benefits_of_Cloud_Computing_Literature_Review_in_a_Maturity_Model_Perspective

- https://www.researchgate.net/publication/241811966_Red_Skies_in_the_Morning-Professional_Ethics_at_the_Dawn_of_Cloud_Computing

- https://www.researchgate.net/publication/224259118_Secure_cloud_computing_Benefits_risks_and_controls

Search string: "cloud computing advantages" - results from 21.11.2019

- https://www.researchgate.net/publication/280331907_A_Survey_of_Mobile_Cloud_Computing_Advantages_Challenges_and_Approaches

- https://www.researchgate.net/publication/287109681_Cloud_Service_Trust_Model_and_Its_Application_Research_Based_on_the_Third_Party_Certification -> is kept because there is one part that describes cloud computing advantages in general

- https://www.researchgate.net/publication/269838335_Software_Engineering_Challenges_A_Cloud_Based_Architecture_For_Earthquake_Forecasting -> is kept because there is one part that describes cloud computing advantages in general

- https://www.researchgate.net/publication/257139857_The_Organizational_Critical_Success_Factors_for_Adopting_Cloud_Computing_in_SMEs -> is kept because there is one part that describes cloud computing advantages in general

Search string: "cloud computing pros" - results from 21.11.2019

- https://www.researchgate.net/publication/286479415_Trust_in_Cloud_Computing_Pro_and_Contra_from_the_User%27s_Point_of_View -> is kept because there is one part that describes cloud computing pros in general

Search string: "Cloud computing for mobile users" (already known paper) - results from 21.11.2019

- https://www.researchgate.net/publication/220475756_Lu_Y-H_Yung-Hsiang_Lu_Cloud_Computing_for_Mobile_Users_Can_Offloading_Computation_Save_Energy_Computer_434_51-56 -> is kept because there is one part that describes cloud computing benefits in general

IEEE
(`https://ieeexplore.ieee.org/search/advanced`)

Search string: "cloud computing benefits" - results from 21.11.2019

- Cloud Computing Benefits and Architecture in E-Learning (`https://ieeexplore.ieee.org/document/6185026`)
- Access control and user authentication concerns in cloud computing environments (`https://ieeexplore.ieee.org/document/7289572`) -> is kept because there is one part that describes cloud computing benefits in general

Search string: "cloud computing advantages" - results from 21.11.2019

- SONA: A service oriented nodes architecture for developing Cloud Computing applications (`https://ieeexplore.ieee.org/document/6938769`) -> is kept because there is one part that describes cloud computing advantages in general

Search string: "cloud computing pros" - results from 21.11.2019

- no results found

ACM DL
(`https://dl.acm.org/`)

Search string: "(+"cloud computing benefits")" - results from 21.11.2019

- no results after applying inclusion and exclusion criteria

Search string: "(+"cloud computing advantages")" - results from 21.11.2019

- Mobile cloud computing: advantage, disadvantage and open challenge (`https://dl.acm.org/citation.cfm?id=2590670`)

Search string: "(+"cloud computing pros")" - results from 21.11.2019

- no results found

Search string: "(+"a view of cloud computing")" (already known paper) - results from 21.11.2019

- A view of cloud computing (`https://dl.acm.org/citation.cfm?id=1721672`) -> includes benefits

Google
(https://www.google.com/)

Search string: "[intitle:"Cloud- versus Edge-Computing"]" - results from 25.11.2019

- Cloud Versus Edge Computing – What's The Difference? (https://www.forbes.com/sites/forbestechcouncil/2019/02/22/cloud-versus-edge-computing\-whats-the-difference/#605158ff8dc5)
- Cloud- versus Edge-Computing (https://www.funkschau.de/office-kommunikation/cloud-versus-edge-computing.130928.html)

Cloud Computing Limitations

ResearchGate
(https://www.researchgate.net/)

Search string: "cloud computing limitations" - results from 21.11.2019

- no results after applying inclusion and exclusion criteria

Search string: "cloud computing disadvantages" - results from 21.11.2019

- no results after applying inclusion and exclusion criteria

Search string: "cloud computing cons" - results from 21.11.2019

- no results after applying inclusion and exclusion criteria

Search string: "cloud computing drawbacks" - results from 21.11.2019

- https://www.researchgate.net/publication/329157524_Fog_computing_from_architecture_to_edge_computing_and_big_data_processing -> is kept because there is one part that addresses drawbacks of cloud computing

Search string: "Cloud Computing Networking Challenges" (already known paper) - results from 25.11.2019

- https://www.researchgate.net/publication/249325475_Cloud_Computing_Networking_Challenges_and_Opportunities_for_Innovations -> includes limitations of cloud computing

Search string: "Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations" (already known paper) - results from 25.11.2019

- https://www.researchgate.net/publication/313879300_Edge-Fog_Cloud_A_Distributed_Cloud_for_Internet_of_Things_Computations -> includes limitations of cloud computing

IEEE
(https://ieeexplore.ieee.org/search/advanced)

Search string: "cloud computing limitations" - results from 21.11.2019

- Opportunistic fog computing: Feasibility assessment and architectural proposal (https://ieeexplore.ieee.org/document/7987320) -> is kept because there is one part that addresses cloud computing limitations

Search string: "cloud computing disadvantages" - results from 21.11.2019

- no results after applying inclusion and exclusion criteria

Search string: "cloud computing cons" - results from 21.11.2019

- no results found

Search string: "cloud computing drawbacks" - results from 21.11.2019

- Opportunistic fog computing: Feasibility assessment and architectural proposal (https://ieeexplore.ieee.org/document/7987320) -> is kept because there is one part that addresses cloud computing limitations

ACM DL
(https://dl.acm.org/)

Search string: "(+"cloud computing limitations")" - results from 21.11.2019

- no results after applying inclusion and exclusion criteria

Search string: "(+"cloud computing disadvantages")" - results from 21.11.2019

- no results found

Search string: "(+"cloud computing cons")" - results from 21.11.2019

- no results found

Search string: "(+"cloud computing drawbacks")" - results from 21.11.2019

- no results found

Google
(https://www.google.com/)

Search string: "cloud computing drawbacks" - results from 25.11.2019

- Cloud Versus Edge Computing – What's The Difference? (https://www.forbes.com/sites/forbestechcouncil/2019/02/22/cloud-versus-edge-computing\-whats-the-difference/#605158ff8dc5)
- Cloud- versus Edge-Computing (https://www.funkschau.de/office-kommunikation/cloud-versus-edge-computing.130928.html)

Edge Computing Limitations

ResearchGate
(https://www.researchgate.net/)

Search string: "edge computing benefits" - results from 25.11.2019

- https://www.researchgate.net/publication/317701794_How_Can_Edge_Computing_Benefit_from_Software-Defined_Networking_A_Survey_Use_Cases_Future_Directions -> is kept because "Benefit areas" are discussed

Search string: "edge computing advantages" - results from 25.11.2019

- https://www.researchgate.net/publication/320806141_Cloud-based_IoT_solution_for_State_Estimation_in_Smart_Grids_exploiting_virtualization_and_edge-intelligence_technologies -> is kept because there is one part that describes edge computing advantages

Search string: "edge computing pros" - results from 25.11.2019

- no results found

Search string: "Edge computing - Vision and Challenges" (already known paper) - results from 25.11.2019

- https://www.researchgate.net/publication/303890546_Edge_Computing_Vision_and_Challenges -> -> is kept because there is one part that describes edge computing benefits

IEEE
(https://ieeexplore.ieee.org/search/advanced)

Search string: "edge computing benefits" - results from 25.11.2019

- Edge computing framework for distributed smart applications (https://ieeexplore.ieee.org/document/8397448) -> is kept because there is one part that describes edge computing benefits

Search string: "edge computing advantages" - results from 25.11.2019

- no results found

Search string: "edge computing pros" - results from 25.11.2019

- no results found

ACM DL
(https://dl.acm.org/)

Search string: "(+"edge computing benefits")" - results from 25.11.2019

- no results after applying inclusion and exclusion criteria

Search string: "(+"edge computing advantages")" - results from 25.11.2019

- no results found

Search string: "(+"edge computing pros")" - results from 25.11.2019

- no results found

Google
(https://www.google.com/)

Search string: "[intitle:"Cloud- versus Edge-Computing"]" - results from 25.11.2019

- Cloud Versus Edge Computing – What's The Difference? (https://www.forbes.com/sites/forbestechcouncil/2019/02/22/cloud-versus-edge-computing\-whats-the-difference/#605158ff8dc5)
- Cloud- versus Edge-Computing (https://www.funkschau.de/office-kommunikation/cloud-versus-edge-computing.130928.html)

Search string: "[intitle:"IoT Edge - Von Gateway bis Machine Learning"]" (already known article) - results from 25.11.2019

- IoT Edge – Von Gateway bis Machine Learning (https://www.crisp-research.com/iot-edge-von-gateway-bis-machine-learning/)

Edge Computing Limitations

ResearchGate

(`https://www.researchgate.net/`)

Search string: "edge computing limitations" - results from 25.11.2019

- no results found

Search string: "edge computing disadvantages" - results from 25.11.2019

- no results found

Search string: "edge computing cons" - results from 25.11.2019

- no results found

Search string: "edge computing drawbacks" - results from 25.11.2019

- no results found

IEEE

(`https://ieeexplore.ieee.org/search/advanced`)

Search string: "edge computing limitations" - results from 25.11.2019

- no results found

Search string: "edge computing disadvantages" - results from 25.11.2019

- no results found

Search string: "edge computing cons" - results from 25.11.2019

- no results found

Search string: "edge computing drawbacks" - results from 25.11.2019

- no results found

ACM DL

(`https://dl.acm.org/`)

Search string: "(+"edge computing limitations")" - results from 25.11.2019

- no results found

Search string: "(+"edge computing disadvantages")" - results from 25.11.2019

- no results found

Search string: "(+"edge computing cons")" - results from 25.11.2019

- no results found

Search string: "(+"edge computing drawbacks")" - results from 25.11.2019

- no results found

Google
(https://www.google.com/)

Search string: "[intitle:"Cloud- versus Edge-Computing"]" - results from 25.11.2019

- Cloud Versus Edge Computing – What's The Difference? (https://www.forbes.com/sites/forbestechcouncil/2019/02/22/cloud-versus-edge-computing\-whats-the-difference/#605158ff8dc5)
- Cloud- versus Edge-Computing (https://www.funkschau.de/office-kommunikation/cloud-versus-edge-computing.130928.html)

Data Extraction and Mapping Process

In the following results of the systematic mapping study we have used different colors:

- Grey: normal extracted results
- Red: results have also been published within another journal
- Blue: no access gained by authors
- Green: information has been extracted from a result from a different search topic

Cloud Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Cloud Computing Benefits	Ayob Sether	rent computing infrastructure including virtual machines, operating systems, middleware, applications, network and other infrastructure	scalability	8	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	vendors are able to scale and allocate cloud computing resources on demand to develop systems meeting customers' need quickly	scalability	8	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	vendor focuses on service covering the performance and availability of the infrastructure	resiliency / availability	8	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	CPU, memory, storage, network and other resources can be selected as per application requirements on subscription based from IaaS vendor as per need basis whenever required.	scalability	8	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	primary benefits of the cloud computing is significantly low cost compared to traditional computing cost	IT cost reduction	9	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	reduce IT infrastructure cost and reduced IT operations costs	IT cost reduction	9	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	Mobility benefits of cloud computing make it perfect for anytime and anywhere access	mobility	9	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	Cloud computing offers better scalability and its resources are more on-demand and need basis, can meet growing need of storage and network resources allows to develop, deploy and run applications on reliable cloud computing clusters that rarely fail	scalability	10	ResearchGate	22.11.2019
Cloud Computing Benefits	Ayob Sether	Scalability is one of the main benefits of cloud computing as scalable solutions are implemented quickly and resources can be utilized whenever required saving time, saving money and increasing efficiency.	scalability	10	ResearchGate	22.11.2019
Cloud Computing Benefits and Architecture in E-Learning	Aida Ghazizadeh	x	x	x	ResearchGate	22.11.2019
Information Systems Security on Cloud Computing (Benefits, Risks, Security Considerations, Recommended Model)	Mohammad Saidur Rahman	Cloud computing reduces costs of IT systems management and maintenance	IT cost reduction	11	ResearchGate	22.11.2019

Cloud Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Information Systems Security on Cloud Computing (Benefits, Risks, Security Considerations, Recommended Model)	Mohammad Saidur Rahman	consumer can purchase as much or as little computing power as they need. Cloud computing helps organizations to scale up or scale down operations and storage needs quickly to suit situation	scalability	11	ResearchGate	22.11.2019
Information Systems Security on Cloud Computing (Benefits, Risks, Security Considerations, Recommended Model)	Mohammad Saidur Rahman	Cloud computing offers flexibility in work. A client gets the access to data from anywhere he wants (i.e. home, offsite, office) and anytime (via the Internet connection)	mobility	11	ResearchGate	22.11.2019
Information Systems Security on Cloud Computing (Benefits, Risks, Security Considerations, Recommended Model)	Mohammad Saidur Rahman	The potentiality of failure in a highly resilient computing environment is reduced	resiliency / availability	11	ResearchGate	22.11.2019
Using cloud computing services in e-learning process: Benefits and challenges	Abderrahim El Mhouti, Mohamed Erradi, Azeddine Nasseh	??			ResearchGate	22.11.2019
A Unified Forensic Framework for Data Identification and Collection in Mobile Cloud Social Network Applications	Muhammad Faheem, Dr Tahar, Dr An	x	x	x	ResearchGate	22.11.2019
Access control and user authentication concerns in cloud computing environments	Mohammad Javad Golkar, Mohammad Ahmadi, Mohammad Chizari, Mohammad Eslami, Mostafa Vali	x	x	x	ResearchGate	22.11.2019
The Cloud Computing: A Systematic Review	Mohammad Oqail Ahmad, Rafiqul Zaman KhanRafiqul Zaman Khan	cost of managing and maintaining has to shift toward the resources of cloud computing vendor	IT cost reduction	9	ResearchGate	22.11.2019
The Cloud Computing: A Systematic Review	Mohammad Oqail Ahmad, Rafiqul Zaman KhanRafiqul Zaman Khan	users have ability to access data anywhere and anytime such as from home, on holiday in the world	mobility	9	ResearchGate	22.11.2019
The Cloud Computing: A Systematic Review	Mohammad Oqail Ahmad, Rafiqul Zaman KhanRafiqul Zaman Khan	devices which are applicable include laptop, desktop, smart phone etc. with internet connection	mobility	9	ResearchGate	22.11.2019
Cloud computing audit	Valentin Sgarciu	??			ResearchGate	22.11.2019
Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective	Sune Dueholm Müller, S.R. Holm, Jens Søndergaard	cloud computing can be a valuable tool in reducing both investment costs (CAPEX) and ongoing operation costs (OPEX)	IT cost reduction	12	ResearchGate	22.11.2019

Cloud Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective	Sune Dueholm Müller, S.R. Holm, Jens Søndergaard	Cloud computing also offers cost savings in terms of batch processing of data, the so-called "cost associativity" concept, because computations are done faster at the same cost. For example, using 1000 computers for one hour might cost the same as using one computer for 1000 hours (Armbrust et al., 2010; Li et al., 2012; Marston et al., 2011), which ultimately enables organizations to perform computations more cost-effectively.	IT cost reduction	12	ResearchGate	22.11.2019
Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective	Sune Dueholm Müller, S.R. Holm, Jens Søndergaard	improving its technical infrastructure by shortening computational lead time and, subsequently, optimizing process efficiency	scalability	13	ResearchGate	22.11.2019
Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective	Sune Dueholm Müller, S.R. Holm, Jens Søndergaard	automation to handle tasks concurrently, which reduced the overall processing time and the potential waiting time for system users	computational power	13	ResearchGate	22.11.2019
Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective	Sune Dueholm Müller, S.R. Holm, Jens Søndergaard	cloud computing into courses gives students greater flexibility	mobility	13	ResearchGate	22.11.2019
Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective	Sune Dueholm Müller, S.R. Holm, Jens Søndergaard	technical scalability offered by virtualization, whereas resource use is optimized across both peak and off-peak periods	scalability	14	ResearchGate	22.11.2019
Benefits of Cloud Computing: Literature Review in a Maturity Model Perspective	Sune Dueholm Müller, S.R. Holm, Jens Søndergaard	allocate IT resources to specific business processes whenever needed	scalability	16	ResearchGate	22.11.2019
Red Skies in the Morning - Professional Ethics at the Dawn of Cloud Computing	Sarah Jane Hughes, Roland L. TropeRoland L. Trope	x	x	x	ResearchGate	22.11.2019
A Survey of Mobile Cloud Computing: Advantages, Challenges and Approaches	Mohammad Rasoul Momeni	can access and use prepared services on-demand	scalability	4	ResearchGate	22.11.2019
Cloud Service Trust Model and Its Application Research Based on the Third Party Certification	Haiyong Xu, Junhai MaJunhai Ma, Lei Li, Xueli Zhan	??			ResearchGate	22.11.2019
Software Engineering Challenges: A Cloud Based Architecture For Earthquake Forecasting	Aamir Khan, Nauman A. QureshiNauman A. Qureshi, Umair AbdullahUmair Abdullah	having peaks and troughs of demands, scalability is the central requirement -> Cloud computing provides the solution towards this problem by providing on-demand access to IT resources	scalability	1	ResearchGate	22.11.2019
Software Engineering Challenges: A Cloud Based Architecture For Earthquake Forecasting	Aamir Khan, Nauman A. QureshiNauman A. Qureshi, Umair AbdullahUmair Abdullah	reducing capital expenditures (CapEx). Cloud computing consumers also reduce the operation expenditures (OpEx) as maintenance is carried by cloud service providers.	IT cost reduction	2	ResearchGate	22.11.2019

Cloud Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Software Engineering Challenges: A Cloud Based Architecture For Earthquake Forecasting	Aamir Khan, Nauman A. QureshiNauman A. Qureshi, Umair AbdullahUmair Abdullah	Cloud computing has provided anytime, anywhere access facility	mobility	2	ResearchGate	22.11.2019
The Organizational Critical Success Factors for Adopting Cloud Computing in SMEs	Azam Abdollahzadegan, Ab Razak Che Hussin, Marjan Moshfegh Gohary, Mahyar AminiMahyar Amini	x	x	x	ResearchGate	22.11.2019
Trust in Cloud Computing: Pro and Contra from the User's Point of View	Nils Backhaus, Manfred ThueringManfred Thuering	any conceivable amount of resources can be made available as required for a specific period	scalability	4	ResearchGate	26.11.2019
Trust in Cloud Computing: Pro and Contra from the User's Point of View	Nils Backhaus, Manfred ThueringManfred Thuering	use-dependent payment (pay-as-you-go) allows for an extremely cost-efficient use of the resources.	IT cost reduction	4	ResearchGate	26.11.2019
Secure cloud computing: Benefits, risks and controls	Mariana Carroll, Alta Van der Merwe, Paula Kotzé	provides compelling savings in IT related costs including lower implementation and maintenance costs; less hardware to purchase and support; the elimination of the cost of power, cooling, floor space and storage as resources are moved to a service provider;	IT cost reduction	3	ResearchGate	25.11.2019
Secure cloud computing: Benefits, risks and controls	Mariana Carroll, Alta Van der Merwe, Paula Kotzé	enables organisations to become more competitive due to flexible and agile computing platforms, providing for scalability and high-performance resources and highly reliable and available applications and data	scalability	3	ResearchGate	25.11.2019
Secure cloud computing: Benefits, risks and controls	Mariana Carroll, Alta Van der Merwe, Paula Kotzé	Cloud computing helps organisations to reduce power, cooling, storage and space usage and thereby facilitates more sustainable, environmentally responsible data centres. Moving to the cloud further frees up existing infrastructure and resources that can be allocated to more strategic tasks.	IT cost reduction	3	ResearchGate	25.11.2019
Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?	Karthik Kumar, Yung-Hsiang Lu	Two sample applications illustrate the benefits of offloading: a chess game and image retrieval; chess provides an example where offloading is beneficial for most wireless networks. Image processing: offloading saves energy only if B is very large—that is, at high bandwidths.	energy consumption	3	ResearchGate	25.11.2019
Cloud Computing Benefits and Architecture in E-Learning	Aida Ghazizadeh	x	x	x	IEEE	22.11.2019
Access control and user authentication concerns in cloud computing environments	Mohammad Javad Golkar, Mohammad Ahmadi, Mohammad Chizari, Mohammad Eslami, Mostafa Vali	x	x	x	IEEE	22.11.2019
SONA: A service oriented nodes architecture for developing Cloud Computing applications	Samah Ahmed Zaki Hassan	payment is done only when the service is used, "pay per use".	IT cost reduction	2	IEEE	22.11.2019
SONA: A service oriented nodes architecture for developing Cloud Computing applications	Samah Ahmed Zaki Hassan	there is no need of hardware upgrades, local hosting servers and other equipment's	IT cost reduction	2	IEEE	22.11.2019

Cloud Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
SONA: A service oriented nodes architecture for developing Cloud Computing applications	Samah Ahmed Zaki Hassan	to use the cloud, only the Internet connection is required by using any device having minimum software/hardware requirements	mobility	2	IEEE	22.11.2019
SONA: A service oriented nodes architecture for developing Cloud Computing applications	Samah Ahmed Zaki Hassan	resources of Cloud can be scaled easily, rapidly, dynamically, automatically, and provisioned "on demand" with all services, applications and processes, regardless of the location of user or device	scalability	2	IEEE	22.11.2019
Mobile cloud computing: advantage, disadvantage and open challenge	Ahmed Alzahrani, Nasser Alalwan, Mohamed Sarrab	Cloud computing allows dynamic scalability as demands fluctuate.	scalability	3	ACM DL	25.11.2019
Mobile cloud computing: advantage, disadvantage and open challenge	Ahmed Alzahrani, Nasser Alalwan, Mohamed Sarrab	pay as you go services	IT cost reduction	3	ACM DL	25.11.2019
Mobile cloud computing: advantage, disadvantage and open challenge	Ahmed Alzahrani, Nasser Alalwan, Mohamed Sarrab	cloud mobility supports access anywhere using a Web connection, services that are available wherever the end user might be located.	mobility	3	ACM DL	25.11.2019
Mobile cloud computing: advantage, disadvantage and open challenge	Ahmed Alzahrani, Nasser Alalwan, Mohamed Sarrab	cloud computing deployment is usually built on a very strong architecture thus providing resiliency	resiliency / availability	3	ACM DL	25.11.2019
Mobile cloud computing: advantage, disadvantage and open challenge	Ahmed Alzahrani, Nasser Alalwan, Mohamed Sarrab	cloud instances are deployed automatically on demand	scalability	3	ACM DL	25.11.2019
Mobile cloud computing: advantage, disadvantage and open challenge	Ahmed Alzahrani, Nasser Alalwan, Mohamed Sarrab	cloud is scaled to meet the required changes in IT system demands. In terms of performance, the cloud utilizes distributed system architectures that offer excellent computations speed	scalability	3	ACM DL	25.11.2019
A view of cloud computing	Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia	A first case is when demand for a service varies with time.	scalability	3	ACM DL	25.11.2019
A view of cloud computing	Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia	second case is when demand is unknown in advance	scalability	3	ACM DL	25.11.2019

Cloud Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
A view of cloud computing	Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia	organizations that perform batch analytics can use the “cost associativity” of cloud computing to finish computations faster (e.g. using 1,000 EC2 machines for one hour costs the same as using one machine for 1,000 hours.)	scalability	3	ACM DL	25.11.2019
A view of cloud computing	Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia	We start with elasticity. The key observation is that cloud computing’s ability to add or remove resources at a fine grain (one server at a time with EC2) and with a lead time of minutes rather than weeks allows matching resources to workload much more closely.	scalability	4	ACM DL	25.11.2019
A view of cloud computing	Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia	need not be concerned about overprovisioning for a service whose popularity does not meet their predictions, thus wasting costly resources, or underprovisioning for one that becomes wildly popular, thus missing potential customers and revenue.	IT cost reduction	1	ACM DL	25.11.2019
A view of cloud computing	Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia	cloud computing is often described as "converting capital expenses to operating expenses" (CapEx to OpEx) -> "pay as you go"	IT cost reduction	3+4	ACM DL	25.11.2019
Cloud Versus Edge Computing -- What's The Difference?	Peter Zornio	cloud processing power is nearly limitless because data centers full of computers	computational power	x	Google	25.11.2019
Cloud Versus Edge Computing -- What's The Difference?	Peter Zornio	cloud can be accessed from anywhere on multiple devices	mobility	x	Google	25.11.2019
Cloud- versus Edge-Computing Part 2	Wolfgang Wanner	Leistungen sind in der Regel anpass- und skalierbar	scalability	x	Google	25.11.2019
Cloud- versus Edge-Computing Part 2	Wolfgang Wanner	hohe Verfügbarkeit	resiliency / availability	x	Google	25.11.2019

Cloud Computing Limitations

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Fog computing: from architecture to edge computing and big data processing	Simar Preet Singh, Anand Nayyar, Rajesh Kumar, Anju Sharma	physical distance between the cloud and the user increases, transmission latency and response time increase with it	bandwidth / latency	2	ResearchGate	26.11.2019
Fog computing: from architecture to edge computing and big data processing	Simar Preet Singh, Anand Nayyar, Rajesh Kumar, Anju Sharma	migrating data to the cloud server and then receiving the response and vice versa consume much amount of time	bandwidth / latency	7	ResearchGate	26.11.2019
Opportunistic fog computing: Feasibility assessment and architectural proposal	Ricardo Silva, Jorge Sá Silva, Fernando Boavida	x	x	x	IEEE	25.11.2019
Cloud Computing Networking: Challenges and Opportunities for Innovations	Siamak Azodolmolky, Philipp Wieder, Ramin Yahyapour	applications require some guaranteed bandwidth between server instances	bandwidth / latency	6	ResearchGate	25.11.2019
Cloud Computing Networking: Challenges and Opportunities for Innovations	Siamak Azodolmolky, Philipp Wieder, Ramin Yahyapour	Insufficient bandwidth between these servers will impose significant latency on user interactions.	bandwidth / latency	6	ResearchGate	25.11.2019
Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations	Nitinder Mohan, Jussi Kangasharju	processing time of time-critical IoT applications can be limited by the network delay for offloading data to cloud	bandwidth / latency	1	ResearchGate	25.11.2019
Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations	Nitinder Mohan, Jussi Kangasharju	uploading data from a large number of IoT generators may induce network congestion	bandwidth / latency	1	ResearchGate	25.11.2019
The Cloud Computing: A Systematic Review	Mohammad Oqail Ahmad, Rafiqul Zaman KhanRafiqul Zaman Khan	some people panic that they might lose some of their rights or are unable to protect the rights of their users	legal/rights	10	ResearchGate	25.11.2019

Cloud Computing Limitations

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Software Engineering Challenges: A Cloud Based Architecture For Earthquake Forecasting	Aamir Khan, Nauman A. QureshiNauman A. Qureshi, Umair AbdullahUmair Abdullah	Network latency is the main problem for real time applications. Due to the distributed nature of the cloud, physical resources may spread over multiple locations, which can cause network latency and reduce the performance of the application.	bandwidth / latency	4	ResearchGate	25.11.2019
Software Engineering Challenges: A Cloud Based Architecture For Earthquake Forecasting	Aamir Khan, Nauman A. QureshiNauman A. Qureshi, Umair AbdullahUmair Abdullah	Due to the distributed nature of the cloud, physical resources may spread over multiple locations -> no geographic context	geographic context	4	ResearchGate	25.11.2019
Trust in Cloud Computing: Pro and Contra from the User's Point of View	Nils Backhaus, Manfred ThueringManfred Thuering	provider and the services may not be subject to the legislation of the user's country -> user cannot be certain which jurisdiction is valid.	legal/rights	8	ResearchGate	26.11.2019
SONA: A service oriented nodes architecture for developing Cloud Computing applications	Samah Ahmed Zaki Hassan	the Internet connection speed may affect the overall performances,	bandwidth / latency	2	IEEE	25.11.2019
A view of cloud computing	Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, Matei Zaharia	Applications continue to become more data-intensive and bandwidth is limited, e.g. Suppose we get 20Mbits/sec over a WAN link. It would take 10 * 1012 Bytes / (20×106 bits/second)= (8×1013)/(2×107) seconds = 4,000,000 seconds, which is more than 45 days.	bandwidth / latency	7	ACM DL	25.11.2019
Cloud- versus Edge-Computing Part 2	Wolfgang Wanner	Datenschutzdiskussionen, vor allem bei Diensten aus dem Ausland, zeigen, dass Fragen zu klären sind: „Wo liegen meine Daten?", „Wer hat darauf Zugriff?"	legal/rights	x	Google	25.11.2019
Cloud- versus Edge-Computing Part 2	Wolfgang Wanner	Übertragung im eigenen Firmennetz und anschließend per DSL-Leitung oder Mobilfunk, sind Szenarien zu entwickeln, die bei einem Ausfall einzelner Komponenten dieser Strecke zum Tragen kommen	bandwidth / latency	x	Google	25.11.2019

Edge Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions	Ahmet Cihat Baktir, Atay Ozgovde, Cem Ersoy	the overall latency can be decreased through high LAN bandwidth and decreased number of hops	bandwidth / latency	7	ResearchGate	26.11.2019
How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions	Ahmet Cihat Baktir, Atay Ozgovde, Cem Ersoy	Ha et al. analyzes energy consumption rates for applications such as face recognition and augmented reality. It is stated that offloading tasks to the edge servers results in lower energy consumption when it is compared to the cloud servers.	energy consumption	7	ResearchGate	26.11.2019
How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions	Ahmet Cihat Baktir, Atay Ozgovde, Cem Ersoy	prevents the consumption of the limited bandwidth of the core network by billions of devices at the edge	bandwidth / latency	7	ResearchGate	26.11.2019
How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions	Ahmet Cihat Baktir, Atay Ozgovde, Cem Ersoy	Edge Computing solves the problem of congestion within both the core network and datacenters, and the traffic can be regulated with less effort	bandwidth / latency	7	ResearchGate	26.11.2019
How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions	Ahmet Cihat Baktir, Atay Ozgovde, Cem Ersoy	distributing the services and applications and replicating them in the form of virtual machines (VMs) over edge servers create an opportunity of enhancing the scalability of the cloud	scalability	7	ResearchGate	26.11.2019
How Can Edge Computing Benefit from Software-Defined Networking: A Survey, Use Cases & Future Directions	Ahmet Cihat Baktir, Atay Ozgovde, Cem Ersoy	provide location-awareness	geographic context	11	ResearchGate	26.11.2019
Cloud-based IoT solution for State Estimation in Smart Grids: exploiting virtualization and edge-intelligence technologies	Alessio Meloni, Paolo Attilio Pegoraro, Luigi Atzori, Andrea Benigni, Sara Sulis	x	x	x	x	26.11.2019
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	data needs to be processed at the edge for shorter response time, more efficient processing and smaller network pressure.	bandwidth / latency	2	ResearchGate	26.11.2019
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	better fulfills privacy protection requirement	legal/rights	2	ResearchGate	26.11.2019
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	energy consumption could also be reduced by 30-40% by cloudlet offloading	energy consumption	3	ResearchGate	26.11.2019

Edge Computing Benefits

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	save the data transmission time and simplify the network structure	bandwidth / latency	4	ResearchGate	26.11.2019
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	data could be collected and processed based on geographic location	geographic context	4	ResearchGate	26.11.2019
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	short distance transmission, we can establish high bandwidth wireless access to send data to the edge	bandwidth / latency	8	ResearchGate	26.11.2019
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	the transmission reliability is also enhanced as the transmission path is short	bandwidth / latency	8	ResearchGate	26.11.2019
Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations	Nitinder Mohan, Jussi Kangasharju	Providing context: Resources in Edge-Fog cloud also provide contextual awareness to data generated by sensors. Edge resources play a role in combining data from sensors using location or application contexts.	geographic context	2	ResearchGate	25.11.2019
Edge-Fog Cloud: A Distributed Cloud for Internet of Things Computations	Nitinder Mohan, Jussi Kangasharju	The Edge-Fog cloud provides computation at the edge of the network near the IoT generators thus reducing the amount of data that flows in the network.	bandwidth / latency	2	ResearchGate	25.11.2019
Edge computing framework for distributed smart applications	Kaikai Liu, Abhishek Gurudutt, Tejeshwar Kamaal, Chinmayi Divakara, Praveen Prabhakaran	x	x	x	IEEE	26.11.2019
Cloud Versus Edge Computing -- What's The Difference?	Peter Zornio	enough or reliable network bandwidth	bandwidth / latency	x	Google	26.11.2019
Cloud Versus Edge Computing -- What's The Difference?	Peter Zornio	calculate results with a minimum amount of delay	bandwidth / latency	x	Google	26.11.2019
Cloud- versus Edge-Computing Part 1	Wolfgang Wanner	Werden geringe Latenzzeiten erwartet oder müssen Daten in Echtzeit zur Verfügung stehen, ist dieser Ansatz das Mittel der Wahl.	bandwidth / latency	x	Google	26.11.2019
Cloud- versus Edge-Computing Part 1	Wolfgang Wanner	Gebühren für die Kommunikation – vor allem bei international verteilten Applikationen – niedrig halten	IT cost reduction	x	Google	26.11.2019
IoT Edge – Von Gateway bis Machine Learning	Stefan Ried	Konnektivität & Gateway-Funktionen: Edge Devices besitzen verschiedene Drahtlose oder Wired Netzwerk Interfaces	bandwidth / latency	x	Google	26.11.2019
IoT Edge – Von Gateway bis Machine Learning	Stefan Ried	Sensoren ihre Daten meist nicht speichern können und in Real-Time "loswerden" müssen, können mittlere Edge-Devices die Echtzeitdaten sicher annehmen und als Queue in die Nicht-Echtzeitfähige Netzwerkwelt nach oben weitergeben. Dabei kommen nicht nur einfache Protokolle, wie MQTT, sondern auch industrienahe Referenz Architekturen wie OPC UA (https://de.wikipedia.org/wiki/OPC_Unied_Architecture) zum Einsatz.	bandwidth / latency	x	Google	26.11.2019

Edge Computing Limitations

Title	Authors	Extracted Data	Keywords	P.	Source	Extraction Date
Cloud- versus Edge-Computing Part 1	Wolfgang Wanner	Edge Computing wird gefährlich wenn eine skalierbare Lösung benötigt wird	scalability	x	Google	26.11.2019
Cloud- versus Edge-Computing Part 1	Wolfgang Wanner	Edge Computing ist nicht geeignet wenn der Rechen- und Speicherbedarf sehr unregelmäßig ist.	scalability	x	Google	26.11.2019
Cloud Versus Edge Computing -- What's The Difference?	Peter Zornio	x	x	x	Google	26.11.2019
IoT Edge – Von Gateway bis Machine Learning	Stefan Ried	Edge Computing ist bewusst selten hochverfügbar	resiliency / availability	x	Google	26.11.2019
IoT Edge – Von Gateway bis Machine Learning	Stefan Ried	Edge Computing ist selten mit großen Speicher- oder CPU-Leistungen ausgelegt, um Kosten und Stromverbrauch zu sparen	computational power	x	Google	26.11.2019
Edge Computing: Vision and Challenges	Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, Lanyu Xu	Reliability: Last but not least, reliability is also a key challenge at the Edge of the network: - From the service point of view: provide the action after node failure - From the system point of view: maintain the network topology of the whole system; services such as failure detection, thing replacement and data quality detection	resiliency / availability	2	ResearchGate	26.11.2019