UNIVERSITY OF NOVI SAD
**FACULTY OF TECHNICAL SCIENCES
NOVI SAD**

Miloš Simić

Micro clouds and edge computing as a service

- Ph. D. Thesis -

Supervisor
Goran Sladić, PhD, associate professor

Novi Sad, 2021.

Miloš Simić: *Micro clouds and edge computing as a service*

SERBIAN TITLE: Micro clouds and edge computing as a service

SUPERVISOR:
Goran Sladić, PhD, associate professor

LOCATION:
Novi Sad, Serbia

DATE:
September 2021

UNIVERZITET U NOVOM SADU ● **FAKULTET TEHNIČKIH NAUKA**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, **RBR**:	
Identifikacioni broj, **IBR**:	
Tip dokumentacije, **TD**:	Monografska dokumentacija
Tip zapisa, **TZ**:	Tekstualni štampani materijal
Vrsta rada, **VR**:	Doktorska disertacija
Autor, **AU**:	Miloš Simić
Mentor, **MH**:	dr Goran Sladić, vanredni profesor
Naslov rada, **NR**:	Micro clouds and edge computing as a service
Jezik publikacije, **JP**:	engleski
Jezik izvoda, **JI**:	srpski
Zemlja publikacije, **ZP**:	Srbija
Uže geografsko područije, **UGP**:	Vojvodina
Godina, **GO**:	2021
Izdavač, **IZ**:	Fakultet tehničkih nauka
Mesto i adresa, **MA**:	Trg Dositeja Obradovića 6, 21000 Novi Sad
Fizički opis rada, **FO**: (poglavlja/strana /citata/tabela/slika/grafika/priloga)	6/159/154/13/13/0/0
Naučna oblast, **NO**:	Elektrotehničko i računarsko inženjerstvo
Naučna disciplina, **ND**:	Distribuirani sistemi
Predmetna odrednica/Ključne reči, **PO**:	distributed systems, cloud computing, microservices, software as a service, edge computing, micro clouds
UDK	
Čuva se, **ČU**:	Biblioteka Fakulteta tehničkih nauka, Trg Dositeraj Obradovića 6, 21000 Novi Sad
Važna napomena, **VN**:	

UNIVERZITET U NOVOM SADU ● **FAKULTET TEHNIČKIH NAUKA**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Izvod, **IZ**:		U sklopu disertacije izvršeno je istraživanje u oblasti razvoja bezbednog softvera. Razvijene su dve metode koje zajedno omogućuju integraciju bezbednosne analize dizajna softvera u proces agilnog razvoja. Prvi metod predstavlja radni okvir za konstruisanje radionica čija svrha je obuka inženjera softvera kako da sprovode bezbednosnu analizu dizajna. Drugi metod je proces koji proširuje metod bezbednosne analize dizajna kako bi podržao bolju integraciju spram potreba organizacije. Prvi metod je evaluiran kroz kontrolisan eksperiment, dok je drugi metod evaluiran upotrebom komparativne analize i analize studija slučaja, gde je proces implementiran u kontekstu dve organizacije koje se bave razvojem softvera.	
Datum prihvatanja teme, **DP**:			
Datum odbrane, **DO**:			
Članovi komisije, **KO**:	Predsednik:	dr Branko Milosavljević, redovni profesor, FTN, Novi Sad	
	Član:	dr Silvia Gilezan, redovni profesor, FTN, Novi Sad	
	Član:	dr Gordana Milosavljević, vanredni profesor, FTN, Novi Sad	Potpis mentora
	Član:	dr Žarko Stanisavljević, docent, ETF, Beograd	
	Član, mentor:	dr Goran Sladić, vanredni profesor, FTN, Novi Sad	

UNIVERSITY OF NOVI SAD • **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monograph documentation
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Ph.D. thesis
Author, **AU**:	Miloš Simić
Mentor, **MN**:	Goran Sladić, Ph.D., Associate Professor
Title, **TI**:	Micro clouds and edge computing as a service
Language of text, **LT**:	English
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2021
Publisher, **PB**:	Faculty of Technical Sciences
Publication place, **PP**:	Trg Dositeja Obradovića 6, 21000 Novi Sad
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/	6/159/154/13/13/0/0
Scientific field, **SF**:	Electrical engineering and computing
Scientific discipline, **SD**:	Distributed systems
Subject/Key words, **S/KW**:	distributed systems, cloud computing, microservices, software as a service, edge computing, micro clouds
UC	
Holding data, **HD**:	Library of Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000
Note, **N**:	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Abstract, **AB**:		This thesis presents research in the field of secure software engineering. Two methods are developed that, when combined, facilitate the integration of software security design analysis into the agile development workflow. The first method is a training framework for creating workshops aimed at teaching software engineers on how to perform security design analysis. The second method is a process that expands on the security design analysis method to facilitate better integration with the needs of the organization. The first method is evaluated through a controlled experiment, while the second method is evaluated through comparative analysis and case study analysis, where the process is tailored and implemented for two different software vendors.	
Accepted by the Scientific Board on, **ASB**:		11.07.2019.	
Defended on, **DE**:			
Defended Board, **DB**:	President:	Branko Milosavljević, PhD, Full Professor, FTN, Novi Sad	
	Member:	Silvia Gilezan, PhD, Full Professor, FTN, Novi Sad	
	Member:	Gordana Milosavljević, PhD, Associate Professor, FTN, Novi Sad	Menthor's signature
	Member:	Žarko Stanisavljević, PhD, Assistant Professor, ETF, Belgrade	
	Member, Mentor:	Goran Sladić, PhD, Associate Professor, FTN, Novi Sad	

Obrazac **Q2.HA.06-05**- Izdanje 1

Acknowledgements

First of all, I would like to express my sincere gratitude to my mentors Professor Jovanka Pantović and Professor Hugo Torres Vieira, for their unselfish help and experienced guidance. Before starting my PhD studies I had no idea what doing research means. They introduced me to the wonderful world of formal methods and shown me how far one needs to go in order to extract the juice out of his work. Above all, I thank them for always encouraging me to "keep up the good work", and for demonstrating what it takes to be a true scientist and a good person.

To my elementary and high school teachers, and my professors at the university, thank you for making me prepared for doing the Ph.D.

I would like to thank my colleagues at the Chair of Mathematics at Faculty of Technical Sciences for letting me be part of the family and for sharing all the good and the bad that our job carries.

To all of my friends, for making my life happier.

To my family, I owe the most. I thank my "in-law" family Radovan, Jasna, and Maja, for their great support. To my sister Draginja, her husband Bojan and my nieces Jelisaveta and Jovana, to my older brother Mladen and his wife Nina, and to my younger brother Aleksandar, thank you for being a perfect family. To my parents Jovanka and Siniša, for their endless love. To my wife Sanja, and our daughters Tamara and Lenka, for their patience, support, and unquestionable love, thank you for being my world.

Abstract

Distributed software systems have changed the way people communicate, learn and run businesses: almost all aspects of human life have become connected to the internet. The system of interconnected computing devices has numerous positive impacts on everyday life, however, it also raises some concerns, among which are security, accessibility and availability issues.

This thesis investigates problems of formal, mathematically based, representation and analysis of controlled usage and sharing of resources in distributed software systems. The thesis is organized into four chapters. The first chapter provides motivation for our work, and the last concludes the thesis. The second and the third chapters are the core of the thesis, the former addresses controlling information passing and the latter addresses controlling usages of resources.

The second chapter presents a model for confidential name passing, called Confidential π-calculus, abbreviated C_π. This model is a simple fragment of the π-calculus that disables information forwarding directly at the syntax level. We provide an initial investigation of the model by presenting some of its properties, such as the non-forwarding property and the creation of closed domains for channels. We also present examples showing that C_π can be used to model restricted information passing, authentication, closed and open-ended groups. We present an encoding of the (sum-free) π-calculus in C_π and we prove the correctness of the encoding via an operational correspondence result.

The third chapter presents a model of floating authorizations. Our process model introduces floating authorizations as first-class entities, encompassing dimensions of accounting, domain, and delegation. We exploit the language of an already existing process algebra for authorizations, and we adopt a different semantic interpretation so as to capture accounting. We define the semantics of our model in two equivalent ways, using a labeled transition system and a reduction relation. We define error processes as undesired configurations that cannot evolve due to lacking authorizations. The thesis also provides a preliminary investigation of the behavioral semantics of our authorization model, showing some fundamental properties and also informing on the specific nature of floating authorizations.

In the third chapter, we also present a typing discipline that allows to statically single out processes that are not errors and that never evolve into errors, addressing configurations where authorization assignment is not statically prescribed in the system specification. We also develop a refinement of our typing discipline to pave the way for a more efficient type-checking procedure. We show an extended example of a scenario that involves the notion of Bring Your Own License, and we exploit this example to provide insight on a possible application of our model in programming language design.

Key words: distributed systms, cloud computing, micro clouds, edge computing

Rezime

Ključ reči: distributed systms, cloud computing, micro clouds, edge computing

Table of Contents

List of Figures

List of Tables

List of Abbreviations

CC	Cloud computing
AWS	Amazon Web Services
IoT	Internet of Things
DS	Distributed systems
DC	Distributed computing
DCs	Data centers
IaaS	infrastructure as a service
PaaS	Platform as a service
SaaS	Software as a service
CaaS	Container as a service
DBaaS	Databae as a service
XaaS	Everything as a Service
P2P	Peer-to-peer
DHT	Distributed Hash Table
NoSQL	Not Only SQL

EC	Edge computing
MEC	Mobile edge computing
MCC	Mobile cloud computing
QoE	Quality of experience
MDCs	Micro data-centers
SoC	Separation of concerns (SoC)
ES	Edge servers
CDN	Content delivery networks
SDN	Software-defined networks
VM	Virtual machine
OS	Operating system

Chapter 1

Introduction

Various software systems has changed the way people communicate, learn and run businesses, and interconnected computing devices has numerous positive applications in everyday life. Over the past decade, computation and data volumes have increased significantly [1]. Augmented reality, online gaming, face recognition, autonomous vehicles, or the Internet of Things (IoT) applications produce huge volumes of data. Workloads like those require latency below a few tens of milliseconds [1]. These requirements are outside what a centralized model like the CC can offer [1]. Even small problems can contribute to large downtime of applications and servies people are depending on. Recent example is yet another outage that happend in Amazon Web Services (AWS), and as a result a large amount of intrnet become unavaleble.

The aim of this thesis is to provide formal models uppon which we implement distributed system for oragnizing cloud-like geo-distributed environments for users or CC providers to utilize, in order to minimize downtime of critical services. The whole system can be looked as a pre-cloud or pre-processing layer sending only important data to the cloud minimizing cost for users, and ensuring avelability of CC services. Ensuring reliability and correctness of any system is very difficult, and should be mathematically based. Formal methods are techniques that allow us specification and verification of complex (software and hardware) systems based on mathematics and formal logic.

We start by describing the general problem area that our work addresses in Section 1.1. Sections 1.2 and 1.3 describe the theoretical background behind the problem, where we examine distributed systems (DS) and distributed computing (DC), focusing on design details, communication patters and organizational structure. In Section 1.4 we describe similar models that might be source of confusion. In Section 1.5 we describe different virtualization methods that are used in CC for system and/or applications. In section 1.6 we describe brifley concurency and parallelism and explain actor system. In Section 1.7, we specify the exact problem that our work addresses and describe our hypothesis and research goals in Section 1.8. Section 1.9 present the structure of the thesis.

1.1 Problem area

Cloud centralized architecture with enormous data-centers (DCs) capacities creates an effective economy of scale to lower administration cost [2]. However, when such a system grows to its limits, centralization brings more problems than solutions [3, 4]. Despite all the CC benefits, applications and services face a serious degradation over time, due to the high bandwidth and latency [5]. This can have a huge consequence on the business and potentially human lives as well. Organizations use cloud services to avoid huge investments [6], like creating and maintaining their own DCs. They consume resources created by others [7] and pay for usage time – a pay as you go model.

Data is required to be moved to the cloud from data sources, which introduces a high latency in the system [8]. For example, Boeing 787s generates half a terabyte of data per single flight, while a self-driving car generates two petabytes of data per single drive. Bandwidth is not large enough to support such requirements [9]. Data transfer is not the only problem: applications like self-driving cars, delivery drones, or power balancing in electric grids require real-time processing for proper decision making [9]. We might face serious issues if a cloud service becomes unavailable due to denial-of-service attack, network, or cloud failure [3].

To overcome cloud latency, research led to new computing areas, and model in which computing and storage utilities are in proximity to data sources [7]. The cloud is enhanced with new ideas for future generation applications [10].

1.2 Distributed systems

There are various definitions of DS, but we can think of DS as a systems where multiple entities can comunicate to one another in some way, but at the same time, they are able to performing some operations. Three significant characteristics of distributed systems are: (1) **concurrency of components**, (2) **lack of a global clock**, and (3) **independent failure of components** [11]. In [12] authors give formal definition "distributed system is a collection of autonomous computing elements that appears to its users as a single coherent system".

When talking about DS, we usually think about computing systems that are connected via network or over the internet. But DS are not exclusiv to domain of cumputer science. They existed before computers started to enrich almost every aspect of human life. DS have been used in varios different domains such as: **telecommunication networks**, **aircraft control systems**, **industrial control systems** etc. DS are used anywhere where amout of users are growing rapidly, so that single entity can't reponse to users demands in (near) real-time.

Distributed systems (in computer science) are consists of various algorithms, techniques and trade-offs to create an illusion that set of nodes act as one. DS algorithms may include: (1) replication, (2) consensus, (3) communication, (4) storage, etc.

This section will explain different aspects of DS in computing systems, that are important for future parts of the thesis. Section 1.2.1 gives more details about scalability and what it meens in modern day computer applications. Section 1.2.2 gives explanation what CC is, organizational aspects of CC as well as used models. Section 1.2.3 gives explanation what peer-to-peer networks are, and why are they important in modern DS. Section 1.2.4 gives general deffinition what mobile computing is and new ways of implementation DS.

1.2.1 Scalability

Scalability is the property of a system to handle a growing amount of work by adding resources to the system [13]. When talking about computer systems scalability can be represented in two flavors:

- **Scaling vertically** means upgradeing the hardware that computer systems are running on. Vertical scaling can increase performance to what latest hardware can offer, and here we are limited by the laws of phisics and Moor's law [14]. Tipical example that require this type of scaleing is relation dataase server. These capabilities are insufficient for moderate to big workloads.

- **Scaling horizontally** means that we scale our system by keep adding more and more computers, rather than upgrading the hardware of a single one. With this apporouch we are (almost) limitless how much we can scale. Whenever performance degrades we can simply add more computers (or nodes). These nodes are not required to be some high-end machines.

Table 1.1 summarize differences between horizontall and verticall scaling.

Scaling horizontally is a preferable way for scaling DS, not because we can scale easier, or because it is significantly cheaper than vertical scaling after a certain threshold [13] but because this approuch comes

	Scaling vertically	**Scaling horizontally**
Scaling	Limited	Unlimited
Managment	Easy	Comlex
Investments	Expensive	Afordable

Table 1.1: Differences between horizontall and verticall scaling.

with few more benefits that are esspecially important when talking large-scale DS. Adding more nodes gives us two important properties:

- **Fault tolerance** means that applications running on multiple places at the same time, are not bound to the fail of node, cluster or even DCs. As long as there is a copy of application running somewhere, user will get response back. This means that service is more **avalible**, that running on a single node no metter how high-end that node is. Eventually all nodes are going to break.

- **Low latency** refers to the idea that the world is limited by the speed of light. If a node running application is too far away, user will wait too long for the response to get back. If same application is running on multiple places, user request will hit node that is closest to the user.

But despite all the obvious benefits, for a DS to work properly, we need the write software in such a way that is able to run on multiple nodes, as well as that accept **failure** and deal with it. This turns out to be not an easy task.

For example users need to be aware when using DS, is related to distributed data storage systems. Storage implementations that relys on vertical scaling to ensure scalability and faulte tolerance, have one nasty feature.

This nasty feature is represented in theorem called **CAP theorem** presented by Eric Brewer [15]. Proven after insepction [16], CAP theorem states that it is impossible for a distributed data store to simultaneously provide more than two out of three guarantees:

1. **C**onsistency
2. **A**vailability
3. **P**artition tolerance

Years after CAP theorem inception, Shapiro et al. prove that we can alleviate CAP theorem problems by only in some cases, and offers **Strong Eventual Consistency (SEC) model** [17]. They prove that if we can represent our data structure to be:

- **Commutative** $a * b = b * a$
- **Associative** $(a * b) * c = a * (b * c)$
- **Idempotent** $(a * a) = a$

where $*$ is a binary operation, for example: *max*, *union*, *or* we can rely on SEC properties,

1.2.2 Cloud computing

We can define cloud computing (CC) like aggregation of computing resources as a utility, and software as a service [18]. Hardware and software in big DCs provide services for user consumption over the internet [19]. Resources like CPU, GPU, storage, and network are utilities and can be used as well as released on-demand [20]. The key strength of the CC are offered services [18].

The traditional CC model provides enormous computing and storage resources elastically, to support the various applications needs. This property refers to the cloud ability to allow services, allocation of additional resources, or release unused ones to match the application workloads on-demand [21]. Services usually fall in one of three main categories:

- **Infrastructure as a service (IaaS)** allows businesses to purchase resources on-demand and as-needed instead of bying and manageing hardware themself.

- **Platform as a service (PaaS)** delivers a framework for developers to create, maintain and manage their applications. All resources are managed by the enterprise or a third-party vendor.
- **Software as a service (SaaS)** deliver applications over the internet to its users. These applications are managed by a third-party vendor, .

Figure 1.1 show difference in control and management of resources between different cloud options and on-premises solution.

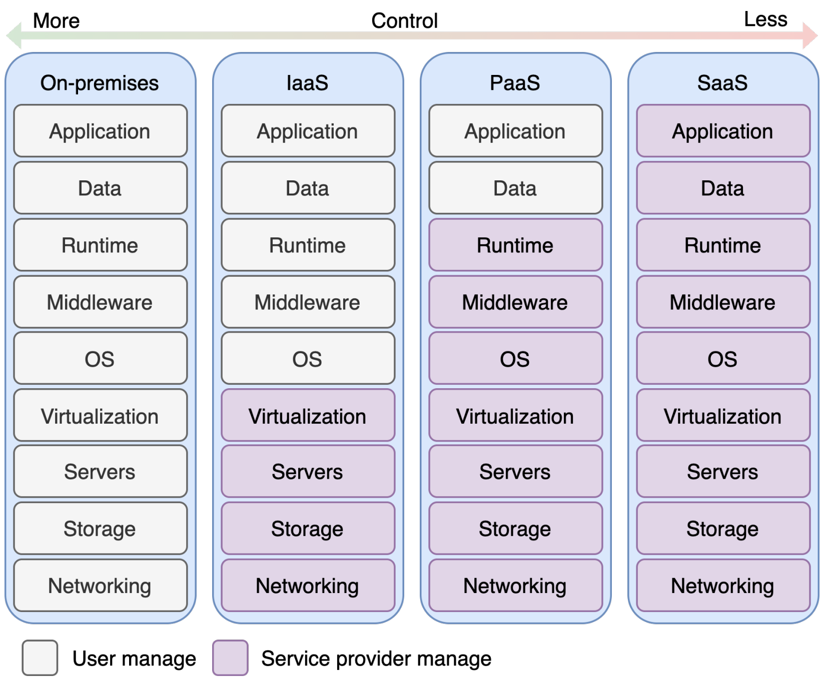

Figure 1.1: Difference between cloud options and on-premises solution.

The user can choose a single solution, or combine more of them if such a thing is required depending on preferences and needs.

CC has been the dominating tool in the past decade in various applications [7]. It is changing, evolving, and offering new types of services. Resources such as container as a service (CaaS), database as a service (DBaaS) [22] are newly introduced. The CC model gives us a few benefits. Centralization relies on the economy of scale to lower the cost of administration of big DCs. Organizations using cloud services avoid huge investments. Like creating and maintaining their own DCs. They consume resources created by others [7] and pay for usage time – a pay as you go model.

But centralization give us few really hard problems to solve. As already stated in section 1.1 data is required to be moved to the cloud from data sources, which introduces a high latency in the system [8].

There are few notable attempts to help data ingestion into the cloud. Remote Direct Memory Access (RDMA) protocol makes it possible to read data directly from the memory of one computer and write that data directly to the memory of another. This is done by using *specialized hardware* interface cards and switches and software as well, and operations like read, write, send, receive etc. do not go through CPU. With this caracteristivs, RDMA have low latencies and overhead, and as such reach better throughputs [23]. This new hardware may not be cheap, and not evey CC provider use them for every use-case. And this may not be enough, esspecially with ever growing amount of IoT devices and services.

Over the years there are more as a service options avalible, forming **everything as a service (XaaS)** model [24]. This model propose that any hardware or software resource can be ofered as a service to the users over the internet.

Table 1.2 shows common examples of SaaS, PaaS, and IaaS applications.

Platform type	**Common Examples**
IaaS	AWS, Microsoft Azure, Google Compute Engine
PaaS	AWS Elastic Beanstalk, Azure, App Engine
SaaS	Gmail, Dropbox, Salesforce, GoToMeeting

Table 1.2: common examples of SaaS, PaaS, and IaaS.

CC is giving a user an illusion that he is using single machine, while the backgroud implementaion is fairly complicated and consists of various elements that are composed of countles machines. CC is tipical example of horizontally scalable system presented in 1.2.1

1.2.3 Peer-to-peer networks

Peer-to-peer (P2P) communication is a networking architecture model that partitions tasks or workloads between peers [25]. All peers are created equally in the system, and there is no such thing as a node that is more important then others. Every Peer have a portion system resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts [25]. P2P nodes are connected and share resources without going through a separate server computer that is responsabile for routing. Figure 1.2 show difference in network topology between P2P networks and client-server architecture.

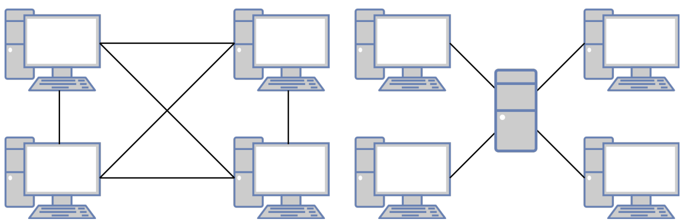

Figure 1.2: P2P network and client-server network.

Peers are creating a sense of virtual community. This community of peers can resolve a greater tasks, beyond those that individual peers can do. Yet, these tasks are beneficial to all the peers in the system [26].

Based on how the nodes are linked to each other within the overlay network, and how resources are indexed and located, we can classify networks as [27]:

- **Unstructured** do not have a particular structure by design, but they are formed by nodes that randomly form connections [28]. Their strenght and weaknes at the same time is ther lack of structure. These networks and robust when peers join and leave network. But when doing query, they must found more possible peers that have same peace of data. Tipical example of this group is a Gossip-based protocols like [29].
- **Structured** peers are organized into a specific topology, and the protocol ensures that any node can efficiently search the network for a resource. The famos type of structured P2P network is a Distributed Hash Table (DHT). These networks maintain lists of neighbors to do more efficent lookup, and as such they are not so robust when nodes join or leave the network. DHT commonly used in resource loopkup systems [30], and as efficent resource lookip management and scheduling of applications, or as an integral part of distributed storage systems and NoSQL[31] databases.
- **Hybrid** combine previous two models in various ways.

P2P networks are great tool in many arsenals, but because their unique ability to act as a server and as a client at the same time we must be aware and pay more attention to security because they are more vulnerable to exploits [32]/

1.2.4 Mobile computing

Mobile cloud computing (MCC), was the first idea that introduced task offloading [33, 34]. Heavy computation remains in the cloud. Mobile devices run small client software and interact with the cloud, over the internet using his resources.

The main problem with MCC is that the cloud is usually far away from end devices. That leads to high latency and bad quality of experience (QoE) [34]. Especially for latency-sensitive applications. Even though MCC is not that much different from the standard cloud model. We had moved a small number of tasks from the cloud. Thus opening the door for future models.

To overcome cloud latency and MCC problems, research led to new computing areas like edge computing (EC). EC is a model in which computing and storage utilities are in proximity to data sources [7]. The cloud is enhanced with new ideas for future generation applications [10].

Over the years, designs like fog [35], cloudlets [6], and mobile edge computing (MEC) [36] emerged. In this thesis, we refer to all these models as edge nodes. They all use the concept of data and computation offloading from the cloud closer to the ground [37], while heavy computation remains in the cloud because of resource availability [10].

EC models introduce small-scale servers that operate between data sources and the cloud. Typically, they have much less capabilities compared to the cloud counterparts [38]. These servers can be spread in base stations [36], coffee shops, or over geographic regions to avoid latency as well as huge bandwidth [6]. They can serve as firewalls [39] and pre-processing tier, while users get a unique ability to dynamically and selectively control the information sent to the cloud.

1.3 Distributed computing

DC can be defined as the use of a DS to solve one large problem by breaking it down into several smaller parts, where each part is computed in the individual node of the DS and coordinatio is done by passing messages to one another [11]. Computer programs that use this strategy and runs on DS are called **distributed programs** [40, 41].

Similar to CC in Section 1.2.2, to a normal user, DC systems appear as a single system similar to one he use every day on his personal computer.

In this section we are going to explain examples of distributed programs that rely on DS that are important for future parts of the thesis. Section 1.3.1 gives more details about using DS to process huge quantity of data. Section 1.3.2 gives explanation how to build scalable applications that are able to whitstand huge request rate and large amout of users.

1.3.1 Big Data

Tearm big data means that the data is unable to be handled, processed or loaded into a single machine [42]. That menas that traditional data mining methods or data analytics tools developed for a centralized processing may not be able to be applied directly to big data [43].

New tools and methos that are developed are relying on DS and one specific feature **data locality**. Data locality can be described as a process of moving the computation closer to the data, instead of moving large data to computation [44]. This simple idea, minimizes network congestion and increases the overall throughput of the system.

In 1.1 we already give two examples how huge generated data could be, and when we incude other IoT sensors and devices these numbers will just keep getting bigger [45].

On contrary to relational databases that mostly deal with structured data, big data is dealing with various kinds of data [42, 43, 44]:

- **Structured** data is kind of data that have some fixed structure and format. Tipical example of this is data stored inside table of some database. organizations susully have no huge problem extracting some kind of value out of the data.

- **Unstructured** data is kind of data where wo do not have any kind of structure at all. These data sources are heterogeneous and may containing a combination of simple text files, images, videos etc. This type of data is usully in raw format, and organizations have hard time to derive value out.

- **Semi-structured** data is kind of data that can contain both previously mentioned types of data. Example of this type of data is XML files.

Along it's share size, big data have other instantly recognizable features called **V's** of big data [46]. Name is derived from starting letters from the other features that are describing big data. Image 1.3 show 6 V's comonly used to represent the big data.

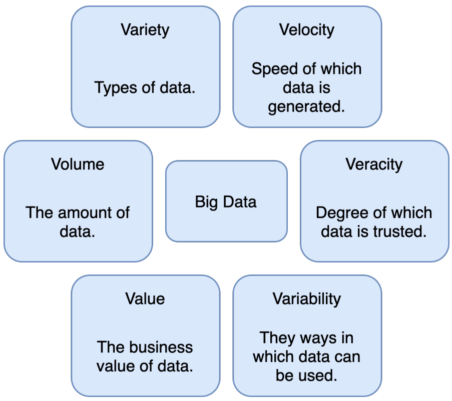

Figure 1.3: V's of Big Data.

Processing in big data systems can be represented as [47, 48]:

- **Batch processing** represents data prcessing technique that is done on huge quantity of the stored data. This type of prcessing is usually slow and requure time.

- **Stream processing** represents data processing technique that is done as data get into the system. This type of processing is usually done on smaller quantity of the data **at the time**, and it is faster.

- **Lambda architectures** represents processing technique where stream processing and handling of massive data volumes in batch are combined in a uniform manner, reducing costs in the process [48].

Big data systems, are not processing and value extracting systems. Big data systems can be separated in few categories: (1) data storage, (2), data ingestion (3), data processing and analytics. All these system aids to properly analyze ever growing requirements [49],

Dispite promise that big data offers to derive value out of the collected data, this task is not easy to do and requre properly set up system filtering and removeing data that contains no value. To aid this idea, data could be filtered and little bit preprocessed on close to the source [50], and as such sent to data lakses [51].

1.3.2 Microservices

There is no single comperhensive deffinition of what a microservice is. Differnet people and organizations use different definition do describe them. A working definition is offered in [52] as "s microservice is a cohesive, independent process interacting via messages". Despite lack of comperhensive definition all agree on few features that come with microservices:

1. they are small computer programs that are independently deployable and developed.
2. they coud be developed using different languages, principles and using differend databses.
3. they communicate over the network to achieve some goal.
4. they are organized around business capabilities [53].
5. they are implemented and mainteined by a small team.

Industry is migrating much of their applications to the cloud, because CC offers to scale their computing resources as per their usage [54]. Microservices are small loosely coupled services that follos

UNIX philosophy "do one thing, and do it well" [55], and they communicate over well defined API [52].

This architecure patter is well aligned to the CC paradigm [54], contrary to previous models like monolith whose modules cannot be executed independently [52, 56], and are not well aligned with the CC paradigm [56]. Table 1.3 summrize differences between monolith and microservie architecture.

	Monolith	**Microservices**
Structure	Single unit	Independent services
Management	Usually easier	Add DS complexity
Scale/Update	Entire app	Per service
Error	Usually crush entire app	App continue to work

Table 1.3: Differences between horizontall and verticall scaling.

Since their inception, microservices architecture is gone through some adaptations. And modern day microservices are extended with two new models each with it's unique abilities and problems:

- **Cloud-native applications**, are specially designed applications for CC. They are distributed, elastic and horizontal scalable system by their nature, and composed of (micro)services which isolates state in a minimum of stateful components [57].

- **Serversles applications** is a new computing model where the developers need to worry only about the logic for processing client requests [58]. Logic is represented as event handler that only runs when client request is received, and billing is done only when these functions are executing [58].

- **Service Mesh** is designed to standardize the runtime operations of applications [59]. As part of the microservices ecosystem, this dedicated communication layer can provide a number of benefits, such as: (1) observability, (2) providing secure connections, or (3) automating retries and backoff for failed requests.

Microservices communicate over a network to fulfil some goal using message passing technique and technology-agnostic protocols such as HTTP. They can be implemented as REST srevices, RPC calls or event-driven services. They are well aligned with text based protocols like HTTP/1 using *JSON* for example, or binary protocols such as HTTP/2 using *protobuf* and *gRPC* for example, and even new faster version like HTTP/3.

It is important to point out, that all flavors of microservices applications rely on continuous delivery and deployment [60]. This is enabled by lightweight containers, instead of virtual machines [61], and orchestration tools such Kubernetes [62]. These concepts wll be described in more detail in Section 1.5.

Microservices architecture are good starting point especially for build as a service applications, and applications that should serve huge amount of requests and users. Esspecially with benefits of CC to pay for usage, and ability to scale parts of the system independently. Although they are not necessarily easy to implement properly. There are more and more critique to the architecture model [63]. Microservices are relying and use parts of the DS, and as such they inherit almost all problems DS has. Best chance to success is to follow existing patters and use existing solutions with proven quality.

1.4 Similar computing models

In this section we are going to shortly describe models that are similar to the DS, and as such they may be the source of confusion. Section 1.4.1 describes parallel computing compared to DC. Section 1.4.2 describe decentralized computing compared to DS.

1.4.1 Parallel computing

DC and parallel computing seems like models that overlap, and that may share some features. Broadly speaking, this might be true [40].

But distinguished between the two can be presented as follows: in parallel computing all processor units have acces to the shared memory, while in DC all processors have their own memory. Figure 1.4 visually summarize the differences between DC (*up*) and parallel computing (*down*).

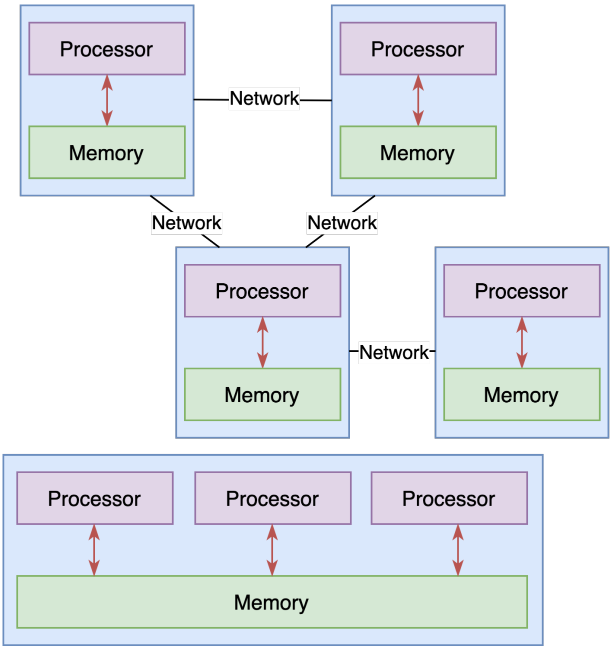

Figure 1.4: Difference between DC and parallel computing.

Parallel computing is ofet used strategy with problems, that due to their nature or constraints must be done on multi-core machines simultaneously [64].

1.4.2 Decentralized systems

Decentralized systems are similar to DS, in technical sense they are still DS. But if we take closer look, these systems **should not** be owned by single the entity. CC for example is example of DS, but centralized by the owner like AWS or Google because all computation needs to be moved to big DCs, which introduces a high latency in the system [8].

By today standards, if people are talking about decentralyzed systems, they usually mean on blockchain technology [65]. But even if this technology is run in the cloud, it is loosing the decentralized feature. This is the caveat people needs to be aware. These systems are facing different issues, because any participent in the sustem might be malicious and they need to handle this case. Nontheless, CC can and should be decentralized in a sense that some computation can happend outside CC DCs, closer to the sources of data.

1.5 Virtualization techniques

Virtualization as a technique started long ago in time-sharing systems, to provide isolation between multiple users shareing a single system liek a mainframe computer [66].

In [67] Sharma et al. describe virtualization as technologies which provide a layer of abstraction of the physical computing resources between computer hardware systems and the software systems running on them.

Modeern virtualization diferentiate few different tools. Some of them are used as an integral part of the infrastructure for some flavors like IaaS, while others are used in different CC flavors as well as microservices packageing and distribution format, or are new and still are looking for their place. These options are:

- **Virtual machines (VM)** are the oldest tehnology of the three. In [67] Sharma et al. describe them as a self-contained operating environment consisting of guest operating system and associated applications, but independent of host operating system. VMs enable us to pack isolation and better utilization of hardware in big DCs. They are vidly used in IaaS environment [68, 69] as a base where users can install theirown operating system (OS) and required software tools and applications.

- **Containers** provide almost same functionality to VMs, but there are several subtle differences that make them a goto tool in modern develpment. Instead of the guest OS running on top of host OS, containers use tools that are in Linux kernles like *cgroups* and *namespaces* to provide isolation. Containers reduce time and footprint from development to testing to production, and they utilize even more hardware resources compared to VMs and show better performance compared to the VMs [70, 61]. Containers provide easier way to pack servies and deploy and they are esspecially used in microservices architecture and service orchestration tools like Kubernetes [62]. Google stated few times in their on-line talks that hey have used container technology for all their services, even they run VMs inside containers for their cloud platform. Even though they exist for a while, containers get popularized when companies like Docker and CoreOS developed user-friendly APIs.

- **Unikernels** are the newest addition to the virtualization space. In [71] Pavlicek define unikernels as small, fast, secure virtual machines that lack operating systems. Unikernels are comprised of source code, along with only the required system calls and drivers. Because of their specific design, they have single process and they contains and executes what it absolutely needs to nothing more and nothing less [72]. They are advertised that new technology that will safe resources and that they are *green* [73] meaning they save both power and money. When put to the test and compared to containers they give interesting results [72, 74]. Unikernels are still new technology and they are not widly adopted yet. But they give promessing features for the future, esspecially **if** properly ported to ARM architectures, and various development languages. Unikernes will probably be used as a user applications and functions virtualization tool, because their specific architecture, esspecailly for serverless applications presened in 1.3.2.

Figure 1.5 represent architectural differences between VMs, containers and unikernels.

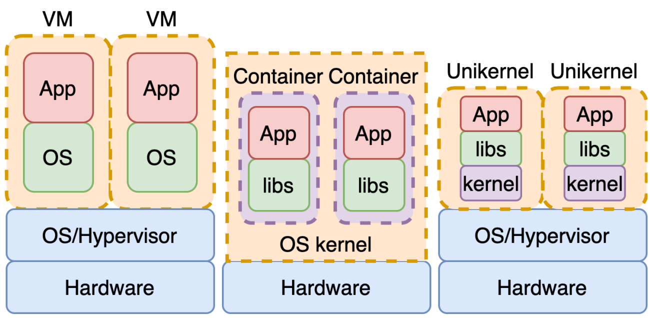

Figure 1.5: Architectural differences between VMs, containers and unikernels.

1.6 Concurency and parallelism

People usually confuse these two concepts. Even they looks similar, they are different way of doing things. In his talk Rob Pike [75] give great explanation and examples on this topic. In this toke he give great deffinitions of these concepts like:
Concurrency is composition of independently executing things. Concurrency is about dealing with a lot of things at once.
Parallelism is simultaneous execution of multiple things. Parallelism is about doing a lot of things at once.

These things are important, esspecially when building applications and systems that should achieve very high throughput. We must build them with good structure and good concurrency can provide this enableing possible parallelism, but with communication [75, 76].

1.6.1 Actor model

Similar model to the [76] is actor model [77]. In actor model, the main idea is that actors are small concurrent code that communicate independently by sending messages, removing the need for lock-based synchronization.

Actors do not share memory, and actor can create another actor/s and even watch on them in case they stop unexpectedly. And when an actor finished its job, and he is not needed anymore, it disappears.

1.7 Motivation and Problem Statement

In [78] Greenberg et al. point out that micro data-centers (MDCs) are used primarily as nodes in content distribution networks and other "embarrassingly distributed" applications.

One size never fits all, so the cloud should not be our final computing shift. Various models presented in 1.2.4, show possibility that computing could be done closer to the data source, to lower the latency for its clients by contacting the cloud only when needed, while heavy computation remains in the cloud because of resource availability. Send to the cloud only information that is crucial for other services or applications [50]. Not ingest everything as the standard cloud model proposes.

MDCs with a zone-based server organization is a good starting point for building EC as a service, but we need a more available and resilient system with less latency. EC originates from P2P systems [4] as sugested by López et al., but expands it into new directions and blends it with the CC. But, infrastructure deployment will not happen until the process is trivial [79]. Going to every node is tedious and time consuming. Especially when geo-distribution is taken into consideration.

A well defined system could be offered as a service, like any other resource in the CC. We can offer it to researchers and developers to create new human-centered applications. If we need more resources on one side, we can take from one pool of resources and move to another one. But on the other hand, some CC providers might choose to embed it into their own existing system, hiding unnecessary complexity, behind some communication interface or proposed application model.

The idea of small servers with heterogeneous compute, storage, and network resources, raise interesting research idea and motivation for this thesis. Taking advantage of resources organized locally as micro clouds, community clouds, or edge clouds [80] suggested by Ryden et al., to help power-hungry servers reduce traffic [81]. Contact the cloud only when needed [50]. Send to the cloud only information that is crucial for other services or applications. Not ingest everything as the standard CC model proposes.

To achieve such behavior, dynamic resource management, and device management is essential. We must perceive available resources, configuration, and utilization [82, 36].

Traditional DCs is a well organized and connected system. On the other hand, these MDCs consist of various devices, including ones presented in 1.2.4 that are not [83]. This idea, brings us to the problem this thesis address.

EC and MDCs models lack dynamic geo-organization, well defined native applications model, and clear separation of concerns. As such they cannot be offered as a service to the users. They usually exist independently from one another, scattered without communication between them, offered by providers who mostly lock users in their own ecosystem. Co-located edge nodes should be organized locally, making the whole system and applications more available and reliable, but also extending resources beyond the single node or group of nodes, maintaining good performance to build servers and clusters [84].

This cloud extension deepens and strengthens our understanding of the CC as a whole. With the separation of concerns setup, EC native applications model, and a unified node organization, we are moving towards the idea of EC as a service.

Based on this, we define the problem through the following research questions two segments:

1. *Can we organize geo-distributed edge nodes in a similar way to the cloud, adopted for the different environment, with clear separation of concerns and familiar applications model for users.*

2. *Can we offer these organized nodes as a service to the developers and researchers for new human-centered applications, based on the cloud pay as you go model?*

3. *Can we make model in such a way that is formaly correct, easy to extend, understand and reason about?*

This cloud-like extension makes the whole system and applications more available and reliable, but also extends resources beyond the single node. Satyanarayanan et al. in [39] show that MDCs can serve as firewalls, while Simić et al, in [50] use similar idea as preprocessing tier. At the same time, users are getting a unique ability to dynamically and selectively control the information sent to the cloud. Years after its inception, EC is no longer just an idea [39] but a must-have tool for novel applications to come.

1.8 Research Hypotheses, and Goals

Based on reserach questions and motivation presented in 1.7, we derive the hypothesis around which the thesis is based. It can be summarized as follows:

- Organize EC nodes in a standard way based on cloud architecture, with adaptation for an EC geo-distributed environment. Give users the ability to organize nodes in the best possible way in some geographic areas to serve only the local population in near proximity.

- Offer it to researchers and developers to create new human-centered applications. If we need more resources on one side, we can take from one pool of resources and move to another one, or organize them any other way needed.

- Present clear separation of concerns for the future EC as a service model, and establish a well-organized system where every part has an intuitive role.

- Present unified model that supports heterogeneous EC nodes, with a set of technical requirements that nodes must fulfil, if they want to join the system.

- Present a clear application model so that users can use full potential of newly created infrastructure.

From this hypothesis, we ce derive the primary goals of this thesis, where the expected results include:

1. *The construction of a model with a clear separation of concerns for the model influenced by cloud organization, with adaptations for a different environment. With a model for EC applications utilizing these adaptations. This addresses the first research question, and is the topic of Chapter 3.*

2. *The constructed model is more available, resilient with less latency, and as such it can be offered to the general public as a service like any other service in the cloud. This addresses the second research question, and is the topic of Chapter 3.*

3. *The constructed is is well described formaly, using solid mathematical theory, but also easy to extend both formaly and technicaly, easy to understand and reason about. This addresses the third research question, and is the topic of Chapter 3.*

1.9 Structure of the thesis

Throughout this introductory Chapter, we defined the motication for out work with problems that this thesis addresses and presented the necessary background informations and areas to support our work. Here we outline the rest of the thesis.

Chapter 2 presents the literature review, where we examine different aspects of existing systems and methods important for the thesis. We analyze existing nodes organizational abilities in both industry and academia frameworks and solutions to address our first research question. We further exemine platform models from industry and academia tools and frameworks to address our second research question. And last but not least, we examine current strategies to offload tasks from the cloud. All three parts address our third research question.

Chapter 3 details our model, how it is related to other research and where it connects to other existing models and solutions. We further describe our solution as well as protocols requried for such sysrtem to be implemented foramly. We give examples of how exisintg infrastructure could be used, as well as familiar application model for developers. At the end we present our implementation details with limitations.

Chapter 4 concludes our work and presents opportunities for further research and development.

Chapter 2

Research review

Faced real issues and limits of cloud computing, both academia, and the industry started researching and developing viable solutions. Some research is focused more on adapting existing solutions to fit EC, while others experiment with new ideas and solutions.

In this Chapter, we present the results of our research reviews existing nodes organizational abilities, as well platform models from industry and academia and cloud offloading techniques, addressing issues discussed earlier.

2.1 Nodes organization

A zone-based organization for edge servers (ES) presented by Guo et al. in [85], gives an interesting perspective on EC in a smart vehicles application. They showed how zone-based models enable continuity of dynamic services, and reduce the connection handovers. Also, they show how to enlarge the coverage of ESs to a bigger zone, thus expanding the computing power and storage capacity of ESs. Since one of the premises of EC is geo-distributed workloads, organizing ESs into zones and regions could potentially benefit EC.

Baktir et al. [86] explored the programming capabilities of software-defined networks (SDN). Findings show SDN can simplify the management of the network in cloud-like environment. SDN is a good candidate for networking because it hides the complexity of the heterogeneous environment from the end-users. Kurniawan et al. [87] argue about very bad scalability in centralized delivery models like cloud content delivery networks (CDN). They proposed a decentralized solution using nano DCs as a network of gateways for internet services at home [87]. These DCs are equipped with some storage as well. Authors show a possible usage of nano DCs for some large scale applications with much less energy consumption.

Ciobanu et al. [88] introduce an interesting idea called drop computing. The authors show that we can compose EC platforms ad-hoc, thus enabling collaborative computing dynamically, using a decentralized model over multilayered social crowd networks. Instead of sending requests to the cloud, drop computing employees the mobile crowd formed of nearby devices, hence enabling quick and efficient access to resources. The authors show an interesting idea of how to form a computing group ad-hoc. Forming ad-hoc platforms from crowd resources might raise a few possible concerns: (1) crowd nodes availability, and (2) offered resources. Crowd nodes might be an interesting idea as a backup option, in cases we need more computing power or storage and there are no more available resources to use.

MDCs are an interesting model and area of rapid innovation and development. Greenberg et al. [78] introduce MDCs as DCs that operate in proximity to a big population (on contrary to nano DCs that serves a lot smaller population), thus minimizing the latency and costs for end-users [89, 78], and reducing the fixed costs of traditional DCs. Minimum size of an MDCs is defined by the needs of the local population [78, 90], with agility as a key feature. Here agility means an ability to dynamically grow and shrink resources and satisfy the demands and usage of resources from the most optimal location [78].

2.2 Platform models

Kubernetes [62] is an open-source variant of Google orchestrator Borg [91]. All workloads end in the domain of one cluster [62, 91, 92]. Rossi et al. [92] focuses on adapting Kubernetes for geo-distributed workloads using a reinforcement learning (RL) solution, to learn a suitable scaling policy from experience. Like every other machine learning implementation this could be potentially slow due to the required model training. Kubernetes is a promising solution, but it might not be the best proposal for EC. By design, Kubernetes operate in a completely different environment from EC. The second potential issue is the deployment concept that might be too complicated for EC workloads. On the other hand, there are a few ideas that are worth exploring, such as loosely coupling elements with labels and selectors. Nonetheless, researches show that adapted Kubernetes architecture works for geo-distributed workloads like EC.

Ryden et al. [80] present a platform for distributed computing with attention to user-based applications. Unlike other systems, the goal is not to implement a resource management policy, but to give users more flexibility for application development. Users implement applications using Javascript (JS) programming language, with some embedded native code for efficiency. Similar to [88], the authors use volunteer nodes to run all the workloads, with the difference that some nodes are used for storage, while others are used for calculation. Sandboxing technique protects nodes running applications from malicious code. Users develop their applications using JS only. This restriction comes from using Google Chrome Web browser-based Native Client (NaCl) sandbox [93]. JS is a popular language at the moment, but the restriction of a single language might be a deal-breaker for some usages. If standard virtual machines become too resource-demanding, a solution using containers could provide sandboxing and bring better resource utilization. It is an interesting idea to show how users can develop their applications and run them in an EC environment.

Lèbre et al. [94] describe a promising solution of extending OpenStack, an open-source IaaS platform for fog/edge use cases. They try to manage both cloud and edge resources using a NoSQL database. Implementation of a massively distributed multi-site IaaS, using OpenStack is a challenging task [94]. Communication between nodes of different sites can be subject to important network latencies [94]. The major advantage is that users of the IaaS solution can continue using the same familiar infrastructure. In [95] Shao et al. present a possible MDCs structure serving only the local population, in the smart city use-case.

In [10], the Ning et al. show current open issues of EC platforms based on the literature survey. They illustrate the usage of edge computing platforms to build specific applications. In the survey, the authors outline how CC needs EC nodes to pre-process data, while EC needs massive storage and strong computing capacity of CC.

There are few industry operating frameworks for EC, like Amazon Greengrass [96], deeply connected to the entire Amazon cloud ecosystem, or KubeEdge [97], a lightweight extension of Kubernetes framework. These frameworks are mainly used for user-based applications, while, for instance, General Electric Predix [98] is a scalable platform used for industrial IoT applications.

2.3 Task offloading

As we already mention in 1.2.4 EC nodes rely on the concept of data and computation offloading from the cloud closer to the ground [37], while heavy computation remains in the cloud because of resource availability [10].

In literature, there are few platforms proposing task offloading [56-59] to the nearby edge layer. These offloading techniques are based on different parameters, options, and techniques, to put tasks to different sets of nodes.

2.4 Applications

2.5 Thesis position

Different from the aforementioned work,this thesis focuses on descriptive dynamic organization of geo-distributed nodes over an arbitrary vast area that lacks in other solutions. To achieve such a task, thesis model is influenced by the CC organization, but adapted for a different environment such as EC. These adaptations are followed by clear Separation of concerns (SoC) and EC applications model. All these allow us to push the whole solution more towards EC as a service model like any other utility in the cloud.

Chapter 3

Micro clouds and edge computing as a service

3.1 Formal model

3.1.1 Multiparty asynchronous session types

3.1.2 Health-check protocol

3.1.3 Cluster formation protocol

3.1.4 List detail protocol

3.2 Configurable Model Structure

3.3 Applications Model

3.3.1 Packaging

3.3.2 Execution

3.4 Separation of concers

3.5 As a service model

3.6 Results

3.7 Limitations

Chapter 4

Conclusion

4.1 Summary of contributions

4.2 Future work

Bibliography

[1] M. Chiang, T. Zhang, Fog and iot: An overview of research opportunities, IEEE Internet Things J. 3 (6) (2016) 854–864. doi:10.1109/JIOT.2016.2584538.
URL https://doi.org/10.1109/JIOT.2016.2584538

[2] M. F. Bari, R. Boutaba, R. P. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, M. F. Zhani, Data center network virtualization: A survey, IEEE Commun. Surv. Tutorials 15 (2) (2013) 909–928. doi:10.1109/SURV.2012.090512.00043.
URL https://doi.org/10.1109/SURV.2012.090512.00043

[3] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, K. J. Eliazar, Why does the cloud stop computing? lessons from hundreds of service outages, in: M. K. Aguilera, B. Cooper, Y. Diao (Eds.), Proceedings of the Seventh ACM Symposium on Cloud Computing, Santa Clara, CA, USA, October 5-7, 2016, ACM, 2016, pp. 1–16. doi:10.1145/2987550.2987583.
URL https://doi.org/10.1145/2987550.2987583

[4] P. G. López, A. Montresor, D. H. J. Epema, A. Datta, T. Higashino, A. Iamnitchi, M. P. Barcellos, P. Felber, E. Rivière, Edge-centric computing: Vision and challenges, Comput. Commun. Rev. 45 (5) (2015) 37–42. doi:10.1145/2831347.2831354.
URL https://doi.org/10.1145/2831347.2831354

[5] M. B. A. Karim, B. I. Ismail, M. Wong, E. M. Goortani, S. Setapa, L. J. Yuan, H. Ong, Extending cloud resources to the edge: Possible scenarios, challenges, and experiments, in: International Conference on Cloud Computing Research and Innovations, ICCCRI 2016, Singapore, Singapore, May 4-5, 2016, IEEE Computer Society, 2016, pp. 78–85. doi:10.1109/ICCCRI.2016.20.
URL https://doi.org/10.1109/ICCCRI.2016.20

[6] S. A. Monsalve, F. G. Carballeira, A. Calderón, A heterogeneous mobile cloud computing model for hybrid clouds, Future Gener. Comput. Syst. 87 (2018) 651–666. doi:10.1016/j.future.2018.04.005.
URL https://doi.org/10.1016/j.future.2018.04.005

[7] M. Satyanarayanan, The emergence of edge computing, Computer 50 (1) (2017) 30–39. doi:10.1109/MC.2017.9.
URL https://doi.org/10.1109/MC.2017.9

[8] S. K. A. Hossain, M. A. Rahman, M. A. Hossain, Edge computing framework for enabling situation awareness in iot based smart city, J. Parallel Distributed Comput. 122 (2018) 226–237. doi:10.1016/j.jpdc.2018.08.009.
URL https://doi.org/10.1016/j.jpdc.2018.08.009

[9] J. Cao, Q. Zhang, W. Shi, Edge Computing: A Primer, Springer Briefs in Computer Science, Springer, 2018. doi:10.1007/978-3-030-02083-5.
URL https://doi.org/10.1007/978-3-030-02083-5

[10] H. Ning, Y. Li, F. Shi, L. T. Yang, Heterogeneous edge computing open platforms and tools for internet of things, Future Gener. Comput. Syst. 106 (2020) 67–76. doi:10.1016/j.future.2019.12.036.
URL https://doi.org/10.1016/j.future.2019.12.036

[11] A. S. Tanenbaum, M. van Steen, Distributed systems - principles and paradigms, 2nd Edition, Pearson Education, 2007.

[12] M. van Steen, A. S. Tanenbaum, A brief introduction to distributed systems, Computing 98 (10) (2016) 967–1009. doi:10.1007/s00607-016-0508-7.
URL https://doi.org/10.1007/s00607-016-0508-7

[13] A. B. Bondi, Characteristics of scalability and their impact on performance, in: Second International Workshop on Software and Performance, WOSP 2000, Ottawa, Canada, September 17-20, 2000, ACM, 2000, pp. 195–203. doi:10.1145/350391.350432.
URL https://doi.org/10.1145/350391.350432

[14] J. L. Gustafson, Moore's Law, Springer US, Boston, MA, 2011, pp. 1177–1184. doi:10.1007/978-0-387-09766-4_81.
URL https://doi.org/10.1007/978-0-387-09766-4_81

[15] E. A. Brewer, Towards robust distributed systems., in: Symposium on Principles of Distributed Computing (PODC), 2000.
URL http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

[16] S. Gilbert, N. A. Lynch, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services, SIGACT News 33 (2) (2002) 51–59. doi:10.1145/564585.564601.
URL https://doi.org/10.1145/564585.564601

[17] M. Shapiro, N. M. Preguiça, C. Baquero, M. Zawirski, Conflict-free replicated data types, in: X. Défago, F. Petit, V. Villain (Eds.), Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings, Vol. 6976 of Lecture Notes in Computer Science, Springer, 2011, pp. 386–400. doi:10.1007/978-3-642-24550-3_29.
URL https://doi.org/10.1007/978-3-642-24550-3_29

[18] W. Vogels, A head in the clouds the power of infrastructure as a service, in: Proceedings of the 1st Workshop on Cloud Computing and Applications, 2008.

[19] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, M. Zaharia, Above the clouds: A berkeley view of cloud computing, Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley (Feb 2009).
URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html

[20] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, J. Internet Serv. Appl. 1 (1) (2010) 7–18. doi:10.1007/s13174-010-0007-6.
URL https://doi.org/10.1007/s13174-010-0007-6

[21] M. D. de Assunção, A. D. S. Veith, R. Buyya, Distributed data stream processing and edge computing: A survey on resource elasticity and future directions, J. Netw. Comput. Appl. 103 (2018) 1–17. doi:10.1016/j.jnca.2017.12.001.
URL https://doi.org/10.1016/j.jnca.2017.12.001

[22] P. M. Mell, T. Grance, Sp 800-145. the nist definition of cloud computing, Tech. rep., Gaithersburg, MD, USA (2011).

[23] D. Cohen, T. Talpey, A. Kanevsky, U. Cummings, M. Krause, R. Recio, D. Crupnicoff, L. Dickman, P. Grun, Remote direct memory access over the converged enhanced ethernet fabric: Evaluating the options, in: K. Bergman, R. Brightwell, F. Petrini, H. Bubba (Eds.), 17th IEEE Symposium on High Performance Interconnects, HOTI 2009, New York, New York, USA, August 25-27, 2009, IEEE Computer Society, 2009, pp. 123–130. doi:10.1109/HOTI.2009.23.
URL https://doi.org/10.1109/HOTI.2009.23

[24] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, B. Hu, Everything as a service (xaas) on the cloud: Origins, current and future trends, in: C. Pu, A. Mohindra (Eds.), 8th IEEE International Conference on Cloud Computing, CLOUD 2015, New York City, NY, USA, June 27 - July 2, 2015, IEEE Computer Society, 2015, pp. 621–628. doi:10.1109/CLOUD.2015.88.
URL https://doi.org/10.1109/CLOUD.2015.88

[25] R. Schollmeier, A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications, in: R. L. Graham, N. Shahmehri (Eds.), 1st International Conference on Peer-to-Peer Computing (P2P 2001), 27-29 August 2001, Linköping, Sweden, IEEE Computer Society, 2001, pp. 101–102. doi:10.1109/P2P.2001.990434.
URL https://doi.org/10.1109/P2P.2001.990434

[26] H. M. N. D. Bandara, A. P. Jayasumana, Collaborative applications over peer-to-peer systems-challenges and solutions, Peer Peer Netw. Appl. 6 (3) (2013) 257–276. doi:10.1007/s12083-012-0157-3.
URL https://doi.org/10.1007/s12083-012-0157-3

[27] M. Kamel, C. M. Scoglio, T. Easton, Optimal topology design for overlay networks, in: I. F. Akyildiz, R. Sivakumar, E. Ekici, J. C. de Oliveira, J. McNair (Eds.), NETWORKING 2007. Ad Hoc and Sensor Networks, Wireless Networks, Next Generation Internet, 6th International IFIP-TC6 Networking Conference, Atlanta, GA, USA, May 14-18, 2007, Proceedings, Vol. 4479 of Lecture Notes in Computer Science, Springer, 2007, pp. 714–725. doi:10.1007/978-3-540-72606-7_61.
URL https://doi.org/10.1007/978-3-540-72606-7_61

[28] I. Filali, F. Bongiovanni, F. Huet, F. Baude, A survey of structured P2P systems for RDF data storage and retrieval, Trans. Large Scale Data Knowl. Centered Syst. 3 (2011) 20–55. doi:

10.1007/978-3-642-23074-5_2.
URL https://doi.org/10.1007/978-3-642-23074-5_2

[29] A. Das, I. Gupta, A. Motivala, SWIM: scalable weakly-consistent infection-style process group membership protocol, in: 2002 International Conference on Dependable Systems and Networks (DSN 2002), 23-26 June 2002, Bethesda, MD, USA, Proceedings, IEEE Computer Society, 2002, pp. 303–312. doi:10.1109/DSN.2002.1028914.
URL https://doi.org/10.1109/DSN.2002.1028914

[30] I. Stoica, R. T. Morris, D. R. Karger, M. F. Kaashoek, H. Balakrishnan, Chord: A scalable peer-to-peer lookup service for internet applications, in: R. L. Cruz, G. Varghese (Eds.), Proceedings of the ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, August 27-31, 2001, San Diego, CA, USA, ACM, 2001, pp. 149–160. doi:10.1145/383059.383071.
URL https://doi.org/10.1145/383059.383071

[31] N. Leavitt, Will nosql databases live up to their promise?, Computer 43 (2) (2010) 12–14. doi:10.1109/MC.2010.58.
URL https://doi.org/10.1109/MC.2010.58

[32] Q. H. Vu, M. Lupu, B. C. Ooi, Peer-to-Peer Computing - Principles and Applications, Springer, 2010. doi:10.1007/978-3-642-03514-2.
URL https://doi.org/10.1007/978-3-642-03514-2

[33] N. Fernando, S. W. Loke, J. W. Rahayu, Mobile cloud computing: A survey, Future Gener. Comput. Syst. 29 (1) (2013) 84–106. doi:10.1016/j.future.2012.05.023.
URL https://doi.org/10.1016/j.future.2012.05.023

[34] L. Lin, X. Liao, H. Jin, P. Li, Computation offloading toward edge computing, Proc. IEEE 107 (8) (2019) 1584–1607. doi:

10.1109/JPROC.2019.2922285.
URL https://doi.org/10.1109/JPROC.2019.2922285

[35] F. Bonomi, R. A. Milito, P. Natarajan, J. Zhu, Fog computing: A platform for internet of things and analytics, in: N. Bessis, C. Dobre (Eds.), Big Data and Internet of Things: A Roadmap for Smart Environments, Vol. 546 of Studies in Computational Intelligence, Springer, 2014, pp. 169–186. doi:10.1007/978-3-319-05029-4_7.
URL https://doi.org/10.1007/978-3-319-05029-4_7

[36] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, W. Wang, A survey on mobile edge networks: Convergence of computing, caching and communications, IEEE Access 5 (2017) 6757–6779. doi:10.1109/ACCESS.2017.2685434.
URL https://doi.org/10.1109/ACCESS.2017.2685434

[37] A. Khune, S. Pasricha, Mobile network-aware middleware framework for cloud offloading: Using reinforcement learning to make reward-based decisions in smartphone applications, IEEE Consumer Electron. Mag. 8 (1) (2019) 42–48. doi:10.1109/MCE.2018.2867972.
URL https://doi.org/10.1109/MCE.2018.2867972

[38] M. Chen, Y. Hao, Y. Li, C. Lai, D. Wu, On the computation offloading at ad hoc cloudlet: architecture and service modes, IEEE Commun. Mag. 53 (6-Supplement) (2015) 18–24. doi:10.1109/MCOM.2015.7120041.
URL https://doi.org/10.1109/MCOM.2015.7120041

[39] M. Satyanarayanan, G. Klas, M. D. Silva, S. Mangiante, The seminal role of edge-native applications, in: E. Bertino, C. K. Chang, P. Chen, E. Damiani, M. Goul, K. Oyama (Eds.), 3rd IEEE International Conference on Edge Computing, EDGE 2019, Milan, Italy, July 8-13, 2019, IEEE, 2019, pp. 33–40. doi:10.

1109/EDGE.2019.00022.
URL https://doi.org/10.1109/EDGE.2019.00022

[40] R. de Vera Jr., Review of: Distributed systems: An algorithmic approach (2nd edition) by sukumar ghosh, SIGACT News 47 (4) (2016) 13–14. doi:10.1145/3023855.3023860.
URL https://doi.org/10.1145/3023855.3023860

[41] G. Andrews, , parallel, and distributed programming, Addison-Wesley, 2000.
URL http://books.google.com.br/books?id=npRQAAAAMAAJ

[42] D. Fisher, R. DeLine, M. Czerwinski, S. M. Drucker, Interactions with big data analytics, Interactions 19 (3) (2012) 50–59. doi:10.1145/2168931.2168943.
URL https://doi.org/10.1145/2168931.2168943

[43] C.-W. Tsai, C.-F. Lai, H.-C. Chao, A. V. Vasilakos, Big data analytics: a survey, Journal of Big Data 2 (1) (2015) 21. doi:10.1186/s40537-015-0030-3.
URL https://doi.org/10.1186/s40537-015-0030-3

[44] Z. Guo, G. C. Fox, M. Zhou, Investigation of data locality in mapreduce, in: 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2012, Ottawa, Canada, May 13-16, 2012, IEEE Computer Society, 2012, pp. 419–426. doi:10.1109/CCGrid.2012.42.
URL https://doi.org/10.1109/CCGrid.2012.42

[45] P. G. Sarigiannidis, T. Lagkas, K. Rantos, P. Bellavista, The big data era in iot-enabled smart farming: Re-defining systems, tools, and techniques, Comput. Networks 168 (2020). doi:10.1016/j.comnet.2019.107043.
URL https://doi.org/10.1016/j.comnet.2019.107043

[46] R. Patgiri, A. Ahmed, Big data: The v's of the game changer paradigm, in: J. Chen, L. T. Yang (Eds.), 18th IEEE International Conference on High Performance Computing and Communications; 14th IEEE International Conference on Smart City; 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016, Sydney, Australia, December 12-14, 2016, IEEE Computer Society, 2016, pp. 17–24. doi:10.1109/HPCC-SmartCity-DSS.2016.0014.
URL https://doi.org/10.1109/HPCC-SmartCity-DSS.2016.0014

[47] Y. Kumar, Lambda architecture - realtime data processing, Ph.D. thesis (01 2020). doi:10.13140/RG.2.2.19091.84004.

[48] M. Kiran, P. Murphy, I. Monga, J. Dugan, S. S. Baveja, Lambda architecture for cost-effective batch and speed big data processing, in: 2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015, IEEE Computer Society, 2015, pp. 2785–2792. doi:10.1109/BigData.2015.7364082.
URL https://doi.org/10.1109/BigData.2015.7364082

[49] T. R. Rao, P. Mitra, R. Bhatt, A. Goswami, The big data system, components, tools, and technologies: a survey, Knowl. Inf. Syst. 60 (3) (2019) 1165–1245. doi:10.1007/s10115-018-1248-0.
URL https://doi.org/10.1007/s10115-018-1248-0

[50] M. Simic, M. Stojkov, G. Sladic, B. Milosavljević, Edge computing system for large-scale distributed sensing systems, 2018.

[51] J. E. Marynowski, A. O. Santin, A. R. Pimentel, Method for testing the fault tolerance of mapreduce frameworks, Comput. Networks 86 (2015) 1–13. doi:10.1016/j.comnet.2015.04.009.
URL https://doi.org/10.1016/j.comnet.2015.04.009

[52] N. Dragoni, S. Giallorenzo, A. Lluch-Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, Microservices: yesterday, today, and tomorrow, CoRR abs/1606.04036 (2016). arXiv:1606.04036.
URL http://arxiv.org/abs/1606.04036

[53] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, N. M. Josuttis, Microservices in practice, part 1: Reality check and service design, IEEE Softw. 34 (1) (2017) 91–98. doi:10.1109/MS.2017.24.
URL https://doi.org/10.1109/MS.2017.24

[54] S. Li, H. Zhang, Z. Jia, Z. Li, C. Zhang, J. Li, Q. Gao, J. Ge, Z. Shan, A dataflow-driven approach to identifying microservices from monolithic applications, J. Syst. Softw. 157 (2019). doi:10.1016/j.jss.2019.07.008.
URL https://doi.org/10.1016/j.jss.2019.07.008

[55] L. Krause, Microservices: Patterns and Applications: Designing Fine-Grained Services by Applying Patterns, Lucas Krause, 2015.
URL https://books.google.rs/books?id=dd5-rgEACAAJ

[56] O. Al-Debagy, P. Martinek, A comparative review of microservices and monolithic architectures, CoRR abs/1905.07997 (2019). arXiv:1905.07997.
URL http://arxiv.org/abs/1905.07997

[57] N. Kratzke, P. Quint, Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study, J. Syst. Softw. 126 (2017) 1–16. doi:10.1016/j.jss.2017.01.001.
URL https://doi.org/10.1016/j.jss.2017.01.001

[58] G. Adzic, R. Chatley, Serverless computing: economic and architectural impact, in: E. Bodden, W. Schäfer, A. van Deursen, A. Zisman (Eds.), Proceedings of the 2017 11th Joint Meeting on

Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017, ACM, 2017, pp. 884–889. doi:10.1145/3106237.3117767.
URL https://doi.org/10.1145/3106237.3117767

[59] W. Li, Y. Lemieux, J. Gao, Z. Zhao, Y. Han, Service mesh: Challenges, state of the art, and future research opportunities, in: 13th IEEE International Conference on Service-Oriented System Engineering, SOSE 2019, San Francisco, CA, USA, April 4-9, 2019, IEEE, 2019. doi:10.1109/SOSE.2019.00026.
URL https://doi.org/10.1109/SOSE.2019.00026

[60] A. Balalaie, A. Heydarnoori, P. Jamshidi, Microservices architecture enables devops: Migration to a cloud-native architecture, IEEE Software 33 (3) (2016) 42–52. doi:10.1109/MS.2016.64.

[61] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: 2015 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2015, Philadelphia, PA, USA, March 29-31, 2015, IEEE Computer Society, 2015, pp. 171–172. doi:10.1109/ISPASS.2015.7095802.
URL https://doi.org/10.1109/ISPASS.2015.7095802

[62] B. Burns, B. Grant, D. Oppenheimer, E. A. Brewer, J. Wilkes, Borg, omega, and kubernetes, Commun. ACM 59 (5) (2016) 50–57. doi:10.1145/2890784.
URL https://doi.org/10.1145/2890784

[63] J. Soldani, D. A. Tamburri, W. van den Heuvel, The pains and gains of microservices: A systematic grey literature review, J. Syst. Softw. 146 (2018) 215–232. doi:10.1016/j.jss.2018.09.082.
URL https://doi.org/10.1016/j.jss.2018.09.082

[64] G. S. Almási, A. Gottlieb, Highly parallel computing (2. ed.), Addison-Wesley, 1994.

[65] S. Leible, S. Schlager, M. Schubotz, B. Gipp, A review on blockchain technology and blockchain projects fostering open science, Frontiers Blockchain 2 (2019) 16. doi:10.3389/fbloc.2019.00016.
URL https://doi.org/10.3389/fbloc.2019.00016

[66] S. Crosby, D. Brown, The virtualization reality, ACM Queue 4 (10) (2006) 34–41. doi:10.1145/1189276.1189289.
URL https://doi.org/10.1145/1189276.1189289

[67] S. Sharma, Y. Park, Virtualization: A review and future directions executive overview, American Journal of Information Technology 1 (2011) 1–37.

[68] E. E. Absalom, S. M. Buhari, S. B. Junaidu, Virtual machine allocation in cloud computing environment, Int. J. Cloud Appl. Comput. 3 (2) (2013) 47–60. doi:10.4018/ijcac.2013040105.
URL https://doi.org/10.4018/ijcac.2013040105

[69] C. Yang, K. Huang, W. C. Chu, F. Leu, S. Wang, Implementation of cloud iaas for virtualization with live migration, in: J. J. Park, H. R. Arabnia, C. Kim, W. Shi, J. Gil (Eds.), Grid and Pervasive Computing - 8th International Conference, GPC 2013 and Colocated Workshops, Seoul, Korea, May 9-11, 2013. Proceedings, Vol. 7861 of Lecture Notes in Computer Science, Springer, 2013, pp. 199–207. doi:10.1007/978-3-642-38027-3_21.
URL https://doi.org/10.1007/978-3-642-38027-3_21

[70] K.-T. Seo, H.-S. Hwang, I. Moon, O.-Y. Kwon, B. jun Kim, Performance comparison analysis of linux container and virtual machine for building cloud, 2014.

[71] R. Pavlicek, Unikernels: Beyond Containers to the Next Generation of Cloud, O'Reilly Media, 2016.
URL https://books.google.rs/books?id=qfDXuQEACAAJ

[72] T. Goethals, M. Sebrechts, A. Atrey, B. Volckaert, F. D. Turck, Unikernels vs containers: An in-depth benchmarking study in the context of microservice applications, in: 8th IEEE International Symposium on Cloud and Service Computing, SC2 2018, Paris, France, November 18-21, 2018, IEEE, 2018, pp. 1–8. doi:10.1109/SC2.2018.00008.
URL https://doi.org/10.1109/SC2.2018.00008

[73] R. Pavlicek, The next generation cloud: Unleashing the power of the unikernel, USENIX Association, Washington, D.C., 2015.

[74] M. Plauth, L. Feinbube, A. Polze, A performance survey of lightweight virtualization techniques, in: F. D. Paoli, S. Schulte, E. B. Johnsen (Eds.), Service-Oriented and Cloud Computing - 6th IFIP WG 2.14 European Conference, ESOCC 2017, Oslo, Norway, September 27-29, 2017, Proceedings, Vol. 10465 of Lecture Notes in Computer Science, Springer, 2017, pp. 34–48. doi:10.1007/978-3-319-67262-5_3.
URL https://doi.org/10.1007/978-3-319-67262-5_3

[75] R. Pike, Concurrency is not parallelism, waza conference (2013).
URL https://blog.golang.org/waza-talk

[76] C. A. R. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677. doi:10.1145/359576.359585.
URL https://doi.org/10.1145/359576.359585

[77] C. Hewitt, Actor model for discretionary, adaptive concurrency, CoRR abs/1008.1459 (2010). arXiv:1008.1459.
URL http://arxiv.org/abs/1008.1459

[78] A. G. Greenberg, J. R. Hamilton, D. A. Maltz, P. Patel, The cost of a cloud: research problems in data center networks, Comput. Commun. Rev. 39 (1) (2009) 68–73. doi:10.1145/1496091.1496103.
URL https://doi.org/10.1145/1496091.1496103

[79] M. Satyanarayanan, P. Bahl, R. Cáceres, N. Davies, The case for vm-based cloudlets in mobile computing, IEEE Pervasive Comput. 8 (4) (2009) 14–23. doi:10.1109/MPRV.2009.82.
URL https://doi.org/10.1109/MPRV.2009.82

[80] M. Ryden, K. Oh, A. Chandra, J. B. Weissman, Nebula: Distributed edge cloud for data intensive computing, in: 2014 IEEE International Conference on Cloud Engineering, Boston, MA, USA, March 11-14, 2014, IEEE Computer Society, 2014, pp. 57–66. doi:10.1109/IC2E.2014.34.
URL https://doi.org/10.1109/IC2E.2014.34

[81] M. Hirsch, C. Mateos, A. Zunino, Augmenting computing capabilities at the edge by jointly exploiting mobile devices: A survey, Future Gener. Comput. Syst. 88 (2018) 644–662. doi:10.1016/j.future.2018.06.005.
URL https://doi.org/10.1016/j.future.2018.06.005

[82] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (iot): A vision, architectural elements, and future directions, Future Gener. Comput. Syst. 29 (7) (2013) 1645–1660. doi:10.1016/j.future.2013.01.010.
URL https://doi.org/10.1016/j.future.2013.01.010

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

[84] R. V. Aroca, L. M. G. Gonçalves, Towards green data centers: A comparison of x86 and ARM architectures power efficiency, J. Parallel Distributed Comput. 72 (12) (2012) 1770–1780. doi:10.1016/j.jpdc.2012.08.005.
URL https://doi.org/10.1016/j.jpdc.2012.08.005

[85] H. Guo, L. Rui, Z. Gao, A zone-based content pre-caching strategy in vehicular edge networks, Future Gener. Comput. Syst. 106 (2020) 22–33. doi:10.1016/j.future.2019.12.050.
URL https://doi.org/10.1016/j.future.2019.12.050

[86] A. C. Baktir, A. Ozgovde, C. Ersoy, How can edge computing benefit from software-defined networking: A survey, use cases, and future directions, IEEE Commun. Surv. Tutorials 19 (4) (2017) 2359–2391. doi:10.1109/COMST.2017.2717482.
URL https://doi.org/10.1109/COMST.2017.2717482

[87] I. Kurniawan, H. Febiansyah, J. Kwon, Cost-Effective Content Delivery Networks Using Clouds and Nano Data Centers, Vol. 280, 2014, pp. 417–424. doi:10.1007/978-3-642-41671-2_53.

[88] R. Ciobanu, C. Negru, F. Pop, C. Dobre, C. X. Mavromoustakis, G. Mastorakis, Drop computing: Ad-hoc dynamic collaborative computing, Future Gener. Comput. Syst. 92 (2019) 889–899. doi:10.1016/j.future.2017.11.044.
URL https://doi.org/10.1016/j.future.2017.11.044

[89] C. Shi, K. Habak, P. Pandurangan, M. H. Ammar, M. Naik, E. W. Zegura, COSMOS: computation offloading as a service for mobile devices, in: J. Wu, X. Cheng, X. Li, S. Sarkar (Eds.), The Fifteenth ACM International Symposium on Mobile Ad Hoc Networking and Computing, MobiHoc'14, Philadelphia, PA, USA, August 11-14, 2014, ACM, 2014, pp. 287–296. doi:10.1145/2632951.2632958.
URL https://doi.org/10.1145/2632951.2632958

[90] N. Abbas, Y. Zhang, A. Taherkordi, T. Skeie, Mobile edge computing: A survey, IEEE Internet Things J. 5 (1) (2018) 450–465. doi:10.1109/JIOT.2017.2750180.
URL https://doi.org/10.1109/JIOT.2017.2750180

[91] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, J. Wilkes, Large-scale cluster management at google with borg, in: L. Réveillère, T. Harris, M. Herlihy (Eds.), Proceedings of the Tenth European Conference on Computer Systems, EuroSys 2015, Bordeaux, France, April 21-24, 2015, ACM, 2015, pp. 18:1–18:17. doi:10.1145/2741948.2741964.
URL https://doi.org/10.1145/2741948.2741964

[92] F. Rossi, V. Cardellini, F. L. Presti, M. Nardelli, Geo-distributed efficient deployment of containers with kubernetes, Comput. Commun. 159 (2020) 161–174. doi:10.1016/j.comcom.2020.04.061.
URL https://doi.org/10.1016/j.comcom.2020.04.061

[93] B. Yee, D. Sehr, G. Dardyk, J. B. Chen, R. Muth, T. Ormandy, S. Okasaka, N. Narula, N. Fullagar, Native client: a sandbox for portable, untrusted x86 native code, Commun. ACM 53 (1) (2010) 91–99. doi:10.1145/1629175.1629203.
URL https://doi.org/10.1145/1629175.1629203

[94] A. Lèbre, J. Pastor, A. Simonet, F. Desprez, Revising openstack to operate fog/edge computing infrastructures, in: 2017 IEEE International Conference on Cloud Engineering, IC2E 2017, Vancouver, BC, Canada, April 4-7, 2017, IEEE Computer Society, 2017, pp. 138–148. doi:10.1109/IC2E.2017.35.
URL https://doi.org/10.1109/IC2E.2017.35

[95] Y. Shao, C. Li, Z. Fu, L. Jia, Y. Luo, Cost-effective replication management and scheduling in edge computing, J. Netw. Comput. Appl. 129 (2019) 46–61. doi:10.1016/j.jnca.2019.01.

001.
URL https://doi.org/10.1016/j.jnca.2019.01.001

[96] A. Kurniawan, Learning AWS IoT: Effectively manage connected devices on the AWS cloud using services such as AWS Greengrass, AWS button, predictive analytics and machine learning, Packt Publishing, 2018.
URL https://books.google.rs/books?id=7NRJDwAAQBAJ

[97] Linux Foundation, KubeEdge, https://kubeedge.io/ (accessed November 7, 2020).

[98] General Electric, GE. Predix, https://www.ge.com/digital/iiot-platform/ (accessed November 7, 2020).