Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of ICT and Natural Sciences

Torbjørn Kirkevik Soltvedt

A Distributed-to-Centralized Cost Model for Service Selection in Smart Cities

Master's thesis in Master of Science (MSc) in Informatics
Supervisor: Sobah Abbas Petersen, Amir Sinaeepourfard
June 2019

NTNU
Norwegian University of
Science and Technology

Torbjørn Kirkevik Soltvedt

A Distributed-to-Centralized Cost Model for Service Selection in Smart Cities

Master's thesis in Master of Science (MSc) in Informatics
Supervisor: Sobah Abbas Petersen, Amir Sinaeepourfard
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Problem Description

In a distributed-to-centralized architecture for managing the data produced by the smart cities, services and data is made accessible from different actors in different places in the architecture. In order to be able to select the best services and data, an approach is needed to manage the services and data, by helping to find and select them. In order to help select the most suitable alternatives with respect to their cost and benefits, a cost model is to be proposed as part of a selection approach. It is then implemented to work with a distributed-to-centralized ICT data management architecture for collecting and managing data from zero-emission neighborhoods. Which will be evaluated if it is fit for use in managing the selection of services and data in the context of a smart city.

Abstract

With a growing urban population, and new enabling technologies, it is predicted that the future smart city is going to generate large amounts of data that needs be managed in order for it to be used in a manner that produces value for the smart city. A new data management architecture known as a distributed-to-centralized data management architecture has been proposed to help address these data challenges. But there is also a need for an approach to find and select data and services in that architecture for the smart city.

This thesis proposes a cost model that is used to rank data and services as part of the selection approach within the context of a distributed-to-centralized data management architecture for the smart city. It describes which Key Performance Indicators (KPI) that can be used by the cost model to rank different alternatives, as well as how to store and describe the information about data and services. It then proposes a design for this cost model to be implemented in the context of an instance of this architecture designed for FME ZEN research centre for its pilot projects, know as the ZEN ICT architecture. This architecture is responsible for gathering the ZEN KPIs from its pilot project buildings, which are measurements of neighborhood and building performance. But as the distributed-to-centralized architecture is also likely to be beneficial to other smart city architectures, the design of a cost model in this thesis would likely also be beneficial in those cases. The implementation of the design into a prototype is done to see how it is able to find and rank data for selection, and evaluate it based on its performance, proving feasibility and if it is fit for its intended application context. This can hopefully help finding the most suitable data in the ZEN ICT architecture, connecting the data in the architecture with the applications and services that needs that data.

Sammendrag

Med en voksende urban befolkning, og nye støttende teknologier, er det antatt at den fremtidige smarte byen kommer til å generere store mengder data som trenger å bli administrert for å kunne bli brukt på en måte slik at den produserer verdi for den smarte byen. En dataadministreringsarkitektur kjent som en distribuert-til-sentralisert dataadministreringsarkitektur har blitt foreslått for å svare pø disse datautfordringene. Men det er også behov for en framgangsmåte for å finne og velge data og tjenester i den arkitekturen for den smarte byen.

Denne oppgaven foreslår en kostnadsmodell som er brukt til å rangere data og tjenester som en del av en seleksjonsframgangsmåte i kontekst av en distribuert-til-sentralisert dataadministreringsarkitektur for den smarte byen. Den beskriver hvilke viktige nøkkeltall (KPI) som kan brukes av kostnadsmodellen for å rangere ulike alternativ, i tillegg til hvordan man kan lagre og beskrive informasjonen om data og tjenester. Så forslår den et design for hvordan denne kostnadsmodellen kan bli implementert i kontekst av en forekomst av denne arkitekturen designet for FME ZEN forskningssenter sine pilotprosjekter, kjent som ZEN ICT-arkitekturen. Denne arkitekturen har ansvar for å innhente ZEN sine viktige nøkkeltall fra pilotprojektbygningene, som er målinger av nabolag- og bygningsprestasjon. Men siden den distribuert-til-sentraliserte arkitekturen også sannsynligvis er gunstig for andre smart by-arkitekturer, så er designet av kostnadsmodellen i denne oppgaven også gunstig å bruke for de tilfellene. Implementasjonen av en prototype basert på designet er gjort for å se hvordan den klarer å finne og rangere data for seleksjon, og for å evaluere den basert på dens prestasjoner, for å bevise at det lar seg gjøre og at den passer for konteksten den skal brukes i. Det kan forhåpentligvis hjelpe med å finne den mest passende dataen i ZEN ICT-arkitekturen, og koble sammen data i arkitekturen med applikasjoner og tjenester som trenger data.

Preface

This thesis is final part of a Master of Science degree in informatics with a specialization in software engineering, at the Norwegian University of Science and Technology (NTNU). This thesis and project was done under the Department of Computer Science (IDI), and FME ZEN research centre.

I would like to thank Amir Sinaeepourfard and Sobah Abbas Petersen, from the Department of Computer Science, for their support and guidance as supervisors during this project.

Torbjørn Kirkevik Soltvedt
June 2019

Table of Contents

List of Tables

List of Figures

Abbreviations

ICT	=	Information and Communications Technology
ZEN	=	Zero Emissions Neighbourhoods
KPI	=	Key Performance Indicator
IoT	=	Internet of Things
F2C	=	Fog-to-Cloud
SDN	=	Software Defined Networking
SLA	=	Service-Level Agreement
DHT	=	Distributed Hash Table
GUI	=	Graphical User Interface
QoS	=	Quality of Service
QoE	=	Quality of Experience
MAUT	=	Multi-Attribute Utility Theory
P2P	=	Peer-to-peer
PoW	=	Proof-of-Work
XML	=	Extensible Markup Language

Chapter 1

Introduction

1.1 Introduction

According to an UN report on the urbanization of the world, 55% of the world's population lives in urban areas, which is a number that is expected to rise to around 68% in the year 2050 [143]. In the same period, the total population of the world is expected to rise to 9.8 billion, from the 7.3 billion currently in the world [144]. In order to meet new demands of sustainability, environmental friendliness and maintaining a high quality of life in the cities as more people make them their homes, new approaches to building and managing cities is needed.

Researchers around the world envision new ways to build and manage the cities in order to improve them, and one of more central terms used for the new cities, is the term "smart city". Even though there is not a single agreed upon definition of the term, and its not even the only term used to describe this new type of city, one thing that remains in agreement in almost all the of definitions. The use of technology, specifically Information and Communication Technology (ICT) to plan, manage and govern the city and the services.

With new enabling technologies, that connects all of the city through new network technologies that connects the whole city, together with advances in Internet of Things (IoT) technology, new ways to gather data and sense the city is a central part of that. Which means that with the smart city, also comes an expected exponential growth in the amount of data generated [56]. To deal with these new data challenges, an architecture have been proposed to help with the management of data in the smart city, by making up for the limitations of the old centralized models, through combining the distributed data management with the centralized cloud. In this architecture for data management, called a distributed-to-centralized architecture, different actors within the city can gather, store and process their own data and share it with others through this architecture, in different locations in the overall system.

But for services and consumers of data, this opens up new challenges in how to find and select data and services in this architecture. Services in the smart city is dependent on the data generated in the smart city, as well as other services in order to provide their services.

And to help with the selection of the most suitable data or service, this thesis proposes a cost model that is used as part of this selection. A cost model is in this case an approach or a method create a ranking of different alternatives according to how suitable they are depending on the users need, with respect to cost and benefits of the alternatives. Which in turn can be used to select the best alternative.

1.2 Motivation

This thesis is motivated by the work done Zero Emissions Neighbourhoods (ZEN) research centre [1]. ZEN is working on different solutions with the goal of developing neighborhoods with zero greenhouse gas emissions. Part of that work involves different, multidisciplinary research, but most relevant for this thesis is the work on a data management and the pilot projects around Norway that will generate data for the research that needs to managed, for it later to be used as part of testing and analysis of the solutions. This is specifically tied to research done in [126, 127], on the ZEN ICT architecture, which is more described in section 2.10.3. Within the context data management for the ZEN pilot projects, finding a way to find and select the most suitable data in the underlying data management is needed as part of analysing and using of the data. For ZEN this will provide a connection between the collecting, processing, and storage provided by the ZEN ICT architecture, and the software tools used to analyze and visualize that data.

1.3 Project Description

This thesis is part of the work on the ZEN ICT architecture, for managing the data created by the pilot project, and based upon work done in [126, 127] and by ZEN. It is also part of a large body of work based on distributed-to-centralized architectures and fog-to-cloud (F2C) architectures.

The goal of this thesis is to create and implement a cost model that is used to help select among services and data in a distributed-to-centralized data management architecture. A cost model is in this case a function or method that creates a ranking of service or data according to the specified needs of the user, letting the user get the most benefit for the cost of accessing it. The implementation will be a prototype or a proof of concept on how to select data in the ZEN ICT architecture.

Part of this involves a literature review of the different cost model approaches, the different KPI that should be used by the cost model, and how to find and store information about services and data, as part of creating the design and implementation. After that, the implementation is evaluated and discussed, to see if it is an approach that is suitable to be used in this domain.

[1] FME ZEN: https://fmezen.no/

1.4 Thesis Outline

The first chapter after this, is **Chapter 2**, which contains the theoretical background on smart cities, distributed-to-centralized architectures, cost model approaches, relevant KPIs to use with the cost model, approaches to store and describe data and services, and background on ZEN. **Chapter 3** contains the research methodology for this thesis together with the research questions identified. **Chapter 4** presents the design of a cost model approach, together with stakeholders and requirements for it. **Chapter 5** presents the implementation of the cost model as a prototype, explaining how it was implemented. **Chapter 6** presents the results from the implementation in terms of functionality and performance measurements. **Chapter 7** is a discussion of the thesis and implementation, together with an evaluation of the implementation. **Chapter 8** is the final chapter, where a conclusion of the thesis is presented, research questions are answered and future work discussed.

Chapter 2

Background

The background for this thesis is based on attempting to create a cost model that can be used for selecting services and data in a distributed-to-centralized data management architecture in the context of a smart city. The specific architecture looked at and which the design is implemented for, is the ZEN ICT architecture, which will be presented in the background. The background chapter start with an introduction of what a smart city is, before looking at some of the differences between a centralized schema versus a distributed-to-centralized schema. It then looks at different approaches to cost models used to rank multi-criteria alternatives, before looking at different KPIs that can be used with the cost model to select among services and data. It then looks at different ways to store information about services and data in the section, before looking at how to describe them. At the end of the chapter, background regarding ZEN is explained and how it relates to this thesis.

2.1 Smart cities

In [6] and [30], it is shown that the literature about smart cities does not have a single agreed upon definition of what constitutes a smart city. Common for many of the definitions is the use of ICT to improve different aspects of the city, the governance of the city, the services provided, the sustainability and the quality of life in the city as some of the many aspects and dimensions proposed. One of the most cited definitions of a smart city that encapsulates the ideas in this thesis by including the use of ICT as well as the focus on sustainability and quality of life is from [23] and is as follows:
”A city is smart when investments in human and social capital and traditional (transport) and modern (ICT) communication infrastructure fuel sustainable economic growth and a high quality of life, with a wise management of natural resources, through participatory governance”.
Among the different definitions, general themes and dimensions exists that contains the different ideas of what constitutes a smart city. In [83], these dimensions are categorized

into three different categories, and are the technology dimension, the human dimension, and the institutional dimension.
The technological dimension is focused on the ICT of the city, where communication technology is part of the infrastructure of the city that connects the city. Where the virtual space of the city is as much a part of the city as the physical parts and access to the services of the city is ubiquitous. And where data and information is extracted from the city as a whole.
The human dimension is focused on the knowledge and learning of the people inhabiting the city, and to support the creativity amongst its population, which can be supported by the use of ICT.
The institutional dimensions is how the different institutions and communities in the city make use of ICT technology to change and improve work and life around it, and to become more integrated and collaborative among government, business and residents.
Which makes it clear that technology alone is not what makes a smart city, but is a component of a smart city that must also be used by the entities and residents in the city in a manner that is smart. In addition the the different dimensions of a smart city, there are also different domains of a city that themselves can be made smarter in different ways and might need different solutions built upon the same ideas of smartness. In [85], the different domains of a smart city, that also have their own sub-domains, is gathered from literature on smart cities and is as follows:

- Natural resources and energy
- Transport and mobility
- Buildings
- Living
- Government
- Economy and people

In order to create the infrastructure of the smart city, a variety of different technologies is needed in order to collect data, facilitate communication, process and analyze the data collected among other tasks. The enabling technology for the smart city is identified in [145, 65] and described shortly in the subsections below.

IoT

Internet of Things (IoT) is the everyday objects surrounding us connected to networks, that sense, act on the environment they are in, and interact with other devices, machines and humans over networks[65]. These sensors will act as the part of the nervous system of the city, sensing what is happening in the city, as well as being actuators in the same setting.

Networking

In order to enable the ubiquitousness of the smart city and enabling the use of IoT, networking capabilities to reach the IoT devices and the residents is needed. Different technologies like wireless sensor networks, 5G, IEEE 802.11ba, Software Defined Networking as well as new proposals to addressing schemes is part of enabling the smart city [145].

Big data

Enabling technologies like IoT and cloud computing that can harness vast amount of storage and processing power is set to promote a sharp growth in the amount of data that is produces [28]. In a smart city scenario with an ever growing urban population, that is likely also the case. Big data is the technologies that is designed to deal with the 5 'V's of big data. The volume of data, the velocity of the production of it, variety of the different types, formats and structure of data,, the value of data and lastly the veracity of the data which is how much it can be trusted [36].

Cloud and edge computing

Cloud computing is an enabling technology that allows for on demand computing that scales according to the needs of users. The cloud computing providers have a large capacity of processing power and storage that it relatively cost efficient for the user [48]. This computing capacity and storage capacity is an enabling component in dealing with the large amount of data generated in a smart city. The fog/edge computing is a technology that brings processing, network and storage services typically at the edge of networks [17]. It can help reduce the data traffic to the cloud as well as providing the services nearby with a low latency and location awareness.

As mentioned, it is not completely clear what makes a city smart, which makes it a challenge to correctly create systems that contribute to making the different domains or subsystems of the city smart. There are several challenges to be solved to make sure that changes help reach the goals and avoid negative consequences, like issues with security and privacy when collecting and using data [39].

2.2 Technology management

Within smart cities there is a need to manage the technologies used create the systems around the different domains and areas of the cities. Looking at specifically the ICT systems, we can categorize the management of these system into three categories. The management of the resources used and part of the systems, the management of the data generated, stored and used within the system, and the software running on the systems that is part of managing data and resource, as well as depending on them.

2.2.1 Resource management

Resources is in this case the computational resources, the storage resource and the network and communication resources. The pay model for virtual machines is often based on how much these resources are used [141]. Part of managing the resources controlling who gets to use them under what conditions, as well as making sure they are being utilized optimally. In the smart city domain, the resource can also be IoT devices in the form of sensors the generate data or actuators [120].

2.2.2 Data management

Data management consists of several aspect of managing the data. In [124], a model is proposed that contains all these aspects. The main aspects is the acquisition of data, the processing of data, and the preservation of data. Within these aspects, there are other aspects like data classification, analysis, dissemination, archiving and so on. Every step of related to the handling during its life-cycle is considered part of the data management.

2.2.3 Software management

Running on the different resources and devices, the underlying software is part of the management of both data and the resources themselves. It also is part of providing services and applications to the inhabitants and different entities of the smart city, and is itself dependent on the resources and data to do that.

2.2.4 Schemas - Centralized and Disitributed-to-Centralized

Centralized computing

At the very beginning, computing was centralized. From that point, to the cloud computing, there have been six phases of computing models, identified in [47, 147]. The first phase of computers were large computational units known as mainframes, where the users had to connect to the mainframe through terminals. The second phase was the adoption of personal computers, that held enough computational power to be used alone. The third phase was the network computer, being able to access servers and other computers on a network. The fourth phase, powered by the internet, was the internet computing, which allowed a client-server model [47], where the client could access resources through the internet. The client-server model could also exist with the network computer. The fifth phase was the grid computing, where distributed computation on many computational units within a network cooperated on tasks too large for a single computer can be used to aggregate and share resource. The resource can be specialized devices or computation, storage and data sources, that are all geographically distributed [22]. The grid can be considered a distributed schema, depending on how it is organized, but also be computational resources within a geographical area, similar to cloud computing, and be more similar to cloud computing on those terms. The last phase identified, is the cloud computing phase. Although, these are all important, we are going to focus more on the cloud computing, as it is part of architecture of the distributed-to-centralized solution.

Cloud computing

Cloud computing is a centralized schema, as the computing hardware is localized in certain geographical locations. The characteristics of cloud computing, based on the NIST[1] definition [79], is that it provides on-demand access to resources through self-service, ubiquitous access through the internet, through resource pooling and rapid elasticity there is access to large computational power when needed, and the use of measurements of the

[1] U.S. National Institute of Standards and Technology

resource use for optimization and transparency to the user. Having large computational resources, that are cheap through economies of scale and shared utilization of the equipment [11, p. 507], is large benefit, and needed to handle the big data challenges of the smart city. A drawback with the centralized schema and with the cloud is the latency. The cloud location where the hardware is stationed is often geographically distant from the places where the data is generated, which means that latency is often large, which can be detrimental to some uses of the data which requires low latency and real-time data. It also poses certain security and reliability issues as well [78]. Another issue is that with large amounts of data that is generated at the edges, clouds will have trouble processing the constant flow of data through the networks it is connected to [146, 87].

Distributed-to-centralized computing

In order to deal with these problems and limitations, a new approach in the form of disitributed-to-centralized schema or fog-to-cloud (F2C) computing [77] has been proposed. By having processing and storage near the edges for the network, application with a low latency demand and need for real-time data is supported. It also reduces the processing and data amount stored for the cloud by letting the resources near the edges perform these tasks as well [125]. And allows more processing and energy constrained devices near the edge offload computations to the cloud [104]. The architecture consist of fog nodes near the edges of the network, the provides computation and storage with low latency, with additional layers of nodes close to the cloud with more computational and storage capacity, but higher latency, with the cloud at the top as high latency source of large computational and storage capabilities. A distributed-to-centralized architecture is presented in section 2.10.3, with the ZEN ICT architecture. Some of the drawback identified, is the potential volatility of services and increased service disruption probability due the mobility and power constraints is identified in [104, 78]. Although there might be some issues with security, as there might be limited resources for cryptographic operations, being able to trust the different nodes at the edge, as it is not just one known provider, the devices might be physically available for attacks [7].

2.3 Data management

2.3.1 Centralized data management

In the centralized schema based on the cloud. All data is sent to the large data centers that represents the cloud, where most of the data life-cycle is handled by the cloud, where more or less processed data are sent to it to be processed, stored, shared and used. As mentioned with the cloud, it has large pool of resources, also in the form of storage for data, that can be scaled up according to needs of the user. A single location for all data makes it easier to find, but the geographical distance means that real-time data going through the cloud has significant delay on it.

2.3.2 Distributed-to-centralized data management

In the distributed-to-centralized schema, the data can be stored on several locations, either in the distributed parts of the architecture and/or in the cloud, which gives some challenges in finding data in this schema, which is at the essence of the problem in this thesis. As shown in [125, 126], the different locations, either in the form of fog nodes or the cloud can all be part of the different aspects of the data management, and can gather, process, store and share data. The processing of the data can change the nature of the data, removing errors, redundancies through compression and aggregate it, so that the data found at the different locations might change.

2.4 Resource management

2.4.1 Centralized resource management

In the centralized schema, the computational resources are also centralized. In the case with the cloud, resources in the terms of storage, processors and network and I/O is often managed by a hypervisor for different virtual machines running on the hypervisor. A hypervisor can be considered a operating system for virtual machine instances where each virtual machine can be loaded with its own operating system and applications that are managed by the operating system in the virtual machine. The hypervisor is also responsible for scheduling resources in the same way an operating system does, but for the virtual machines [11, p. 510]. In the context of a smart city, the IoT resources interact directly with the cloud, without any use of nearby computational resources that the Fog would provide.

2.4.2 Distributed-to-centralized resource management

In the distributed-to-centralized scheme, it is a bit more complex to manage the resources. The first issue is the large variety in the devices that can become a potential node providing resources in the fog. The other is the coordination and service allocation of the fog, as the devices that run the fog can in many cases be more volatile, mobile and dynamic than the devices running the cloud, but there might also be more geographical distance between the devices as well. To deal with the first issue, abstractions and operating systems for the devices is proposed in [69, 16]. Cisco, that actually introduced the term ”fog computing”, has introduced ”IOx” as a operating system for resource-rich fog computing devices called Cloudlets [33], and similarly from ParaDrop, with its own Cloudlet architecture [157]. The issue with coordination, means that there is a need for a way to communicate between the nodes as well as an approach on how to orchestrate and control the nodes. In [69], the fog devices are managed by the cloud as the central controller, where the communication is done over a Software Defined Network (SDN), and the cloud manage what applications run on the fog server and what data is kept there. In [16], a service orchestration layer, laying on top of the abstraction of the underlying device, that is responsible for finding the appropriate service for service requests, set system wide policies for the devices and provide communication between the devices. There is also work done looking at service allocation strategies, such as in [132], that could be used by orchestration and coordination systems for the fog, and work on resource management specifically for F2C in [119], with

a focus on resource and service request categorization as part of the service allocation strategy.

2.5 Software management

2.5.1 Centralized software management

In the centralized schema, the software can be run as a single instance in a single place. Which means that the need for coordination or communication between different instances can be eliminated. But even in a centralized scheme, scaling horizontally, by adding more instances of the running software can be a way to scale up the capacity of the system, if the architecture allows for it, which can require communication between the running instances. In the context of the cloud, the software is run as instances of virtual machines, or in containers, that provides a stack of software platforms the software can run on when given processing time by the hypervisor that manage the access to computing resources.

2.5.2 Distributed-to-centralized software management

With a distributed-to-centralized schema, there are now several locations where the software can run. Depending on the needs, it might need to run several different places like on the cloud and on different fog devices, where the software might perform different functions, which means that there is again a challenge of coordinating and orchestrating the different places. For distributed-to-centralized systems, [138] identifies two hybrid approaches, that are more fitting with the distributed-to-centralized schema. The first is the Edge-server systems, that have centralized core, but servers placed at the edge of network near the users, in the given example to serve content, but in the case with the smart city, this is reversed as the edges now also collect data and send it to the centralized core. And the collaborative distributed system, that starts as a centralized client-server system, but grows distributed once users join. It is also dependent on the system how much communication and coordination is needed, and of which type of communication is needed. In some cases, the distributed software might need to communicate with the core only, and in other cases, with other distributed software instances, which raises the question if this should be done indirectly through the core, or between distributed instances.

2.6 Cost models

As mentioned in the introduction, there are many different applications and services in the smart city. Selecting among services and data is an important issue in this context, as services are often dependent on data and other services in order to provide their services. To be able to select, appraising the quality of the different services and data is important. Quality is defined as The result of judgment of the perceived composition of an entity with respect to its desired composition [62], in other words, how close it is to the ideal instance of it. The Quality of Service (QoS), based on its definition for telecommunications from ITU-U [106], is defined as all the characteristics of a service that impacts how it is able satisfy the stated and implied needs of the use of the service. These characteristics are

quantitative measurements of the performance of the service. The Quality of Experience (QoE), is based experience of the users, and how they appraise the quality. This can be measured by having users provide feedback, or objectively through models that maps QoS to expected QoE [43]. This gives use two perspectives to take into consideration when we attempt to appraise the quality of a service. Data also has its own quality measurements, which will be expanded upon in section 2.7.3. A characteristic that are deemed important for the service and data, will be referred to as a Key Performance Indicators (KPI) in this thesis. A KPI is a quantifiable metric the reflect the performance of the service or data in relation to the goals and objectives of it. This will further expanded upon in section 2.7, that goes more in the depth of specific KPIs.

As both services and data can have several KPIs, makes the problem of selecting among these a multi-criteria decision problem if the selection is dependent on more than one KPI. A criterion is in this case a KPI and how it is judged, like judging the prize more positively the cheaper it is. If you have a single criterion when selecting, the selection is trivial in many cases. Take as an example deciding to buy a car, if the only criterion is prize, then you pick the cheapest one. If you introduce two criteria, like prize and comfort, it is suddenly not that easy. As there exists no value relation between the two criteria, how much more you are willing to pay for more comfort is highly individual. The difficulty arises because these problems give rise to a set of trade-off optimal solutions (also known as Pareto-optimal solutions), instead of a single optimal or a set of equally optimal solutions [34]. A trade-off optimal solution is a solution that is not dominated by another alternative. Being dominated means that there exists an alternative that is better on every criterion. Figure 2.1 shows an example of trade-off optimal solutions, and a single dominated alternative d, where d has a higher cost and lower comfort than solution A and B.

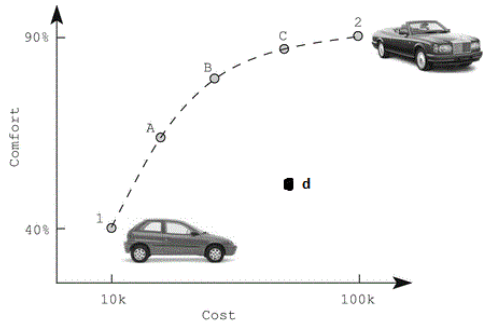


Figure 2.1: Five trade-off optimal solutions for a car, and one dominated alternative. Taken from [34]

There are two main approaches to selecting among these alternatives. The first involves finding all the trade-off optimal solutions, and then selecting one in the set of trade-off optimal solution. This is an approach fit for human decision makers that use their intuition and knowledge to chose among a smaller set of alternatives. The other approach is called scalarization [34], which used a method or approach to convert the multi-criteria decision

problem into a single-criteria decision problem, which is easy to solve and can easily be automated. This approach or method is what is referred to as the cost model in this thesis, that takes in a set of KPI measurement for the alternatives and produces a ranking of the alternatives. That ranking can then by used to select from, by picking the best ranked alternative. Figure 2.2 shows an example of a cost model used for three alternative cars with cost and comfort KPIs. After ranking and scoring the alternatives with a cost model, the highest scoring one is chosen.

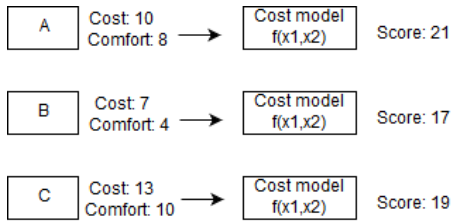


Figure 2.2: Three alternatives with their KPIs used by the cost model to calculate their ranking

As there is a large amount of selection and cost model approaches, looking specifically at the approaches used to select cloud service providers has been used to select most of the approaches to look into further in this thesis. A fair bit of the selection and cost model approaches is written about in [136], that does a literature review of cloud service provider selection, but some outside of that has also been chosen. The goal has in this case been to try to look at instances within different categories of approaches. The adaptive approaches presented includes a selection method in addition to the cost model.

Some work has also been done on selection and service allocation within distributed-to-centralized systems, mostly looking at fog-to-cloud. Some of the approaches, like in [132] improve load balance for a large set of services, while the problem in this thesis is more narrow than that, as it is about how to find the best service or data repository for a user. As part of a service allocation approach in [131], First Fit is proposed as a strategy where a service task is allocated the first place available in the Fog, and sent to the cloud if no resource is available.. A similar strategy is presented [4], where a task is attempted to first be allocated to the nearest fog, then to other connected fogs, and the to the cloud. It also looks at response time and delay. [131] also presents a Best-Fit approach that offloads tasks to the cloud if no resource with a given capacity threshold that is larger than the task is available, and a Best-Fit with queue that will wait if it is faster to wait for a fog resource instead of sending the task to the cloud.

Work that is a bit closer to the cost model and selection, within the domain of distributed-to-centralized, or fog-to-cloud is found in [133], that uses Integer Linear Programming, which can be considered a subset of linear programming discussed in section 2.6.1, to attempt the lower the average delay when processing tasks. In [161], a mobile computing offloading scheme is proposed, that uses TOPSIS to select an interface to offload computation, which will be discussed in section 2.6.6.

2.6.1 Linear programming models

As an example of a linear programming model, an example from [129] is used, as it is used as an approach to select a cloud provider, called the Simple Cloud Provider Selection (SCPS) algorithm. The SCPS algorithm is a linear programming model meant to help choose a cloud computing provider based on the user needs and the performance of the cloud computing providers. The score is calculated with a set of weights provided by the user to give preference to some KPIs over others, and the score in the different KPIs used to evaluate the cloud computing provider. The score of an alternative can be calculated by formula 2.1, where x_i is some sort of performance measurement for KPI i, and w_i is the weight of importance set by the user for that KPI, this is summed up for each KPI, which then gives the score. The cloud provider with the highest score is selected.

$$score = \sum x_i * w_i \tag{2.1}$$

In SCPS, the performance measurement or x_i is calculated by the average measurement and its standard deviation, and either divided by or by dividing the norm of all measurement with it. The norm is the square root of all measurements for a KPI squared and summed up. An example, where this is done for a KPI one would like to maximized and one that should be minimized is shown in 2.2 and 2.3.

$$latency\_score_i = \frac{||latency||_\alpha}{\overline{latency_i} + std(latency)} \tag{2.2}$$

$$bandwidth\_score_i = \frac{\overline{bandwidth_i} + std(bandwidth)}{||bandwidth||_\alpha} \tag{2.3}$$

The SCPS algorithm is a simple solution. In terms of complexity, you only need to iterate through the measurements of all the cloud providers two times to find the highest scoring one. The first time to calculate the norms, the second time to calculate the score and find the highest scoring solution. With the algorithm itself taking input in the form of a set of performance measures where some of them are highly dynamic, especially those related networking performance, it means that a choice taken by the algorithm might be right when it is taken, but not after some time has passed, as it is not an adaptive algorithm. Even if the algorithm were called on regular intervals, there is quite a lot of overhead gathering all those measurements each time if it is not needed. As shown in [43], the relationship between the QoS and QoE is not necessarily of linear character. If the customer has a requirement with latency below a given value, but as long as it is below it, it does not matter by how much, which might be hard to represent in the model in certain scenarios. Setting the correct weights can affect which solution is selected, with only four dimensions, setting those is simple, but with more added dimensions, it might be hard to set the correct weights. It also allows for scenarios where cloud computing providers are chosen with unacceptable values in some of the dimension with low weights, as strict constraints are not part of the model. Since the measurements of performance for an alternative is specified in terms it performance in relation to all other alternatives performance, it is not good to attempt to specify specific wanted KPI performance.

2.6.2 Epsilon-Constraint method

The ε-Constraint approach is a scalarization approach to a multi-objective criteria problem [34]. A scalarization approach to multi-objective optimization problems is an approach that cast a multi-criteria optimization problem to a single-criteria optimization problem. The approach consist of setting constraints on all KPIs measurements, except one KPI. This filters the total solutions into the ones that are acceptable. After that, the solutions are ranked after the one KPI without the constraint, choosing the one with the highest or lowest value depending on if the goal is to minimize or maximize that objective.

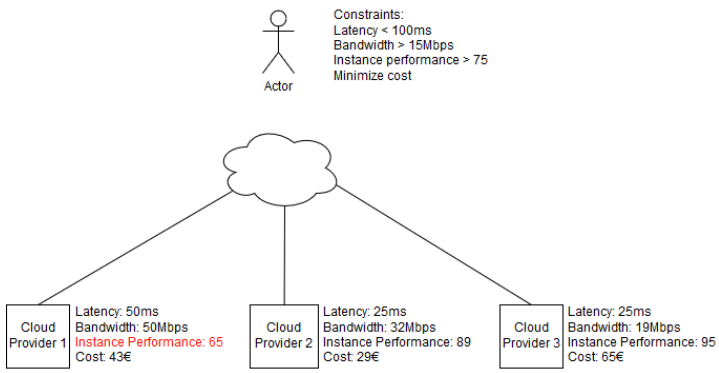


Figure 2.3: Example scenario using the ε-Constraint method

An example scenario is shown in figure 2.3, where ε-Constraint approach is used, with three specified constraints, and a goal of minimizing cost. Cloud provider 1 get filtered away by the constraint indicated by the red text on instance performance, and Cloud provider 2 gets selected because it has the lowest price.
This approach is very simple, with a low computational complexity. You only need to iterate through the measurements for all cloud computing providers once, to find the best alternative. It is not an adaptive approach, so any change in performance can mean that the chosen alternative at time t is not the best alternative at time t+1. The constraints can potentially be too strict, and give no solutions, and the solutions found can depend on this. It is not good for specifying more complex needs, and might exclude more overall good alternatives with its focus on one KPI. The same issues with measurement and utility not being in a linear relationship, might be true for the selected KPI to minimize or maximize.

2.6.3 Analytical Hierarchy Process (AHP)

AHP, which was developed by Saaty in the 1970s [116], is another approach to multiple-objective decision problems that solves the problem by casting it as a single-objective decision problem before solving it. The method presented here to solve a multi-objective decision problem is based same method used for cloud provider selection used in [51]. The process of solving a multi-criteria decision problem is done in several steps, where

the first set of steps only need to be done once, and the rest of the steps that needs to be redone for different measurements. The first step is to build a hierarchy of attributes, where the top attribute is the overall performance, and the attributes below is sub attributes of the top attribute or other sub attributes. This ends up creating a hierarchy tree with more specific attributes towards the leaves, until they are specific enough that they can be measured and given a score. The next step to assign relative weights to all sub attributes such that they all add up to 1 for a child of a node in the hierarchy, this can be done by pair-wise comparisons of the attributes assigning relative importance between them, or they can be assigned by the user by another method like cumulative voting, where the user assigns relative importance based limited set of units that must divided according to their importance [15]. The following steps is the calculation of a score using the hierarchy tree and its weights together with the different measurement.

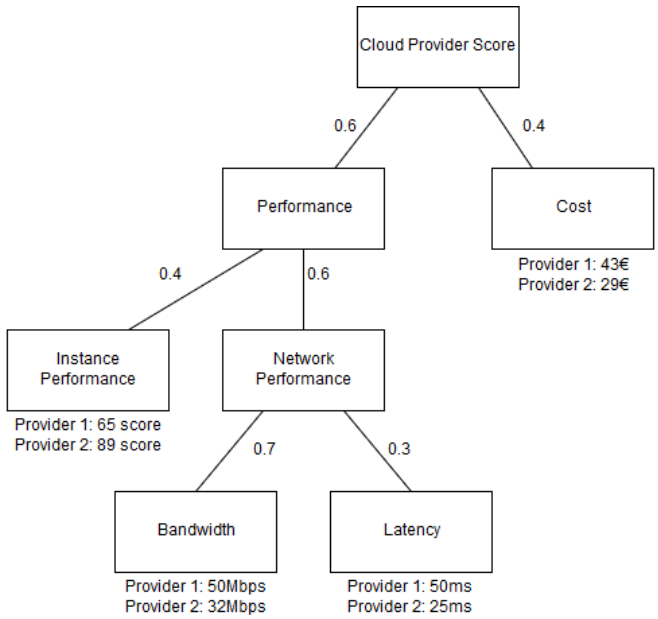


Figure 2.4: Example scenario with an AHP comparing two providers

Figure 2.4 shows an instance of a AHP hierarchy tree with weights and the measurements of two providers that is being compared. The first step is to calculate a relative score at the leaf nodes, this can be done the using the norm, the same way as in 2.2 and 2.3, or some other way to get a relative score. This gives each alternative's KPI a score relative to the other alternatives. This score is then multiplied with its weight and aggregated in the parent node. This is down from the leafs up to the root, where each alternative gets a final score based in their performance.

This is not an adaptive approach to a multiple-criteria selection problem. It is a bit more complex in terms of calculation compared to the linear programming model approach, but

it allows more complexity in stating the preferential relationships between the attributes. In computational complexity, it needs to iterate through the measurements of the cloud providers twice, once to calculate the norm of the measurements, and the second run to calculate the score. It can also be used to create the weights to be used in the linear programming model, instead of using the hierarchy for calculating the score. In that case, with figure 2.4, instance performance weight would be the weight of performance multiplied with its own weight, $0.6 * 0.4 = 0.24$, latency weight would be $0.6 * 0.6 * 0.3 = 0.108$, bandwidth weight would be $0.6 * 0.6 * 0.7 = 0.252$, and cost weight would just be 0.4. The example in figure 2.4 used the measurements to create relative performance scores for the providers, but you can also use similar approaches, such as utility functions represent preferences that are even more complex. It does not include to ability to specify hard criteria to filter out unacceptable values from being considered as alternatives.

2.6.4 Decision Envelopment Analysis (DEA)

Decision Envelopment Analysis (DEA) [18] is a linear programming-based technique used in decision making. It is used to measure the performance of an alternative with multiple inputs and outputs. In a simple case of service selection, the cost is the only input, but for a more nuanced evaluation of much benefit is given for the cost, other factors can also be used as input that is not converted into monetary cost. It uses the term "Efficiency" as its measurement of the performance of the alternatives, where the Efficiency is the result of the weighted sum of outputs divided by the weighted sum of the inputs, described in formula 2.4. Where $weight_i$ is the importance of $output_i$, and $weight_j$ the importance of $input_j$.

$$Efficiency = \frac{\sum output_i * weight_i}{\sum input_j * weight_j} \tag{2.4}$$

It is itself a version of a linear programming model, similar to the one shown in 2.6.1 and has the same drawback and limitations in that it is not adaptive to changes on its own, it can not specify hard constraints to filter alternatives, it does not directly deal with a not linear relationship between a performance measurement and utility. But it does provide a model that has a focus on get the most value out of the alternative in relation to cost, and provides a more nuanced way to specify cost than just a simple linear programming model shown in 2.6.1. Another issue is how fractions act as the denominator goes towards zero, where it goes to infinity as it get closer to zero if the numerator is a constant value. This is also a problem for alternatives with no cost, which would not work in this cost model.

2.6.5 Goal programming

Goal programming consists of several approaches for selection among multi-criteria alternatives using goals set for the different KPI. The goals can be soft, where a solution might be have a lower score than the goal of that KPI, or a hard goal which is a constraint that must be held. In [64], which is a visual and interactive approach to goal programming, it selects all non-dominated alternatives that meets the set goals. Or in a case where no alternatives meet the goals, the ones that are closest to the goal. Which can then be used

decision makers. It can also be used in a manner similar to the linear programming approach, with objective functions that calculate the score of each alternative in relation with the set goals. And simple example of goal programming is shown in [109], where it uses a weighted goal programming model similar to the on in 2.6.1, illustrated in formula 2.5.

$$score = \sum(\alpha_i n_i + \beta_i p_i) \tag{2.5}$$

For an alternative, n_i is the negative deviation for the goal of i, with α_i as the weight assigned to it. Similarly, p_i is the positive deviation for the goal of i, with β_i as the weight assigned to it, where the best alternative is the alternative with the lowest score, which would be the alternative closest to the goal. In many cases, it is positive have a lower or a higher a KPI that is being measured than the set goals, which is what is done in [142]. It proposes a formula that negatively impacts not reaching the goal, while not affecting the score one reaching past the goal, with its Exponential Weighted Difference, shown in 2.6, where x_i is the measurement for a KPI i, g_i the goal i, and w_i the importance weight for i.

$$score \sum e^{-(x_i - g_i)w_i} \tag{2.6}$$

This method is good for ensuring reaching clearly defined goals, where getting better performance on some KPIs past that is not important. But for minimizing or maximizing the value of KPIs, it is a bad fit. It is also dependent on the goals, as it treats solutions that perform far better than the stated goals equal or even worse, depending on the approach. If the goal is above all alternatives, it will likely rank the alternatives appropriately and rank the closest one in performance as the best alternative, so setting high goals seems important. It is very simple in runtime, need to only iterate through all alternatives once to find the result. It is not adaptive either.

2.6.6 TOPSIS

Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) was first presented in [60]. It attempts to select as solution that is the closest to the optimal solution and the furthest away from least optimal one, that is a solution that would be a collection of all the best scoring KPIs, and a solution that is the collection of all the worst scoring KPIs, from the collection of alternatives. The basis for the explanation of how it works is from [13]. The first step is to normalize all the values for all the alternatives, which is done by dividing the values in each column with the square root of the sum of all values raised to a power of two, as shown in formula 2.7.

$$R_{ij} = \frac{X_{ij}}{\sqrt{\sum_{j=1} X_{ij}^2}} \tag{2.7}$$

The second step is to add weights to the different KPIs in the different columns, according to their importance. In [161], the weights are calculated using AHP, with pairwise comparisons, but another approach, like cumulative voting [15] could also be used. Formula 2.8 shows how this is calculated.

$$V_{ij} = w_j r_{ij} \tag{2.8}$$

The third step is to calculate the positive ideal and the negative ideal solutions, by for all alternative's KPIs, find the best and the worst scoring ones from each category of KPIs, and use them to create the ideal solutions.
The fourth step is for each alternative, to calculate the Euclidean distance from the ideal positive and the ideal negative solutions. This is done with formula 2.9 for the positive ideal, where v_j^+ is the positive ideal value of KPI in column j, with the same approach to the negative distance, shown in formula 2.11.

$$S_i^+ = \sqrt{\sum (v_j^+ - v_{ij})^2} \tag{2.9}$$

$$S_i^- = \sqrt{\sum (v_j^- - v_{ij})^2} \tag{2.10}$$

The final step is to calculate the relative closeness to the optimal solution, where the alternative C_i that is closest to 1 is the best solution, calculated with the following formula:

$$C_i = \frac{S_i^-}{S_i^- + S_i^+} \tag{2.11}$$

The method is fairly simple, as the only things that needs to be specified by the user, is a set of weights for each of the KPIs of the different alternatives. Computational, it needs one run through all the alternatives to calculate the norm. Then it must run through the alternatives again to normalize and add weights to the value and find the positive and negative ideal. A last iteration of the alternatives is needed to calculate the final score. The score is in this case based on all the alternatives, through creating the ideals based on them. It is not an adaptive approach either.

2.6.7 Multi-Attribute Utility Theory (MAUT)

The MAUT approach is an approach to solving multi-attribute/criteria decision or optimization problems using utility functions to rate the different KPIs [148]. It works almost the same way as the linear programming approach in the SCPS algorithm, but it differs in how it calculates the scores of each attribute. It includes two extra steps, the first being the identification of the values of the KPIs themselves, and the second being the assigning of utility values to the KPI values identified. After that, like in the linear programming approach, preference weights are set, and the score for each solution is calculated by multiplying the utility value of each objective in the solution with its weight, and summing up the results for each objective. An example of the MAUT function is shown in formula 2.12, where each KPI and its measurement it passed through an utility function $u_i(x)$ and multiplied with its weight w_i. An example of a user defined utility function is shown in formula 2.15, which maps latency measurements to utility values. The utility functions could be any type of function that takes in a measurement and returns a utility score between 0 and 100, where 100 is the optimal utility.

$$score = \sum_{i=1} u_i(x) * w_i \tag{2.12}$$

$$u_{\text{latency}}(x) = \begin{cases} 100 & x \leq 25ms \\ 75 & 26ms \leq x \leq 50ms \\ 50 & 51ms \leq x \leq 75ms \\ 25 & 76ms \leq x \leq 100ms \\ 0 & 101ms \leq x \end{cases} \tag{2.13}$$

The method is computationally simple, only requiring a single iteration through all measurements of the cloud providers to find the best one, giving it O(n) run-time complexity for n cloud computing providers. It is also not an adaptive approach, and will stick with the first chosen alternative. It also requires more work from the customer for the three first steps identifying the intervals, setting the values for them, and adding the weights. This however, allows for more complexity in the rating of the different measurements of the dimensions, and can help deal with the issues mentioned in [43], that the relationship between QoS and QoE is not necessarily of linear character. In addition to helping represent nonlinear relationship between utility and measurements, it allows the representation of nominal scales with values as well. As an example, a cloud computing provider can be perhaps be given a score based on a nominal scale security features with this method. With many different objectives, setting the correct weight numbers gets less intuitive and harder to get right. It does not include hard constraints, the worst a badly performing KPI can do, is give a 0 in utility score. The proposed version of a utility function, has the downside that it can rank an alternative that is not trade-off optimal equally to a trade-off optimal one, if the score is within all the same utility intervals, but slightly worse. A solution to this can be to use utility functions based on mathematical functions like in [54]. Another approach, could be to instead specify utility points, mapping some KPI measurements to some utility values, and calculate the utility of any measurement between these points by using the gradient between them to decide the value.

2.6.8 ELECTRE methods

ELECTRE consists of a series of outranking methods for multi-criteria problems [45]. Within ELECTRE, there are several versions created with some differences between them. It consists of several steps to create a ranking of the alternatives. As an example of how the method work, ELECTRE III will be used, based on [20]. In outranking methods, binary relationships of preference or indifference between solution, ELECTRE adds more nuance by adding the concepts of strong preference, weak preference and indifference in the ranking, based on user-defined thresholds. How this works is demonstrated beneath. Let matrix x in 2.14 be three alternatives in each row with the score for the different KPIs in the columns, which we for simplicity would want to maximize.

$$\begin{bmatrix} 50 & 10 & 70 \\ 25 & 50 & 90 \\ 52 & 4 & 15 \end{bmatrix} \tag{2.14}$$

The first step for the user would be for each of the KPIs in the columns, to decide their weights in terms of importance, an indifference threshold (q) and a preference threshold

(p). The indifference threshold described how much worse a KPI can score and still be equal in the eyes of the user, and the preference threshold describes when the KPI is ranked worse, leaving a gap between indifference and preference where there is some nuance to whether its worse. This is shown in the table:

	KPI 1	KPI 2	KPI 3
Weight	5	3	2
Indifference (q)	15	5	15
Preference (p)	30	10	30

Using this table and the matrix, we can demonstrate how the calculation for the concordance matrix is done. As an example, row 1 compared with row 2 in matrix 2.14, to see if alternative 1 at least as good alternative 2. For each KPI, compare with function C in formula 2.15, where a is the KPI, and X_i and X_j are the two alternatives. We will continue to use the comparison of alternative 1 against alternative 2, as examples to show how math is done, as well as calculating the matrices for all comparisons.

$$C_a(x_i, x_j) = \begin{cases} 1, & x_i + q_a \geq x_j \\ 0, & x_i + p_a \leq x_j \\ \frac{p + x_i - x_j}{p_a - q_a}, & x_i + q_a < x_j < x_i + p_a \end{cases} \tag{2.15}$$

- For KPI 1, this would give $C_1(X_1, X_2) = 1$ since $50 + 15 \geq 25$
- For KPI 2, this would give $C_2(X_1, X_2) = 0$ since $10 + 10 \leq 50$
- For KPI 3, this would give $C_3(X_1, X_2) = \frac{30+70-90}{30-15} = 0.667$ since $70+15 < 90 < 70 + 30$

This would all be calculated together for each KPI and its weight to calculate the degree which an alternative is equally good or better, shown by the formula in 2.16, and being calculated for alternative 1 at least as good as alternative 2, shown below it.

$$C(x_i, x_j) = \frac{\sum C_a(x_i, x_j) * w_a}{\sum w_a} \tag{2.16}$$

$$C(x_1, x_2) = \frac{1 * 5 + 0 * 3 + 0.667 * 2}{5 + 3 + 2} = 0.633$$

This creates a concordance matrix of pairwise comparisons that contains the degree which an alternative is at least as good as the other, expressed as number between and including 0 and 1, shown in matrix 2.17. The diagonal of numbers in the matrix going down from the top left to bottom right is comparisons against itself, which always is 1.

$$\begin{bmatrix} 1 & 0.633 & 1 \\ 0.667 & 1 & 0.6 \\ 0.74 & 0.5 & 1 \end{bmatrix} \tag{2.17}$$

The next step is to try to disprove this by creating a discordance matrix, that attempts to disprove the assertion of the concordance matrix. To get this, we need the user to specify a new set of values, a veto threshold for individual KPI measurements that have a unacceptably large difference between them in a negative way, shown below.

Veto threshold (v)	80	30	90

A very similar function to formula 2.15 is used, but now we use the veto threshold to see if we can disconfirm what we found in the concordance matrix. That formula is shown in formula 2.18.

$$D_a(x_i, x_j) = \begin{cases} 0, & x_i + p_a \geq x_j \\ 1, & x_i + v_a \leq x_j \\ \frac{x_j - x_i - p_a}{v_a - p_a}, & x_i + p_a < x_j < x_i + v_a \end{cases} \tag{2.18}$$

- For KPI 1, this would give $D_1(X_1, X_2) = 0$ since $50 + 30 \geq 25$
- For KPI 2, this would give $D_2(X_1, X_2) = 1$ since $10 + 30 \leq 50$
- For KPI 3, this would give $D_3(X_1, X_2) = 0$ since $70 + 30 \geq 90$

These values are going to be used together with the concordance matrix to calculate a credibility matrix, that assert that an alternative is at least as good as another one. If no discordance value D_a is higher than its concordance value C, then the aggregated concordance value C is the credibility value, otherwise it affects the credibility value, as shown in the other option in formula 2.19. The ones that are aggregated, is the set of those discordance values that are higher than the concordance value, is called $J(x_i, x_j)$ where all members of the set are $D_a(x_i.x_j) > C(x_i, x_j)$.

$$S_a(x_i, x_j) = \begin{cases} C(x_i, x_j), & C(x_i, x_j) \geq D_a(x_i, x_j) \forall a \\ C(x_i, x_j) * \prod_{j \in J(x_i, x_j)} \frac{1 - D_a(x_i, x_j)}{1 - C_a(x_i, x_j)} \end{cases} \tag{2.19}$$

This finally creates the credibility matrix shown in 2.20. The only changed entries, is $S(x_1, x_2)$, which had a discordance value of 1 for KPI 2, and $S(x_3, x_2)$ which had a discordance value of 1 for KPI 2, which both give a credibility value of 0 after formula 2.19 is used.

$$\begin{bmatrix} 1 & 0 & 1 \\ 0.667 & 1 & 0.6 \\ 0.74 & 0 & 1 \end{bmatrix} \tag{2.20}$$

With this matrix, a ranking of the alternatives is made through a process called descending and ascending distillation that, that works as follow. Defined a credibility value $s(\lambda)$ that defines a cutoff of the values considered relative to the highest credibility matrix value λ, so only values above $\lambda - s(\lambda)$ is considered as outranked.
Then for each entry in the credibility matrix, calculate how many alternatives it outranks minus the amount of projects that outranks it. All the alternatives with the highest value is ranked first, if it is a shared set, that calculation is done again, but only with the values inside the set of the highest, repeat until it is the only alternative is left, which is the highest rank in the descending distillation, this is repeated for other entries until a descending ranking is made. The same is the done in ascending order, and the ranking are combined to make a final rating, where if an alternative is ranked above another in both orders, it remains that way in the final result.

The first thing to notice, is the amount of information that is needed from the user to make this ranking. It is one of the more complex models in terms of the different needed user input and how to inform the user on how to use it. The upside with that complexity is that it allows the user specify more clearly what they want, unlike a simple model where only a few options are available to specify the needs, which might be unclear how the affect the selection. The approach is computationally demanding as it involves pairwise comparisons, and since the each pair is compared twice, it gives us a complexity of at least $O(n^2)$ comparisons for n alternatives, with multiple steps of computation, increasing depending on the amount of KPIs, and after that, there is that final ranking process with its own computation. In [41], the amount computations is specified by $O(n^2 * m)$ where n is the alternatives, and m the KPIs. It is not an adaptive approach either. The ranking is anchored in the relative performance of the alternatives, and not goals or utility. It also doesn't give a score to the alternatives, but ranks the in a order that can be used to select from, by selecting the highest ranked alternative.

2.6.9 Adaptive Cloud Provider Selection (ACPS)

The Adaptive Cloud Provider Selection algorithm is an algorithm based on the Page Hinkley approach [118]. The Page Hinkley approach is a sequential analysis technique. It is used in [129] together with the SCPS algorithm presented in 2.6.1, to get create an adaptive algorithm that change cloud provider on degrading performance of the cloud provider. Specifically, it is used to trigger a reselection of cloud computing providers using SCPS, when the network performance of the currently selection degrades under a given threshold. As mentioned in the portion about SCPS, a problem with it is that it only makes a single selection of the cloud provider, and stays with that choice regardless of the degradation of performance by that cloud provider. ACPS algorithm solves this issue using continuous measures of the currently selected cloud computing provider score, using the same dimensions as SCPS: Latency, bandwidth, instance performance and cost. It works in following way: It continuously measures and calculates the current score currently selected cloud computing provider using its dimensions of measurement. It uses this to calculate the average of all measurements it has done from it started. With this is average, it calculates a cumulative variable of the difference between the average and the current score. The cumulative variable is then tested if it is over a given threshold value. If it is, it triggers a new selection procedure with SCPS. An example of the Page Hinkley approach implemented

as an algorithm is shown algorithm 1, taken from [129].

```
σ: allowed change value;
count = 0;
while true do
    mean = (mean + measurement) * (count/(count + 1));
    cumulative_difference = cumulative_difference + mean - measurement;
    smallest_cumulative_difference;
    if cumulative_difference < smallest_cumulative_difference then
        smallest_cumulative_difference = cumulative_difference;
    end
    delta = cumulative_difference - smallest_cumulative_difference;
    if delta ≥ σ then
        ALARM();
    end
    count++;
end
```

Algorithm 1: The ACPS approach based on Page Hinkley approach

ACPS allows for an adaptive approach to the cloud computing provider selection problems. It itself only calculates the currently selected alternative's performance, and is dependent on another cost model to do the reselection. It is also simple in terms of computational complexity, as it only uses measures the currently selected cloud computing providers. However, as it does that, it cannot detect if another cloud computing provider alternative suddenly increases its overall performance so much that it would be a better choice. There is also the issue of balancing the threshold where the algorithm triggers a new selection procedure with SCPS. Too low, and it might trigger too many changes as the performance swings, and too high, it might be stuck on worse solution. Even in cases where there is a close alternative in terms of performance, a too large threshold will make it keep a subpar alternative, unlike what Q-learning and Boltzmann would do, presented in the next sections.

2.6.10 Q-learning

Q-learning is a model-free reinforcement learning technique that can used to find an optimal solution in Markovian domains, where the goal is to find the best action to take in the current state [152]. A Markovian domain allows us to represent the selection problem as a series of states, state transitions and probabilities, where the cost model is the expected utility for each state transition. Finding the best action to take is done through observing the value of the current state after taking an action to get to the state. After trying all actions in all states several times, an optimal solution will eventually be found. In [130], Q-learning is used as an adaptive approach to the problem of choosing the best network connection when there are several network connections available. A problem can be solved with Q-learning if it can be transformed into a set of states and actions in those states, where the actions can be given expected utility values through observing the outcome of choosing an action.

Figure 2.5 shows the formula for updating the utility of taking an action. To calculate

$$Q_{t+1}(s_t, a_t) = \underbrace{Q_t(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t, a_t)}_{\text{learning rate}} \times \Big[\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t, a_t)}_{\text{old value}} \Big]$$

Figure 2.5: The formula for expected utility in Q-learning [130]

the utility of taking an action, it uses the old value of it and adds it together with the reward observed for taking the action, as well the difference between the old value and the estimate of the optimal future value, which is the highest expected utility function in the new state.

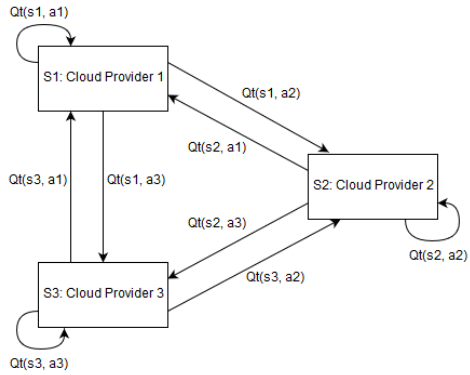


Figure 2.6: A cloud provider selection scenario with three alternatives

In Figure 2.6 the cloud computing provider scenario used earlier is transformed into a Q-learning problem. As an example of how this could work, assume the agent is in S1, and this is the currently selected cloud provider, and the highest expected utility action is Qt(s1, a1). The agent takes that choice, as this has the highest expected utility. It now enters the same state and observes a reward. In this case, the reward can be score of the cloud computing provider in this state in t+1, calculated in the same manner as the SCPS algorithm calculates the score, minus the score cloud computing provider in the former state in t. Assume that the score of the provider has degraded, action Qt(s1, a1) will now have lower expected utility, and one of the other actions might have higher utility in S1, leading to a change of state and cloud computing provider.

This is an adaptive approach that will change solution if the currently selected one degrades in performance. However, if other alternatives suddenly increases its performance, it will not be detected until an action selection those alternatives is selected. Each calculation in itself is fairly simple in terms of complexity, and once that optimal solution is found, it is a question of recalculating this action or actions with some interval. It is required to at least store all the measured performance of all alternatives, unlike the ACPS algorithm. Another issue is however is that it requires a period of trying other different actions to

find this optimal solution. As this was used in dynamic environment, where the network performance, or other changing performance measures affect the decision, you also have to set a learning rate to allows change to another action quick enough, but not too quick, to allow it to retain some memory of the past utility values.

2.6.11 Boltzmann Exploration

In paper [54], an adaptive approach to service selection is proposed. Boltzmann exploration is another reinforcement learning technique, that similarly to Q-learning, explained in 2.6.10, use exploration of different states in order to discover the best state. Similarly to, it uses utility functions to select the states, but unlike in Q-learning, is also an annealing algorithm [72]. Which means that it has a temperature parameter, that depending value makes it more or less likely to choose an option with less expected utility. The proposed solution uses mathematically defined expected utility functions where the input is the KPIs, instead of the intervals presented in MAUT in 2.6.7, as the cost model used to make the selection.

$$Pr(P_j|EU_i) = \frac{e^{T*EU_i(P_j)}}{\sum_{p_a \in P} e^{T*EU_i(p_a)}} \tag{2.21}$$

The learning policy is shown in formula 2.21, is the chance of selecting provide P_j given the expected utility function of the user EU_i. The numerator of the fraction is the exponential of the temperature variable T times the expected utility of a provider while the denominator is the sum of all exponentials of the different expected utilities for all providers. As the temperature variable T gets a lower value, the higher the chance of picking the alternative with the highest expected value gets. When it is warm, it will change between the state, exploring the different performances, and over time as the temperature gets colder, choose the better alternatives.
Like the Q-learning, it needs to keep information about all the different alternatives for it to work, and might only start changing state once the current selection degrades as the temperature gets lower. It is however possible to keep the temperature such that it does exploration from with a small chances, or to perhaps reheat the temperature as a response to degrading performance. The behavior is tied to how temperature is adjusted, and there is the chance that in a dynamic environment, it might get stuck in a worse state if not allowed to continuously explore. If the temperature is kept high, it might keep exploring worse alternatives even if there exists a stable, best alternative. Finding an appropriate temperature policy can be a challenge.

2.6.12 Cost model comparison

The different cost models and selection methods each have strength and weaknesses, and the best one depends on these properties and the needs of the user. The properties that are used to compare the different solutions are as follows:

- Computational complexity: Specified in relation to the number of alternatives to rank n. For the adaptive ones, that will be running with regularity, we look at space complexity.

Approach	Complexity	Input	Comparison	Adaptive	Strict
Linear programming	O(n)	Set of weights	Relative	No	No
Epslion-constraint	O(n)	Constraints and attribute to optimize	Absolute value	No	Yes
AHP	O(n)	Hierarchy with weights	Relative	No	No
DEA	O(n)	Set of weights	Absolute value	No	No
Goal programming	O(n)	A goal and weights	Proximity to the goal	No	No
TOPSIS	O(n)	Set of weights	Relative	No	No
MAUT	O(n)	Utility functions and weights	Utility	No	No
ELECTRE	O(n^2)	Weights, indifference, preference, veto threshold and crediblity value	Absolute value	No	No
ACPS	s: O(1)	Reselection threshold and reselection algorithm	Absolute value	Yes	No
Q-learning	s: O(n)	Utility functions	Utility	Yes	No
Boltzmann	s: O(n)	Utility functions	Utility	Yes	No

Table 2.1: Comparison of the different cost model approaches

- User input: What type of input is needed
- Comparison: On what basis does the different models score different alternative. The relative difference between the alternatives, by using utility functions, in relation to goals or absolute value
- Adaptivity: Is the model able to adapt to changes in the performances?
- Strict constraints: Is it able to represent strict constraints?

The different cost model approaches that is presented in this section, is compared by the stated criteria in the table 2.1. These are just some of different properties of the selection methods. In many of the cases, the comparison methods can be changed to use utility functions, or other types of comparisons, and the currently represented is the once used for the methods in their description. The other aspects of the different approaches is discussed below them and could not as easily be summed up to fit in comparison table.

2.7 Key Performance Indicator (KPI)

In [12], KPI is given the following definition:
”KPIs are quantifiable metrics which reflect the performance of an organization in achieving its goals and objectives. KPIs reflect strategic value drivers rather than just measuring non-critical business activities and processes.”
But in this case, we are looking specifically at the services and the data provided by them, that reflect that value given to the user of the services base on the different quantifiable metrics. The KPIs are important as they are part of the criteria used to choose one alternative over another, through the cost models. A user should be able to specify which KPIs are the most important for them and their requirements for them through the cost model, and part of helping them do it is to present which KPIs are relevant and allow them to add those to the cost model. Not only can the KPIs be used for service and data selection, some of them are also relevant when evaluating a system in the distributed-to-centralized schema, like the implementation of the cost model would be. To choose services and data in the context of a distributed-to-centralized schema in the smart city, we need to look at KPIs for the cloud as the centralized component, for the distributed-to-centralized schema as a whole and for data quality.

2.7.1 Cloud computing KPI

These are KPIs based on what is found in the literature as important performance measurements and aspects of consideration when selecting a cloud computing provider service, as well as definitions on how to measure them.

Availability

Availability is often mentioned as one of the most important attributes of a cloud service. If the service can not be reached, it is of no use to the user requesting it, and how some of the other attributes performs at that time does not matter if it can not be accessed. Smaller and longer downtime of the service can be detrimental to the use case of those using the services and result in loss of business and sales [59]. A measurement of the availability of the system as an average over time, or the ”steady state availability” can be calculated by the following formula [11, p. 510]:

$$Availability = \frac{MeanTimeBeforeFailure}{(MeanTimeBeforeFailure + MeanTimeToRepair)}$$

The terminology used her is similar to the terminology used with reliability. The the two are tightly connected as failures can result in loss of availability, but also be available, but having the wrong external behavior. In this case, the failure in the ”mean time before failure” parameter should be considered a failure that results in the loss of availability. Similarly, as shown in [51], the availability which is called the ”robustness of service” can be calculated by dividing the time the system or service was available by the total time that has passed has passed for a given time period:

$$Availability = \frac{TimeAvailable}{TotalTime}$$

As these formulas only calculated the average over time, the results obscure information about the service downtime patterns, like variations in length of downtime and time between the downtime. Two different downtime patterns, one of many short outages and one with a few long outages might have the same measurement of performance with these formulas.

Instance Performance

Instance performance is concerned with how the provisioned instance from the cloud service performs in terms of computational power and speed. This can be done with several benchmark tests and measurements for different aspects of the instance. Benchmark testing is the comparison of system, programs or hardware according to some metric, that is done to evaluate the performance of the system [46]. This can be done by running different tests on the system that measure the different metrics. As done in [129] and [42], this can be done by using software running on the cloud computing platform that measure the processing power, the I/O to data speed and the memory access speed. It can also be done by running applications on the platform and test the response time from the virtual machine instances, as it is done in [35], instead of doing the measurements on the platform.
The CPU performance is measured by how fast it can process its tasks, which is its processing power. As modern CPUs are complex with different optimizations like branch prediction, caches, multiple cores, power limits and specialized hardware, a general purpose benchmarking tool or platform needs to make use of these when testing as well as covering a wide range of computational patterns, algorithms and different inputs and situations when testing a CPU. In the case with specific applications and uses in mind, the benchmark testing should use tests that are close to what the intended use for the platform is. In the case with [42], which seeks to evaluate the performance of cloud service providers for scientific computation, benchmarking tests with CPU operations similar to the ones used in scientific computation is used.
With memory and I/O performance, it is also measured on how fast it is. And as is the case with CPU performance, the memory and I/O performance is also dependent on the different types of task that is done, as some systems are optimized for different use cases. This can affect things like the write speed or read speed in different scenarios. Since not all data can always be stored in memory, the speed of the retrieval from disk and how the data is transferred to memory can affect how long time that takes for different scenarios. The same is also the case of writing data to the disk, so it is often the case that there is a trade off between optimizing for writes and for reads [31].

Network Performance

The network performance is concerned with everything that goes on in the network connecting the service to the user. The aspects of network performance can be measured quantitatively through different metrics. In [129], the metrics used to rank a cloud service provider network performance is the latency and the bandwidth of service. Packet delay

variation (also called jitter) and packet loss are also metrics that can be used to measure the performance of the network [43], especially in cases with application that stream real-time data that loses its value if it is not received within enough time from the last packet received. Examples of this can be video and audio streams and online video games. Latency is a measurement of the time duration between when data is sent and when it is received. Bandwidth is a measurement of the amount of data that is transferred each second in terms of data size. Packet loss is a measurement of the amounts of data packets lost, either as a percentage of total sent or the amount over a period. These are data packets that never reached their destination. Packet delay variation has two possible definitions, the first one is Inter-Packet Delay Variation (IPDV), where the reference of the delay difference is the delay of the previous packet, which is shown in Figure 2.7. The figure shows two lines which represents the packets sent, when they're received and the delay between. Calculation is done by summing up the differences between packet delay between packets and dividing by the amount compared pairs. The second definition is Packet Delay Variation (PDV), which uses the packet with the lowest delay as a reference of the delay difference between packets [82].

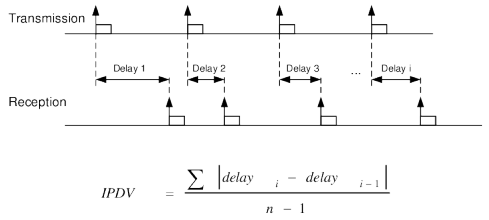


Figure 2.7: Calculation of IPDV, taken from [112]

Stability

The stability of a cloud service provider is in [51] defined as the variability in the performance over time. This can be specified in relation to the performance guaranteed in the Service-Level Agreement (SLA) or in relation to the average performance. The differences are accumulated and used to score the cloud service provider with respect to stability. In it, the calculation is done with difference from SLA for computational resources, while memory access performance is done with the variance from the average. The formula for this is presented below, where α is some measure of performance:

$$\sum \frac{\frac{\alpha_{avg,i} - \alpha_{sla,i}}{T}}{n}$$

The top numerator is the average performance for a user i, minus the average performance promised in the SLA for user i. This is divided the service time T, and this is again divided by the total amount of users n, and then summed up to calculate a total stability score based on all users experience of the service.
The stability of the bandwidth is also another metric which is mentioned [128] as a source of problem for mobile use of cloud services. The idea behind stability could also be used

together with other KPIs besides just instance performance, that also are dynamic and subject to change, and as something that could also be used to calculate the stability for a single user.

Security

Security can be divided into some subcategories or themes of security in the context of information security. That is the availability, identification authentication of the user interacting with the system, authentication of the user to allow access to the system, confidentiality to restrict access to private and sensitive information, integrity to make sure that only those authorized can manipulate data in the intended manner, and non-repudiation so no action can be taken that affects the system and not be noticed [103]. Security is not as easily subject to quantitative measurements in the same way as some of the other KPI, and is also not likely to be subject to changes as often as other KPI, but is nonetheless important in the selection of a cloud service provider. The security of cloud service providers can be scored by either experts or by using frameworks that scores the security [102].
In [122] a trust model is proposed that is structured like a tree, each node, except the root, being a subcategory of a security parameter. The leaf nodes are scored by the adherence to different security policies and a final trust value is calculated in the same manner as in AHP (link to AHP later). The user can select which parameters in the trust model to include, according to their needs. A similar approach is used in [71, 70] that uses Quantative Policy Trees (QPT) to calculate the a security score according to the security information that is part of the Service Level Agreement (SLA) of a cloud service provider.

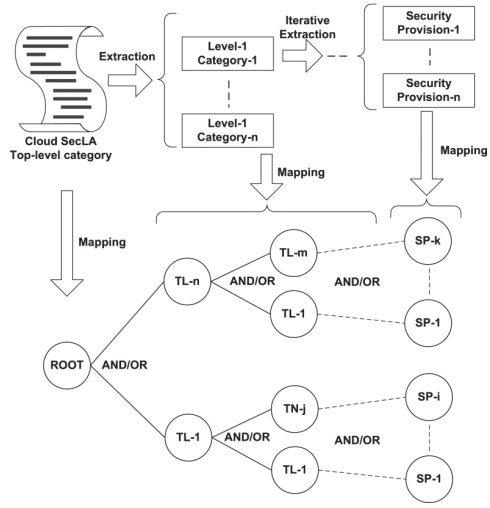


Figure 2.8: Converting the security information from a SLA into a QPT, taken from [70]

Figure 2.8 shows how the security information of a SLA, which is named SecLA in this case, is used to generate the QPT, which is an AND/OR tree representing the security provisions in the leaf nodes group under the different security categories in the nodes above. The leaf nodes have AND/OR relationship with their parents, that decide if a

single provision affects it, or all provisions affects its parent node. To create a scoring between the user needs and the security provisions of a cloud service provider, weights are set on the different security provisions in the leaf nodes to represent the importance of the security provisions. The next step is to calculate the parents of the leaf nodes. This is done by calculating the score the cloud service provider gets in the leafs, and the score the user requirements gets. This is calculated as follows: If it is an AND-node the values are aggregated or if it is an OR-node the smallest value is chosen. The value of the parent node is calculated with this formula:

$$NodeScore = \frac{CLOUDparent - USERparent}{MaxPossibleScore}$$

And for the rest of nodes above this, the score flows upwards with the same AND/OR logic of aggregation or choosing the smallest value below. This will give a final score to the cloud service provider. There are some contradictions in the logic in the papers, when it says that the OR-relationship is to be used for soft-requirements that only one provision is needed to fulfill the parent goal, but also saying that the lowest value is chosen due to the logic of the weakest link dominating the others in the case of the OR-relationship. This also means that giving low weights in the leafs can affect value that is chosen in an OR-node, which is weird as it is meant to intend low priority of a provision. If the max value was chosen instead, this would have made more sense.
Another approach of measuring security that is proposed in [113] by keeping track of the security breaching incidents in a system. It uses measures for confidentiality that is the percentage of authorized accesses among total accesses, data integrity is measured by the data accuracy before and after modification and privacy is measured with the rate of third party access to the data. An issue with this approach is that this information found after security breaches, which might mean that the security issue that caused the security breach is fixed and the numbers are no longer representable for the service.
A system that measures the security of a system through penetration testing is proposed in [154]. It is a third party system that attempts several attack vectors and then sends the metrics of the penetration testing to the test cloud service provider.

Elasticity and Scalability

Scalability and elasticity are two related concepts in terms of supporting increasing demands of a system. The scalability of a system is the ability of the system to handle increasing workloads and make use of additional resources regardless of what time frame this is done in. Elasticity on the other hand also takes the temporal aspect of scaling into consideration and is defined as the degree a system adapts to changes in workload demand by provisioning resources automatically to attempt to match the current demand to available resources. There are two dimensions of the elasticity, which is the speed of which the system can adapt to resource demand changes and the precision in matching resources with demand[57]. In order to account for the changes in resource demand, three different scaling options are used. Vertical scaling, which is the increase of computational resources to the system; horizontal scaling, which is the increase of virtual machine instance running; and virtual machine live migration, which moves the instance to a machine that is

under less load or with more resources [84]. In [57], a set of different metrics for elasticity is proposed, which are as following:

$\overline{A}$ = The average time it takes to switch away from underprovisioned states.
ΣA = The total time spent in an underprovisioned state.
$\overline{U}$ = The average amount of underprovisioned resources for underprovisioned periods.
$\overline{B}$ = The average time it takes to switch away from overprovisioned states.
ΣB = The total time spent in an overprovisioned state.
$\overline{O}$ = The average amount of overprovisioned resources for overprovisioned periods.

Using these metrics, you can calculate the elasticity for scaling up (E_u) and the elasticity for scaling down (E_d):

$$E_u = \frac{1}{\overline{A} \times \overline{U}}$$

$$E_d = \frac{1}{\overline{B} \times \overline{O}}$$

In figure 2.9 an example of the over- and underprovision of the resource is shown in a graph representing the resource demand and available resources over time. The metrics mentioned is also shown in the figure.

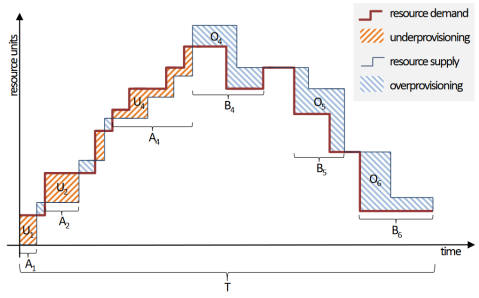


Figure 2.9: Capturing core elasticity metrics, taken from [57]

The same metrics are also included in [153], but also include other metrics for elasticity, that measure the amount of scale events, which is the number of "scale up demand" events (D_u) or "scale down demand" events (D_d) compared to the scale up events (A_u) or scale down events (A_d). This gives the scale up and scale down timings of the system:

$$Scale\_up\_timing = \frac{|D_u - A_u|}{D_u}$$

$$Scale\_down\_timing = \frac{|D_d - A_d|}{D_d}$$

If there is imbalance between the amount of events where there is a demand for scaling compared to the actual scaling events, it might be an indicator for bad timing behavior.
In [123], another measure of elasticity is proposed that is based on the definition of elasticity in physics, which is the ability of a a material to withstand stress. Stress is in physics

the force applied on a given area of the material. The strain is how much the material stretches. In physics, the formula for elasticity is $E = \frac{Stress}{Strain}$. This formula is also used for cloud computing, but with a new definition for stress and strain. Stress is the demand for computational resources divided with the allocated ones. Strain is the difference in bandwidth after scaling up divided by the bandwidth after scaling up and multiplied by the time it took to scale up.
Measurements are done through benchmark testing with similar metrics as above, or using simulations that allow the same [5].

Cost

The cost of the cloud service is specified in terms of the monetary cost of using the service. Specifying and comparing the cost between the different services is not straightforward, as the different cloud service providers have different cost and payment models that depend that is dependent on different factors. The cost can be related to the other cloud KPI, like the instance performance of the cloud service.
In [100], the two basic approaches identified, was either an advance payment which is common for web hosting where the payment cover a certain amount of computational units, or a consumption payment which is more common among IaaS providers, where the user is charged for the resources used. Another thing that is common is to use a hybrid between the two approaches, where you have to use advance payment initially, and then you pay for the additional resources used outside of the ones included in the advance payment. It defines three different metrics to measure the cost of the different cloud service providers: Cost per virtual machine per hour, cost per gigabytes traffic in/out and cost per gigabytes storage per month.
As a way to benchmark different cloud service provider with relation to cost, a cost for running a benchmark test with different tasks was done in [101], which is an example that a cost model can be defined in relation to measured instance performance as a way compare different cloud service providers that have different payment models. It also showed that there is a difference in price depending on the length of time or the size of the problem the cloud service is provisioned for.
In [51], in order to compare cost between cloud service providers with different storage, bandwidth and computational units, the cost is defined in relation to those attributes with the following formula:

$$Cost = \frac{Price}{cpu^{\mathrm{a}} \times net^{\mathrm{b}} \times data^{\mathrm{c}} \times RAM^{\mathrm{d}}}$$

In this formula, a, b, c and d are weights representing the relative importance of each attribute, and should sum up to one: $a + b + c + d = 1$.
An extensive and detailed for predicting the cost for cloud computing is proposed in [141] together with a cost estimation, monitoring and analysis service. The four first basic components for modeling the cost are as following:

$M_{\text{dataStorage}} = size(total) \times t_{\text{sub}} \times cost(storage)\ where\ t_{\text{sub}}\ is\ the\ subscription\ time$
$M_{\text{Computation machine}} = cost(machine)$
$M_{\text{Data transfer into cloud}} = cost(transfer_{\text{in}})$
$M_{\text{Data transfer out of the cloud}} = cost(transfer_{\text{out}})$

These components are used together with execution models to create more detailed corresponding prediciting of cost for different workflows and for different computational execution models, like sequential or parallel programs running on a single or multiple machines.

Reliability

The reliability for a cloud service is tied the successful use of the service functions and that it almost always provides the correct results and guarantees against loss of data. This in turn is tied to some of the other performance indexes, like availability[2]. Events where the unsuccessful use of the service functions is detectable externally, is known as a failure, which is in turn is caused by a fault in the system. A system can be resilient to faults, which mean that it prevents or handles faults so that they don't cause a failure [11, p. 80]. In [51], the reliability of a cloud service provider is measured by the chance of failure based observations done by the users multiplied with the cloud service provider's own promised mean time to failure:

$$Reliability = Probability\,of\ violation\ \times Promised\,mean\,time\,to\,failure$$

$$= (1 - \frac{numfailure}{n}) \times Promised\,mean\,time\,to\,failure$$

Another aspect of reliability, is the fault tolerance of the system. In [49], two metrics are used to describe the fault tolerance when it comes to the consequences of faults or disasters and data loss: Recovery Point Objective, which defines the amount of data that will be lost during a fault or disaster, and Recovery Time Objective, which determines the minimal downtime when recovering from faults.

Usability

How easy it is for a user to understand and be efficient and effective with the cloud service. Can be quantified by the average time it takes a user learn and become proficient with using the cloud service [51]. For both first time use and setup of the service, and continued operation on it. As well as the possibility to customize the service to fit more in line with the user needs [108].

Management of services

This part is not tied directly to the services themselves, but the company that provides the services. The accountability of the cloud service provider is often detailed in the SLA, which provides guarantees about different aspects of the service, as well as repercussions if cloud service provider fail provide a service according to the guarantees given. The cloud service provider should provide support and information, and specifications of what kind of support is given under different circumstances [108].

Features

Features to the service or add-on services that comes in addition the service itself can be important when deciding to use a cloud service. In [51], it quantifies this by suitability, which is the amount of non-essential features that match with the non-essential features needed by the customer, sustainability which depends on the amount of features provided by the service in relation the ones needed. It also categorizes interoperability as the amount of supported platforms by the provider in relation to the different platforms needed by the customer.

Reputation and Trust

In cloud provider service selection, finding and choosing the right cloud service provider is difficult enough with the different service descriptions detailed in the SLA that comes with different trade-offs. But coupled with the fact that the cloud service provider might not provide what is promised, makes it even harder to find the right one. But by using the knowledge gathered by other users' experience of the cloud provider services, expectations of the service can be brought more in line with the actual performance of cloud provider services.

There are two approaches to the reputation models for cloud services. The first one is similar to the one used for restaurants, movies and in online stores, which is the subjective feedback and ratings provided by the users of the service which is aggregated and used to create a score for a cloud service. This approach is used in [151, 53, 89, 1, 88]. Another approach that is used is to use performance measurements gathered from the cloud services by the users as data to score the cloud services reputation. In [3] a reputation score is used as part of a service selection system that schedules jobs to different web services. In order to select the best services, a reputation score created for the services by gathering information about completion times of jobs and service failure information for the jobs scheduled. In [162] a trust and reputation system is proposed for cloud service providers and sensor network providers. The trust of a cloud service provider is calculated by the error of calculation, third party access to the data and if it is able to transmit the data to the user. The reputation is calculated by the number of users that chose to use the service. In [74], the trust score calculated by user submitted information about the availability, the reliability, the data integrity and the time it takes to process jobs submitted to the service together with the weights submitted by the user in the same manner as in linear programming, presented in section 2.6.1. In [95] there is a hybrid approach between measurements and subjective user feedback where the trustworthiness of an IaaS provider is calculated using a mix of the measured of the compliance to the SLA, the times a user chooses to use the service, and the satisfaction feedback given to the users. A problem with reputation based ranking of services where anyone can provide their feedback, is the risk for malicious and false feedback that is used to distort a reputation [90]. Several of the approaches to reputation and trust management take that into consideration and attempts to filter out that feed.

2.7.2 Distributed-to-centralized KPI

These are different KPIs found in the literature for measuring the performance of distributed-to centralized nad F2C systems and the applications running on them.

Processing load balance

In order to not overload or burden single processing nodes in the F2C with too many processing task, or letting nodes be idle without using their processing capacity, it is important to distribute the computational tasks evenly among the processing nodes. In [132], load balancing among nodes is one of the goals in their service allocation strategy for F2C. In the model, each node have a designated amount of slots to run small atomic services that might be a component of a larger service. The load in each layer of the F2C is measured by the amount slots that are used or not in that layer. The importance of utilizing resources and reducing load is also mentioned in [78]. The need for efficient utilization of resources, reduction of the computational processing on the higher levels near the cloud and being able to offload computation from devices with less computational capacity are issues described in [157]. By reducing the computation needed in the cloud, through doing the computations near the edge, the overall system can be made more scalable, while at the same time supporting devices near the edge that needs computation offloading [33]. This is an important aspect of the F2C-architecture, to be able to support the large of amounts of data that a smart city can potentially come to produce. Taking a look at the specific parts and comparing the processing load of these in isolation can also help us gauge the performance of the architecture.

Energy consumption

In order to be environment friendly, reducing the energy consumed is important. But another reason that energy consumption and energy efficiency is an important focus, is introduction of devices that can provide computation that have a limited power and energy. An example of this could be a smart phone, that is dependent on its battery for power. The service allocation strategy in [132] also takes into consideration the energy consumption in the service allocation, attempting to balance the energy consumption among the different fog in the F2C-architecture. In [104], a power consumption parameter for performance is proposed that is the total power consumed at both fog and cloud premises, although the study only considers power consumption at the cloud, as it would require a different approach to include fog devices. There is a trade-off between the response time and power consumed in response to growing workload allocation in a F2C-system, which is simulated in [37]. It found that the power consumption decreases when it is allocated to the cloud as it is more energy efficient, but allocating it to resources in the fog reduces response delay, so it is a balance between wanting fast response times and saving energy. The importance of energy efficiency is also mentioned in [33, 78, 157].

Latency

One of the arguments that have been put forward in favor of the F2C model in favor of a centralized cloud model is the improvement in latency by having data and processing

resource geographically near where it is needed. In order to support real-time and applications dependent on low latency, this is needed. In [37], the trade-off between power consumption and response time is looked and and a model is created and tested through simulation. In [132], the allocation focuses on minimizing latency as part of the goals in service allocation. The service response time is mentioned as one of the most importance performance metrics in [104], and in testing two service allocation strategies is measured by the average service response time. Experimenting with the response time using fog layers near a pipeline monitoring system measures the response time of a pipeline monitoring system in a F2C-architecture, and shows that the response time significantly lowers for hazardous events compared to using the cloud [139]. One of the three design goals for a fog computing platform in [157] is the latency that must be sufficient to provide low-latency guaranteed applications and services. Also in [78] the performance measurements when comparing the cloud, the F2C and an optimized F2C is the response time on a scenario with service tasks that is simulated on the different platforms. Almost any source on F2C mentions the latency as one of the advantages and performance metrics of the architecture.

Volatility and reliability

The devices at the in the fog and near the edge is not only different from the cloud in their processing constraints and energy constraints, but also in terms of volatility. You can be somewhat sure that the cloud provider will be there tomorrow with their guarantees of nearly 100 percent availability, but when devices like smart phones and computational devices in a car enters the pool of computational resources in an area, knowing how long they will remain available is less certain. This is mentioned as one of the challenges in [104], where the volatility in the fog layer and the service disruption probability is proposed as important metrics for evaluating the fog layer and how the mobility and power constraint might impact system. The service disruption probability is the amount disrupted services during execution. The issue of volatility due the intermittent presence of computing resource due to their mobility is also mentioned in [78]. This dynamic nature of the computational can also influence the quality of the services provided if they do not straight up disrupt the services, impacting the reliability of the services.

Network load and traffic

When being able to process the collected data near the edge where it is generated, it can be processed to be made smaller in size, as well as sending it at times where the overall network load is low. The data storage size and the data sent from the edge, fog and to the cloud is looked at in [125]. Through using data aggregation and compression in the different layers, the amount total of amount of data sent will also be lower. The reduced data size is also one of the parameters in the test of a pipeline monitoring system implemented in a F2C-architecture, where it showed a significant decrease in the data sent [139]. Network bandwidth occupancy is defined as the amount of traffic to the cloud in [104], but does not include the traffic on the lower fog layers. Offloading the cloud's bandwidth is also a stated goal in [37], and a constraint included in the model it proposes. The reduction of network traffic stated as an advantage of using the cloud in [33], and that is it not sensible to send

raw data to the cloud as there is estimated a large amounts of data generating devices in the future.

Security

With the devices no longer being secured on a single location, the lack of computation power and energy, as well as the possibility almost anyone being able to provide computational power, there are some new strengths and weaknesses when it comes to security. Without the strict control of the provider, the devices might be physically available to attackers, but also comes with the control of not having all data centralized in a single place that could be attacked. The issues to security and privacy in fog is unsolved might need their own approaches [78]. In [157] the security and privacy is mentioned as important in the design of a fog computing platform, and would need support on every layer in the form of access control and intrusion detection systems.

2.7.3 Data Quality

The quality of data is relative to the intended use of it, and can Quality of Data can therefore defined as the fitness for use of the data [140]. It has also been defined as the agreement between the data and the real world phenomena it represents. The different dimensions of data can be classified into for different quality classes according to [135]. These are:
1. The Intrinsic Data Quality, that is concerned with the attributes data themselves, like the accuracy.
2. The Contextual Data Quality, that is characterized by other characteristics of the data, like how complete the data set is or how new the data is.
3. The Accessibility Data Quality, which is how and who are able to access the data.
4. The Representational Data Quality, which is how the data is presented and how easily it can be interpreted.
Data can also be classified as either soft or hard data, where the hard data is quantitative data, like sensor measurement, and soft data which is qualitative like a customer evaluation [9]. In [10, 96, 68], several dimensions of data quality is identified, which will be further expanded upon in subsections below.

Accessibility

How easy it is get access to the data for those who should have access to it. This can be cause by the access mechanisms themselves not working, or that the understandability or interpretability of the data is such that making use of it is hard. It can also be caused by data not being timely enough or that the data amount is large enough to cause problems processing it [135].

Accuracy

Accuracy in data is defined as deviation from the state it originally is supposed to have or the real world state it is supposed to represent. In [91], it is given as the percentage of

correct entries in a data set. It can also represent the accuracy of a single measurement where the accuracy is the closeness between the measured value and the actual value of the object that is being measured [80].

Amount of data

The amount of data should be appropriate for the task at hand. Can be specified as the ratio between the amount of data needed by the user and the amount of data provided [96].

Believability

Defined by [150] as ”the extent to which data are accepted or regarded as true, real and credible”. Can be measured by using the trustworthiness of source of the data, the reasonableness of the data and the degree which the temporal aspect of the data also is credible, according to [99].

Completeness

Completeness is defined as the degree which all entity objects that exist in the abstract universe is also represented in the data set [52]. The abstract universe is in the case the specific instance or area we wish to represent in data. As an example from [67], the abstract universe can be all instances of diagnosed tumors in a country, and the data set is the registry of these incidents. The completeness would be the degree to which all instances are in the data set.

Concise representation

The representation of data, and if it is concise, meaning that is conveys meaning while still being as compact as possible both in format and presentation [68].

Consistency

May be related to the data values themselves or the representation. In the case of data values, a data is consistent if any redundant data is consistent across several tables, which might be know as a Referential Integrity constraint in a case with a database [96]. With the representation consistency, the focus is on whether the representation is in a consistent format [68].

Ease of manipulation

Or the Ease of operation. This is easy the data is to manipulate for the users needs, how easy it is to aggregate and how easily it can be combined with other data [68].

Granularity

The granularity of the data is the level of detail represented in the data [38]. For geospatial data, data for an area might be represented for each square meter, or for a square mile. For temporal data, it might be a measurement every second or measurement every hour as examples of differing levels of details or granularity. Different temporal granularity of measurement might also be referred to as the sampling rate.

Interpretability and Understandability

How easy it is to interpret and understand the data or information [68].

Objectivity

The degree to which the data is impartial, unbiased, objectively collected and based on facts [68].

Precision

Defined as ”The closeness of agreement between independent test results obtained under stipulated conditions” in [80]. Which means the degree to which different measurement data of the same object under the same conditions are close to each other in value. So to compare precision with accuracy, if we have high precision, but low accuracy, the measurements are always nearly the same, but deviates from the actual temperature.

Relevancy and value-added

The degree to which the data is relevant for its use and the value-added to the users and the organization of the users. The value-added is the value or the utility gained by the user or organization by using or owning the data [9]. The relevancy of the data ties into the value-added, as poor relevancy of the data means that there is less use for it [135].

Security

The security is tied to the availability of the data, which is the extent the data is restricted properly to allow for its security [96]. Only those who are intended to have access to the data are the ones who it should be available for. This is related to the security KPI for cloud and F2C as well.

Timeliness

Describes the temporal dimension of data and how up-to-date the data is in relation to the task it is going to be used for [96].Some uses of data require data that is very recent or real-time data with very low delay in order to work. An extreme example is the high frequency trading applications where just a few milliseconds of delay can mean you lose against a faster trader [21].

2.7.4 Context Information

Often defined as any information that can be used to characterize the situation of an entity that is considered relevant for an interaction [66]. An example of an entity can be a smart phone, and the context information in this case can be the spatial position of the device or the weather in that area. When the information is generated, there are often several sources of context information, like the characteristics of the sensor, data about the situation for the measurement [66].

Quality of Context

The quality of context describes the quality of the information that is used as context information. It can be used to determine the value of that information [66]. This information is in itself subject to many of the dimensions that the ones mentioned for quality of data [75, 25].

2.8 Service registry

Part of managing services and data, is being able to provide access and information about them. In many instances, data and services might be distributed themselves, which means part of the information on how access the data and service also needs to include how to connect to them, directly or indirectly. In a smart city scenario, and in the scenario with the ZEN pilot and the ZEN ICT architecture, which will be explained in 2.10.1 and 2.10.3, the data is distributed around different locations in geographically proximity to the pilot areas or further away. Which is why looking at different ways to store and use that information is important to create a solution for the described problem. In this case, the term registry is used as overall description of approaches used to store information about services and data, and information on how to access them.

2.8.1 Centralized registry

A simple way to manage a distributed set of services and data repositories is by storing the information about the services and data in a single centralized system. A common pattern for implementing this for services, is by using the ”Broker Pattern” [11, p. 212] to create a service broker that allows services to add their services to it, and for clients to help match with a suitable service.
Figure 2.10 shows the broker pattern. The central component is the Broker itself, that can be connected to multiple services/servers and multiple clients. As shown in the figure, the client and servers uses proxies to communicate with it, and the broker implements a bridge component to allow indirect communication between the server and component. The same pattern can also be used as an initial way to pair clients and servers, and the allow direct communication, but this needs established protocols between the components to work.
Similar patterns can also be used for distributed data storage, where a centralized component holds information about distributed data and how to access it. One of the more

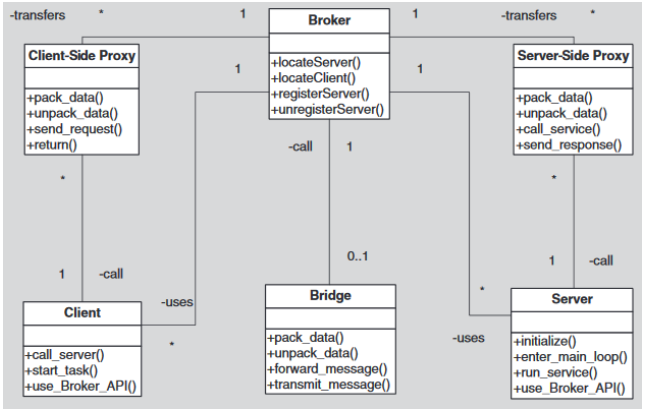


Figure 2.10: The broker pattern from [11, p. 212]

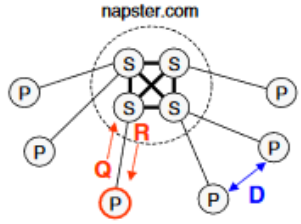


Figure 2.11: Napster architecture from [114]

infamous instances of a distributed data storage and file sharing program was called Napster. Napster utilized a cluster of dedicated central servers keeping an index of files the were being shared by the users of the of the network [114], as illustrated in figure 2.11.

The nodes S are the central servers, and the nodes P are the peers of the network. A query Q is made to the central server, which then responds with a result R that contains the locations of the peers holding the files queried. A peer-to-peer connection D could then be made to transfer the files.

The downside with a design like, this is that it can be a bottleneck, and in the case of Napster actually was a bottleneck [55]. It can also become a single point of failure, that renders the systems depending on it useless if the central component goes down, and depending on the degree of communication going through the central component, also add latency since communication has to go through it [11, p. 210]. But it might be a less complex approach to ground a distributed system in a centralized component, instead of having every part being distributed.

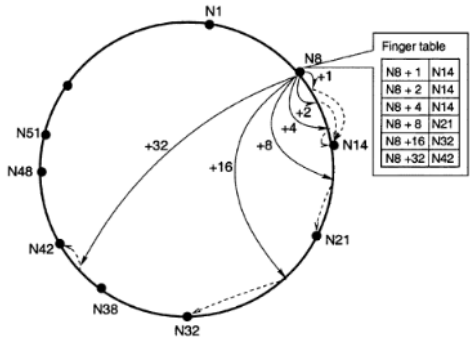


Figure 2.12: Circular key space and finger table for Chord from [134]

2.8.2 Distributed hash tables and routing

Instead all information about the whereabouts and properties of data and service being gathered in a single place, this information can also be distributed as well. One of the most important parts of the internet, the IP-protocol and routing tables used to forward packets of information, are based distributing the knowledge of the paths to the address location, and if a router can not tell you exactly where the address location is, it at least hold enough knowledge to send the packet in the right direction.
Similarly for files and data, distributed hash table (DHT) can be used to find data. Specifically for peer-to-peer applications, where the former centralized model in Napster caused scalability issues, DHTs is an approach to deal with that problem as well as allowing for a much more distributed system. In these file systems, each file is associated with a key, often a hashed value of the name of the file, where the nodes in the system is responsible for storing a range of its keys [105]. Even though there might be a large amount of nodes holding their part of the information, the different routing algorithms presented in [105] have a path length of max $O(log\ n)$ for n, which means that it is scalable, and even for large networks, the time to find and retrieve data is not too long.
In one of the first scalable routing algorithms for DHT, called Plaxton routing from [97], a node is given a number, like 12345, if it is given a query for 12689, it forwards it to a node 126xx, so it must know a set of nodes that shares its prefix, but differ in the next digit. The downside of this approach is that it works best for a static environment where nodes doesn't suddenly appear and disappear. Other routing algorithms for DHTs can deal better with an dynamic, like Tapestry [159], which is variant of Plaxton that can adapt to nodes appearing and disappearing.
Other approaches, like Pastry [110] and Chord [134] uses a circular key space, where the nodes has a set of nearby neighbors, which are near in key space, but also has routing to nodes that are further away in the key space for a short path. This is shown in figure 2.12, which displays "finger table" of Chord, that is an exponentially spaced list of other nodes in the network, shown as solid arrows. This ensure efficient routing.
While the benefits of a distributed designed right might be the removal of bottlenecks, and more robustness, there are still some potential issues, mentioned in [105]. Certain

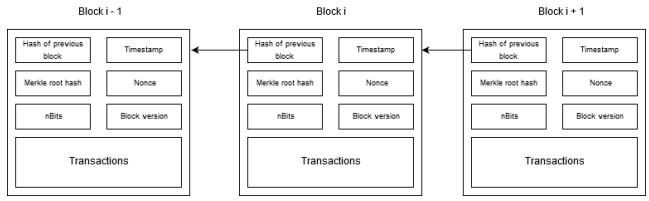


Figure 2.13: The data structure of the blockchain showing the blocks and their content

nodes can be come hotspots for much of the network activity through having popular files. And even though the routing promises few hops between nodes to reach the destination, the nodes can in reality be far apart geographically and causing the queries to have high response time.

2.8.3 Blockchain

Blockchain is at the core of the cryptocurrency Bitcoin, which is implementation that is used as the basis for the explanation of the blockchain here. There are several other areas of its use, and variations on its implementation that also exist, which also will be discussed. Especially the applications for IoT and Cloud is interesting in this case. While blockchain in its narrowest sense can considered to be just a data structure of linked data blocks forming a chain of data blocks where each block is dependent on the prior block through hashing of the content of the block and new blocks pointing to these hash values [50], which can be used as a ledger that is hard to tamper with. However, the overall framework of enabling technologies used with blockchain is usually included in the use of the term. These enabling technologies are: Peer-to-peer (P2P) networking eliminating the need for a central entity for communication, allowing distribution of the ledger throughout the network; Consensus algorithms that decides who gets to add a new block to the BC and verifies the new block; Public/private key encryption for digital signatures to authenticate the communication in the network.
Figure 2.13 shows the structure of three blocks in a BC. Each block, except the genesis block, which is the first block of the BC, have a pointer in the form of the hash of the previous block in the BC. The hash of the previous block is the hash value of the previous blocks timestamp, Merkle root hash, nonce and its own hash of the its previous block [160].
Figure 2.14 shows an example of a Merkle tree, and how a Merkle root hash is generated from the transactions in the data block. All the values that is used to create the hash of a block, is what makes it so hard to tamper with a blockchain. Changes to any transaction or the other fields will change the hash of the block, which will in turn change the hash value of any blocks after it. In addition to those fields, there is also the block version field that indicates the validation rules for the block, and the nBits field which acts as the hashing target for calculating the nonce. The calculated nonce must give a block hash value below the nBits hashing target. This is something that makes finding a nonce harder, which is important for some ways for reaching consensus and a source of computational overhead.

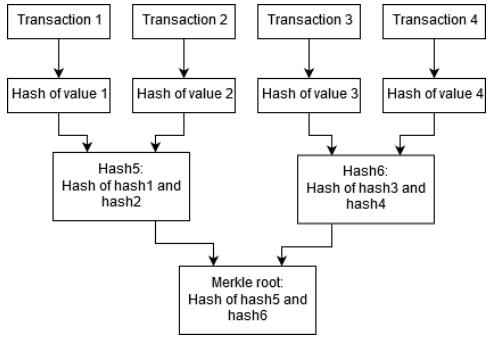


Figure 2.14: The data structure of the Merkle Tree used to generate the Merkle root hash used in the blockchain data structure.

In a network with no central authority, it is not straightforward who gets to add a new block to the BC among equal nodes that hold a copy of the blockchain. This leads us to the use of consensus algorithms as part of the blockchain solutions. This is an area where there is a lot of difference between different BC solution, both for cryptocurrency and other areas of use.

In Bitcoin, Proof-of-Work (PoW) is used as the consensus algorithm to add new blocks. Every node in the network is allowed to mine for a nonce that makes the block hash satisfy the nBits hashing target. It is however computationally intensive to solve, especially as multiple nodes at the same time are mining for this nonce, which is a point of critique against it [94]. It needs to be difficult to limit how often a miner can append a new block to the BC. If the interval is too small, too many forks are made to the blockchain and double spending is a potential danger where someone can pay twice with the same crypto-coin. In the case of forking, the longest blockchain becomes the used one, while the shorter branches are abandoned. For identity management in a system where privacy and anonymity are some of the key features, public/private keys are used. Transactions are signed with the private key by encrypting a hash value created from the transaction, and also providing the public key, so that other nodes can verify the transaction decrypting the digital signature with the public key [160].

The advantages of blockchain is the decentralization of the system, which in turn saves costs as the computational cost and storage is pushed out to the participants of it, but also allows for a system as a whole that is resistant to attacks and faults, as the malicious or faulty behavior from nodes need to affect a large portion of the participating nodes before the system is no longer working. This at the same time as it allows anyone to join and participate in the system without certification, while not sharing private information about the participants. However, it is very computational demanding, stemming from the PoW being run in parallel on all miners, spending large amounts of electricity. Another issue is the scalability, as the transaction are stored in the blockchain, it grows in size, and fewer nodes are able to hold the full blockchain. To distribute the full blockchain also causes much load on the network as well. To limit the growth of the blockchain, there is a limitation on the block size, together with a limitation on how often a new block can be

added controlled by the difficulty of calculating a nonce for a valid block. This limits the overall transaction throughput.

Other blockchain consensus mechanisms

As mentioned in the Blockchain segment, there are several implementations of blockchain, that are both used for cryptocurrencies, and other purposes the differ in the implementations of different components in their blockchain framework. One area with many different approaches is the consensus algorithm used to add block to the blockchain, as this is one of the problem areas that causes the computational overhead, scalability and low throughput of transaction problems of the Bitcoin implementation of blockchain. The consensus algorithms can be categorized as either Proof-of-* or Byzantine Agreement. In the Proof-of* approaches, the basis for it is the selection of a leader to validate a new block and distribute it to the other nodes, while in the Byzantine Agreement approach, this is done through majority voting [94].

Proof-of-Stake (PoS): With a Proof-of-Stake approach as the consensus algorithm, the influence of each participant is dependent on their stake in the network, like the amount of currency held. It could be used to increase chance of leader selection, or in the case of Hashgraph increase their influence in voting [8].

Proof-of-Space or Proof-of-Capacity: Instead of using a computational difficult problem like in PoW, another approach that can save the energy expended in the PoW approach, instead uses problems that requires much space instead computations to solve the problem, while still being easy to verify the solution. Another believed benefit of this approach, is that it the advantage in PoW mining from specialized equipment over a CPU is not as big with this approach [107].

Proof-of-Burn: The idea behind Proof-of-Burn is similar in some way to the Proof-of-Work in that there is some cost to proposing new blocks, but instead of wasting electricity and time, some currency is sent to an address where the currency no longer can be used. The transaction will work as proof of burning currency [93].

Proof-of-Activity: A combination between PoW and PoS. The first part of the consensus approach consists of mining a nonce that creates a hash smaller than the hashing target. Stakeholders are selected based on owning a randomly selected currency unit, which increases the chances with more currency owned. This must sign the hash of the new block with their private key [14].

Practical Byzantine Fault Tolerance: The algorithm in general works on the concept of having every participant in the network vote on the correct response. It consists of three phases. The first phase is the pre-prepare phase where a participant sends request, or in the case of a blockchain, a value or block that should be added to the blockchain. After that, the prepare phase starts, where all the participants broadcast their response to the request, or in the case of a blockchain, the value or block that should be added is broadcasted throughout the network. The last phase is the commit phase, where the result is accepted by the nodes if 2/3 of the participants agree on the result [24].

Ripple: Each round the participants of the consensus process gathers the valid transactions they have received. The participants keep a list of other participants that they query during the consensus process, which they use to share and combine valid transactions with and vote for their inclusion. If a transaction gets at least 80% of the votes, it is included.

If not, it is discarded or kept for the next round of the consensus process and might be included there [117].

Algorand: This is a byzantine agreement algorithm. At first a leader is selected at random, with higher chance of being selected depending on the stake in the network, who proposes a block to be added to the ledger. A committee of verifiers is randomly selected to approve the addition of the block, which digitally signs the block, before it is broadcasted to the rest of the network [27].

Ethereum and smart contracts

Ethereum [40] is an open BC platform with different features than the BitCoin implementation of a BC. Ethereum is intended to be featureless and value-agnostic, and not specially designed for a single purpose, like cryptocurrency. It is however suited for applications with automatic and direct interaction between peers or coordinated group actions as it is a programmable blockchain where smart contracts can be implemented. These programs are run on the Ethereum Virtual Machine(EVM), on the different nodes in the network in parallel to maintain consensus. Ethereum is Turing complete, and can execute code of arbitrary complexity. The use of smart contracts is enabled through the use of contract accounts, which are different from the externally owned accounts that are controlled by the owner with their private key. The contract account act based on their internal code, and if it acts based on human control, it is programmed because it can be programmed to do that. Smart contracts are in this case the code in the contract accounts that execute when sent a transaction. As this code is required to run on all nodes in the network, the outcome must be deterministic to have consensus between the nodes. When someone sends a transaction to a contract account that triggers computation, they are required to pay in Ethereums own cryptocurrency, ether, for the computational and storage resources used by the contract to the nodes that perform these operations as miners in the network.
This means that it might not only be used as a basis to store things in, but also be an more active part in transactions and management of different parts of the smart city through its smart contracts. Not only does this make it possible to access data and service through the blockchain, but also to buy and sell access to these through the platform.

Similar approaches

IOTA Tangle [98]: IOTA is a cryptocurrency that is built on a data structure different from the blockchain. Instead it uses a Directed Acyclical Graph (DAG), to store the transactions in, called The Tangle. It also uses a different method to reach consensus instead of the use of miners that is often used in different blockchain platforms. It still uses a puzzle similar to proof-of-work to prevent the spam of transactions, by finding a nonce.
Figure 2.15, shows the data structure of the tangle. Each square is called a site and represents a single transaction. In order to add a transaction to the tangle, it must first approve two older transaction in the tangle and solve a puzzle to be allowed to add its own transaction. As transaction are directly confirmed or indirectly through the transactions that confirmed them being confirmed, they accumulate weight which means that it is more trusted. The weight set on the transaction itself is dependent on the puzzle it solved. The communication between the nodes in the network is through mutual tethering, so a single

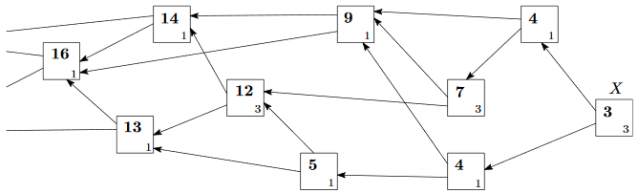


Figure 2.15: The Tangle, which is the DAG holding all transactions [98].

node does not have direct contact through all other nodes, but changes to the network is propagated. The network is also asynchronous, so the nodes can hold different and conflicting transactions in their version of the network, but those transactions will have one of the transactions being orphaned as nodes have to chose between them. The tangle is vulnerable to attacks in the start, so it is currently using a coordinator that eventually confirms the transactions in the tangle. This is a centralized approach with the coordinator, which is planned to be removed once the platform can work safely without it. This is currently an ongoing research, on how and when the coordinator be removed. The coordinator's responsibility is to confirm transactions, by issuing a signed transaction called a milestone, that confirms transactions by referencing them [61]. While it is not clear when the approach can become truly decentralized, there are also some issues with wasting computational, if there is an amount of transaction throughput that is needed for the system to stay secure, there might be a need for honest nodes to constantly sign transactions, even empty ones to make use of their hashing power to prevent adversarial attacks on the DAG [19]. Otherwise, it is an interesting approach in making the users of the systems the miners, but is still in a researching phase despite IOTA tokens being available for what now is a somewhat centralized system.

Hedera Hashgraph [8]: The Hashgraph is also another different approach to the distributed ledger which uses a different data structure than the blockchain to contain the transactions and information within it. That structure is the hashgraph, which is held by the nodes in the system. It consists of a graph of events which contains the transactions, being spread by a gossip protocol. A gossip protocol is a protocol where a node shares everything they know with a random node in the system to spread the information. They do not only spread the transaction, but information of the event of who messaged who which is used to keep the ordering of the event. A thing that is special for this implementation of a public ledger, is that it maintains fairness in the ordering in transactions, meaning that if someone adds a transaction before someone else, that order is kept in the ledger. Figure 2.16 shows the data structure of the hash graph, where the lines are the time for each node, given names, and the dots are gossip event, connected by who gossiped to who. To achieve consensus whether an event happened or not, it uses what is called virtual voting, where nodes calculated what another would vote based the information it holds, where it will confirm it if more than 2/3 of the other nodes can be calculated to agree on it. But this majority is not based on the number nodes agreeing, but their stake, which is represented as positive number for each node, and the 2/3 majority is when 2/3 of the total stake agrees on it. So it implements a sort of a proof-of-stake as part of the consensus algorithm, to discourage Sybil attacks by

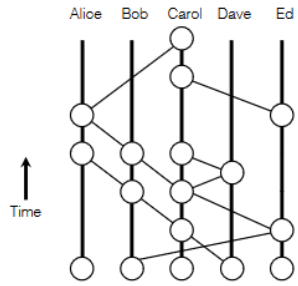


Figure 2.16: The Tangle, which is the DAG holding all transactions [98].

creating loots of node would not be able to get 2/3 majority because of their lack of stake in the network. It claims to have a high transaction throughput and speeds and to be more bandwidth efficient compared to similar Byzantine Fault Tolerant consensus approaches due to the virtual voting. But it is currently patented and only exists as a private network for now. So any use is limited outside of that.

2.9 Service description

In addition to the approaches on how to store information about services and data. There is also a need to decide which types of descriptions of the service or data is needed to be stored together with the information on how to access them. What it is included in the service descriptions in the case of this thesis is likely to have some overlap with the KPIs introduced in section 2.7.

2.9.1 Web Service Description Language (WSDL) and OWL-S

WSDL [29] is an approach on how to describe web services. WSDL documents are used to describe or define services as a set of network endpoints, or ports, in a XML format. It has an abstract and a concrete definition of a web service, where the abstract definition defines with the web services does through defining operations and the messages in the form of input or output, while the concrete part adds information about what protocols is used to contact the service and at what address this is done. This provides a standardized way to describe what the service does, and how to interact with it. This can as an example be used to together with Universal Description, Discovery, and Integration (UDDI), which is registry where services can be stored that maps to a WSDL document, using Simple Object Access Protocol (SOAP) as the communication protocol to the registry, and to the services [81]. Similar work with describing services was also done by creating OWL-S, which built upon the Web Ontology Language (OWL), attempting to describe semantic web services by a ”ServiceProfile” describing what it does for the user as advertisement, a ”ServiceGrounding” describing how to interact with it, and a ”ServiceModel” that explaines how it actually works. These would themselves have several concepts below them,

and the whole thing would be formatted in XML as well [76].

2.9.2 Cloud service descriptions

With the new paradigm of cloud computing, some work has also been done describing services in the cloud. In [111] a XML formatted schema for describing cloud storage services is proposed, as part of an automated approach to cloud storage service selection. It describes the storage services as containers that can hold different objects, with their own descriptions in form of types. It also describes functionality and available operations of the services, as well as the performance and costs for the service. This information is used as a basis for selection.
Work on descriptions that covers cloud hosted services as a whole, looking at Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), covering all these cloud computing layer abstractions under a single abstract description of cloud services, is proposed in [86]. It consists of a template for cloud services formatted in XML, that can be extended with more sections as needed. The most basic properties registered, is an unique id, description, ownership, version and release date. The next section is a offering section that describes the functional capability of the service, technical interfaces, QoS specifications and policies. There is an implementation section specified for those that deploy and provide provisioning of the service, that can describe configurations and dependencies. A resource requirement section that describes the resource requirements of the service. The description provides information both to the consumers of the service as well as those that implements it.

2.9.3 Fog-to-cloud taxonomies

In papers [119, 120, 121], it is identified that a definition for the resources and the service-tasks in Fog-to-cloud (F2C) computing. Looking specifically within a smart city scenario, which is the case in all the papers, there is a lot of heterogeneity and variation in the different devices that will be part of the F2C computing platform. The first part of creating descriptions that can be used to allocate services to resources, is to create a taxonomy of the resources and services for the F2C platform in a smart city scenario. The first concept in the taxonomy, is the F2C resource, which is component that by having an entry point and the ability to run the software so that it can participate in the overall F2C system, is considered a F2C resource. It has a set of five main characteristics, that themselves also have their own characteristics [120]:

- Device attributes
- Cost information
- History and behavioral information
- Security and privacy aspects
- IoT and attached components

Not every characteristics will be presented here, but we will have more specific look at the ones related to data collection, which is the IoT and attached component characteristics. IoT components have the following characteristics:

- Observed properties
- Operating properties
- Measuring properties
- Survival properties

But in addition to that, a more specific characterization for sensors is also proposed, with the following characteristics:

- Survival range: Working lifespan of device
- Accuracy: Same as accuracy presented in section 2.7.3
- Frequency: The rate of capturing and sending data, same as granularity in section 2.7.3
- Geo-location: The geo-spatial position of the sensor itself.
- Latency: The delay between the data is captured, and when they are transmitted.
- Measurement range: The range in which the sensor is able to measure.
- Feature of interest: The feature of interest is the object that the measurements are taken from. If a sensor measuring the CO2 in a room, the room is the feature of interest, and the CO2 levels is a quality of it.

There is also characterization of services and tasks, a service consists of one of several tasks, that needs to be completed in order provide the service. A service is characterized by the context of the service, like if it is tied to governmental, educational service; the service location, which is where in the F2C system it is provided, like the cloud or fog; the security and reliability preferences; the data characteristics, in what type of data processing is needed; the cost of the service, or if it is free; and at last, which tasks it consists of.

The characteristics of the task is mostly tied to their computational demands, like the network, storage, processing, power, and memory requirements; data requirements, related to what type of data is needed; and the time requirement, specifying how much time the task has to execute. In [121], this taxonomy is used to create descriptions formatted in XML as instance of descriptions.

2.10 Zero Emission Neighborhoods (ZEN)

Part of building sustainable cities with a wise management of natural resources, involves the living and the building domain of the city. Buildings are a significant source of energy consumption, with it being responsible for nearly 40% of the final energy consumption as well as 36% of a greenhouse gas emissions in both the private and public sector [63].
A zero energy building is defined as building that supplies at least as much energy as it consumes by supplying energy to the same grid that it consumes energy from [115]. In the UK there is work towards realizing zero carbon homes that contribute zero carbon emissions over their life-cycle [92]. Zero emissions neighborhoods (ZEN) have the same focus on greenhouse gas emissions and low carbon, but instead on the scale of neighborhoods where energy flexible neighborhoods is part of realizing that goal. To understand the scope which ZEN operates within in the context of neighborhoods, their definition of a neighborhood is useful [155]:

> ”In the ZEN Research Centre, a neighbourhood is defined as a group of interconnected buildings (which can be of different types, e.g. new, existing, retrofitted, or a combination) with associated infrastructure (which includes grids and technologies for supply, generation, storage, and export of electricity and heat, and may also include grids and technologies for water, sewage, waste, mobility, and ICT), located within a confined geographical area. The area has a defined physical boundary to external grids (electricity and heat, and if included, water, sewage, waste, mobility, and ICT). However, the system boundary for analysis of energy facilities serving the neighbourhood is not necessarily the same as the geographical area. The system boundary for each ZEN pilot area is also dependent on the case and may vary accordingly.”

According to [73], the ZEN research centre has an aim to answer the following research question:

> How should the sustainable neighbourhoods of the future be designed, built, transformed, and managed to reduce their greenhouse gas (GHG) emissions towards zero?

In order to achieve this, the same document report states that the focus should be the following goals within the neighborhoods:

- Plan, design and operate buildings and their associated infrastructure components towards minimized life cycle GHG emissions.
- Become highly energy efficient and powered by a high share of new renewable energy.
- Manage energy flows (within and between buildings) and exchanges with the surrounding energy system in a flexible way
- Promote sustainable transport patterns and smart mobility systems.
- Plan, design and operate with respect to economic sustainability, by minimising total life cycle costs.
- Plan and locate amenities in the neighbourhood to provide good spatial qualities and stimulate sustainable behaviour.
- Development of the area is characterised by innovative processes based on new forms of cooperation between the involved partners leading to innovative solutions.

The work of ZEN research centre is divided into 6 different work packages that focuses on different areas of research, but are all connected to each other. Most relevant for this thesis is work package 1, which is related to the specification of the ZEN KPIs, further explained in section 2.10.2, and the data management plan for collecting, structuring, and analyzing the ZEN KPI data, which involves the ZEN ICT architecture, further explained in section 2.10.3. And work package 6, which involves the ZEN pilot project and the living labs, further explained in section 2.10.1.
The ZEN research centre is part of a research centre for environmentally friendly energy called ”FME” which does the research on zero emission neighborhoods. And is research done in collaberation with partners in the private sector and the public sector, where NTNU acts as the hosts and leads the centre together with SINTEF Building and Infrastructure and SINTEF Energy.

2.10.1 ZEN pilot

The ZEN pilot projects is part of work package 6, and consist of 9 different locations in Norway where new solutions for the construction, operation and use of buildings are tested. The intention of the ZEN pilot projects is to work as role models to other, by inspiring them to build zero emission neighborhoods and showing how this can be done [155]. The ZEN pilot projects are located at these locations in Norway:

- Airport redevelopment, Bod
- Campus Evenstad
- Campus Evenstad
- Fornebu, Brum
- Furuset, Oslo
- Knowledge Axis Trondheim
- Residential area, Steinkjer
- Ydalir, Elverum
- Zero Village Bergen

The project covers different phases, and involves new development and existing neighborhoods as part of the projects. The phases ranges from the strategic long-term planning, where goals and ambitions for the development are made, and measures are made to fulfil goals and ambitions. To the tactical, mid-term planning and implementation phase, where zoning plans and development agreements are made in order to specify the goals made in the previous phase, and smaller projects might be part of this phase. And the last phase that is the operational phase, where the neighborhood or parts of it is in use, research is being done. The projects differ in which stage they are in, in addition to also have different functionalities, like being a school, and office building or a residential home. They might differ in size, areas, local context and the time frames for the projects. There is also different thematic scopes, where ZEN might look purely at energy use in project, to a more holistic view in others [32]. And as part of testing and analyzing these solution, data management of the gathered data from the project is needed, which is where ZEN ICT architecture is meant as a solution.

2.10.2 ZEN KPI

As part of the testing and analysis of the new solutions in the pilot projects, there is a need for a clearly defined set of assessment criteria and KPIs, that can be used to create methods and tools to assess the progress of ZEN pilot project in terms of reaching their stated goals. ZEN defines a KPI as ”a set of quantifiable performance measurement that define sets of values based on measure data from a project, making it ensures to measure and track the neighbourhood’s performance over time and against other similar projects” [149].
The scope of the ZEN definitions includes seven categories where each category might have one or more assessment criteria, and each of the assessment criteria might have one or more KPIs [73]:

- Greenhouse gas emissions
  - Total greenhouse gas emissions

  - Greenhouse gas emission reduction
- Energy
  - Energy efficiency in buildings
    - Net energy need
    - Gross energy need
    - Total energy need
  - Per energy carrier
    - Energy use
    - Energy generations
    - Delivered energy
    - Exported energy
    - Self consumption
    - Self generation
    - Color coded carpet plot
- Power/load
  - Power/load performance
    - Yearly net load profile
    - Net load duration curve
    - Peak load
    - Peak export
    - Utilization factor
  - Power/load flexibility
    - Daily net load profile
- Mobility
  - Mode of transport
  - Access to public transport
- Economy
  - Lice cycle cost
- Spatial qualities
  - Demographic needs and consultation plan
  - Delivery and proximity to amenities
  - Public space
- Innovation

ZEN Toolbox Guideline

In addition to the development of the ZEN KPIs, there has also a process of developing a toolbox that makes use of the different KPIs, by feeding the different KPIs into the tools [156]. These are multi-criteria decision analysis tools that look at the overall evaluation of trade-offs and performances for the ZEN KPIs for a pilot project, as shown in figure 2.17. As example for tool, might be visualization of overall performance for a project. These

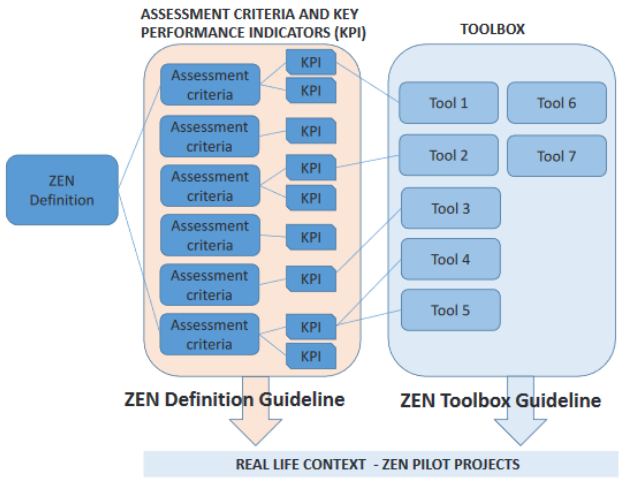


Figure 2.17: The relationship between ZEN KPIs and the ZEN Toolbox [156]

tools have a need to also find the best data available for their use stored in the ZEN ICT architecture, and might be helped by the work done in this thesis to find and select data.

2.10.3 ZEN ICT architecture

In papers [126, 127], the ZEN ICT architecture is proposed and is also shown in Figure 2.19. It is intended for big data management in a smart city scenario in order to manage all the data created in a smart city, and also as an architecture for managing the data from the ZEN pilot projects.
It consists of three different layers, where all layers manage data, through a model called the Comprehensive Scenario Agnostic Data Life Cycle (COSA-DLC), described in [124] and shown in figure 2.18. The three blocks of the model, data acquisition block, data processing block, and the data preservation block details organization of the data management at each layer in the overall architecture. Each block shows the different phases of data management within it, as shown in the figure. Part of the process includes in addition to collection, processing and storing, involves the classification of the different attributes of the data during this process.
The architecture as a whole is shown in figure 2.19, where the architecture consists of three layers, where the two lowest are distributed, and the top cloud layer is the centralized layer. As the layers get closer to the cloud, the computational capacity and storage capacity grows, but so does the latency for the users near the edge that need the data, and as a result, the closer you get to the cloud, the older the data gets as well.

- **Fog-Layer-1.** The lowest layer of the architecture is Fog-Layer-1. It is the layer nearest the end-users of the where the data is collected within the cities. At the top of the layer represented as green dots in the figure, is the fog-devices that organize and

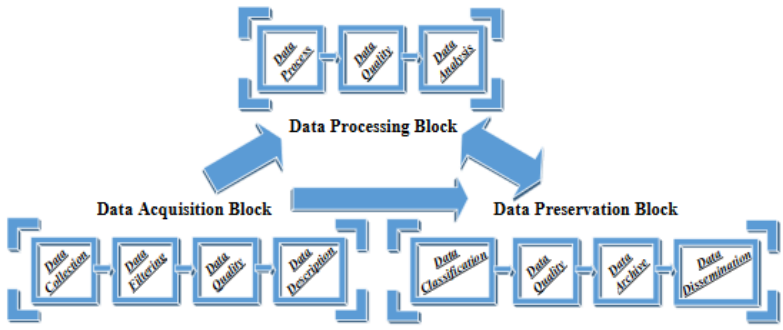


Figure 2.18: The COSA-DLC model

control the IoT devices that collect the data. This is most likely the most powerful node in terms of processing and storage. These IoT-sources operate within a Fog-Area, which in the case of ZEN is the ZEN Pilot neighborhoods. This layer is responsible for managing real-time data.

- **Fog-Layer-2.** Fog-Layer-2 is a middle layer between the between the IoT-Hub and Fog-Layer-1, which is positioned in the city but not in physical proximity to the IoT devices. The IoT-Hub is the most powerful node in terms of processing and storage. At the top of the architecture lies the cloud layer. It will store the last-recent data that is not as fresh as the real-time, but not historic like in the cloud.

- **Cloud Layer.** The cloud layer has the most processing and storage, and will store all historic data from the ZEN pilots. This forms the basis for the data management in the ZEN pilots, and it is within this context a cost model for service selection is to be proposed in the next section.

It is within this architecture that data from ZEN will be managed, and as explained earlier, the problem looked at in this paper is how to find suitable data for the users within this system of data management. The system will be responsible for gather data, processing it and storing it in various locations.

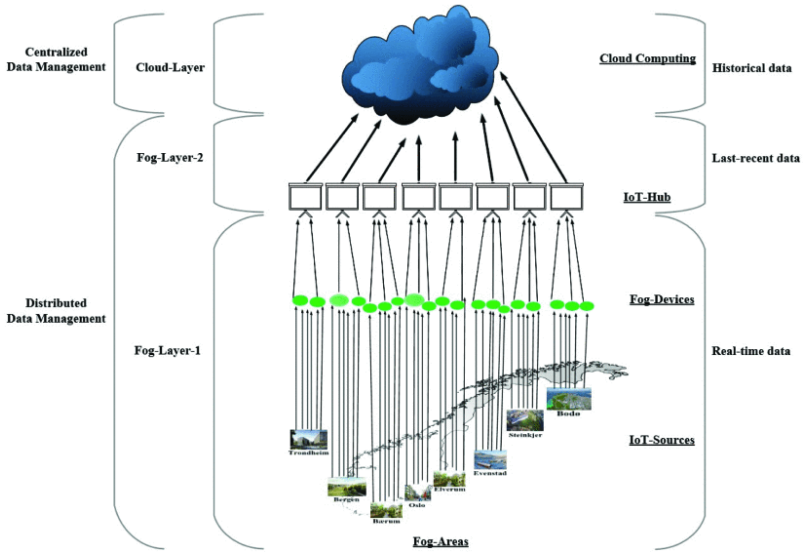


Figure 2.19: ZEN ICT architecture

Chapter 3

Research Methodology

3.1 Research Method

This master thesis is part of ZEN research on distributed-to-centralized data management architectures for smart cities [156]. The focus is as mentioned earlier in the problem description on how to find and select suitable data and services in this context. The problem is in itself just one of many problems that needs to be solved in this context, and different problems are also being worked on by two other master students at the same time period as this thesis, which instead focuses on addressing and routing, and allocation of processing resources. It is also a continuation on the work that was done on an older master thesis with data management in a centralized model in [158]. This is also part of a larger body of work on smart city data management, and distributed-to-centralized architectures or fog-to-cloud architectures.

The research consists of two phases, the first is a literature review to build the background theory of this thesis, which answers some of the research questions. The second phase is based on the design research methodology, which aims to design and implemented an artifact, which then is evaluated, as part of the research.

The literature review done in preparation to this thesis is based several topics related to the problem and research questions as well as the work that is done in this area, based on material given to me by my supervisor and expanded upon by finding related literature by using terms from those papers. It consists of material about smart cities and distributed-to-centralized architecture, KPIs in cloud computing and for decentralized-to-centralized architecture, data quality, cost models, service selection and service management, which are all relevant for the design of a solution.

The design research method in this paper is based upon the design research cycles proposed in [58]. In it, three research cycles in the overall design research cycles is specified. The first is the relevance cycle, which initiates the research by identifying opportunities and problems in an application environment, and often ends with an evaluation of the impact of the research in that application environment. The rigor cycle ties the design

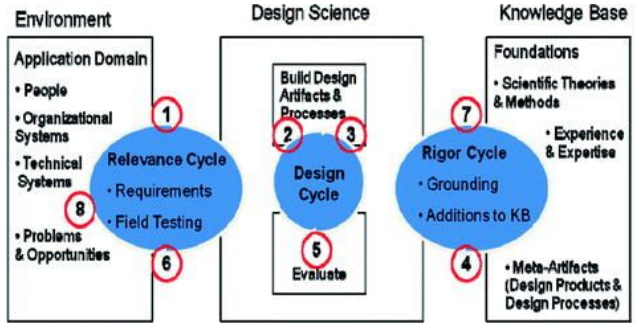


Figure 3.1: How the design science research checklist maps to three design research cycles [58]

research into the scientific knowledge base of the related fields, grounding the design in scientific concepts and also contributing to the knowledge base through the design research. The design cycle is at the center of the design research cycles, and consists of designing, building and evaluating artifacts and process informed by the other two cycle. From this, a checklist of 8 questions is proposed for evaluating a design research project:

1. What is the research question (design requirements)?
2. What is the artifact? How is the artifact represented?
3. What design process (search heuristics) will be used to build the artifact?
4. How are the artifact and the processes grounded by the knowledge base? What, if any, theories support the artifact design and the design process?
5. What evaluations are performed during the internal design cycles? What design improvements are identified during each design cycle?
6. How is the artifact introduced into the application environment and how is it field tested? What metrics are used to demonstrate artifact utility and improvement over previous artifacts?
7. What new knowledge is added to the knowledge base and in what form (e.g., peer-reviewed literature, meta-artifacts, new theory, new method)?
8. Has the research question been satisfactorily addressed?

How these questions maps to both the design research, the environment, and the exiting knowledge base is shown in figure 3.1. With the numbers with red circles around the map to the different parts of the three design research cycles.
For each of the questions, these are the answers or where the answers could be found:
1. The research questions are presented in section 3.2, and are based on the problem description and motivation in section 1.2.
2. The artifact is the design presented in chapter 4 and implemented in chapter 5.

3. The design process is based on [137], where the process consists of suggesting a design as a solution to the problem identified in research questions and in the problem description.
4. The design itself is formed and based on the theory found in the literature review, how the design fits with the theory is explained in chapter 4.
5. There is only one cycle for the design cycle, which means the evaluation will be found in the discussion in section 7.2
6. The testing of the artifact is explained in the evaluation and part of the results in chapter 6. Besides looking at the whether it fulfills the requirements defined in the design, measurements of response times is also used to evaluate the artifact.
7. Identification of KPIs to use for selection in this context, the design of the artifact together with the evaluation of how it works in a simulated application context.
8. The research questions are answered in the conclusion, in section 8.2.

Strength and weaknesses

By not performing a systematic literature review, but what is referred to in [158] as an ad hoc literature review, it makes it harder to reproduce the result from the literature review and how the different literature was found. But since it was identified that the solutions to the problems required literature from a wide variety of different domains, it would be hard find search terms and approach that would cover this wide area and at the same not produce an overwhelming amount of to be reviewed. The search terms use, related to the selection, KPIs, storage and description, together with fog-to-cloud and cloud to find relevant literature. A mistake that was done, was to not write down the terms used, which in hindsight is a mistake.
The design science research allows us to test the feasibility of an idea of an solution to a problem, but is also dependent on good enough evaluation and testing. If the environment it is tested and evaluated in does not correctly correspond the environment is intended to work in. In our case, with an underlying architecture that is not implemented as the intended environment, the conclusions drawn in the evaluation might not translate.

3.2 Research Questions

- RQ1: How can a cost model be defined for service and data selection in a distributed-to-centralized architecture for smart cities?
  - RQ1.1: Which KPI and criteria should be used with the cost model in this context?
  - RQ1.2: What approaches and methods should be used to define a cost model in this context?
  - RQ1.3: How should information about services be stored and shared in this context?
- RQ2: How can an implementation of this cost model contribute to finding data in the ZEN ICT architecture?

Chapter 4

Proposed cost model design

This chapter contains the proposed design and architecture for the cost model and its supporting components. The design is based on background material presented in chapter 2, which will be referred to in the design description. The goal of the design is to help find data and services within a distributed-to-centralized data management architecture for smart cities, with the ZEN ICT architecture as the mold for that type of architecture. The goal of the design is to find data within the stated context, but also be general enough that it can translate to finding services as well as data for similar architectures. First in this chapter, the different stakeholders are identified, before some requirements based on earlier work, and finally the design of the architecture itself.

4.1 Stakeholders

The different pilot projects have different stakeholders, identified in [32]:

- Project owners: In most cases the municipality is the project owners as well, but could also be private developer and land owners.
- Municipality and government: Can be different departments and administrations
- Universities
- Landowners
- Developers
- Consulting agencies
- Transportation agencies
- Energy utility companies
- Waste management companies
- Housing cooperatives

But not all of these can be considered important stakeholders in this design. The stakeholder is in the case the people and organizations that either use the data or generate it in the ZEN pilot projects, but also in the smart city scenario. Based on this list, a short list of stakeholders is presented below:

Citizens

To some degree the subjects of the data, and have a stake in how the data generated by them and their buildings and neighborhoods, but also as potential consumers of their own data through applications and services.

Municipality and government

Consumers of data, that can use the data as part of decision making support, through tools that help analyze and visualize different aspects of the city. But also for creating policies at the highest level of government as well.

Researchers

The pilot projects are themselves part of the research, where the researchers involved have a vested interest in the data generated. In large scale scenarios, the data generated is also something that researchers and scientists should find interesting.

Utility companies

Providers of different utilities like energy, waste management, transportation and water as examples, are like going to be interested in the data produced to improve their services.

Application developers

The developers creating services based on the data from the smart city is going to have an interest in a system the is responsible for managing the data they depend on.

4.2 Requirements

This section contains the requirements for the design. It also contains information of the requirements specified for the previous work done on this topic, and which parts of the requirements remains for the cost model design. The requirements are also to be used in the evaluation of the implementation.

4.2.1 Previous requirements

This project is a continuation on the work on data management for ZEN. In an earlier thesis, a data management architecture using a centralized approach was designed and implemented, and used a set of requirements identified by ZEN as a general set of requirement for a data management platform [158]:

- Hosting of KPI, research and context data and metadata
- Big data and visualisation services
- End user applications for municipalities, citizens and businesses
- Logs and data on the use of services provided
- Open API for third party developers

- Support for exploratory analysis of datasets
- Support for management of KPI data
- Support for research based on the data on the platform
- Future-proof for the duration of the ZEN pilot projects at a minimum 8 years
- Flexibility in order to adapt to changes in KPIs and collected data

From this, the thesis identified six different requirements:

- Flexible processing and analytics
- Interoperable storage for heterogenous data
- Cohesive data management
- General approach
- Facilitation of open data dissemination
- Vendor independent

With the new ZEN ICT architecture, which a is distributed-to-centralized approach to data management, detailed in papers [126, 127] and discussed in section 2.10.3, the requirement are specified in terms of meeting the big data challenges(the 6 Vs) and data life cycle specified in terms of the COSA-DLC [124]. This thesis is however smaller in scope than the whole data management task, and is instead focused on finding and selecting data, which means several of the requirements are not useful for this case, and are deal with in other places in the distributed-to-centralized architecture.

From the first set of requirements formulated, ”flexibility in order to adapt to changes in KPIs and collected data”, is relevant, as the description of the data and services, and changes in the KPIs used to select can change, which demands flexibility for this solution as well. From second set of requirements, ”general approach”, in that it should support finding and selecting different kind of data and services, which ties into the flexibility requirement. ”Facilitation of open data dissemination”, it should make it easy to access and explore data for everyone. ”Vendor independence” is also relevant for this solution.

From the 6Vs, the variety of data must be supported, and is related to the flexibility of and general approach requirements. The velocity of the data, as it is quickly created, must be attainable as soon as possible for the end user. The value of the data is at the core of this thesis, attempting to find the most suitable data for the least cost. From the COSA-DLC model, the responsibility of following the model lies on the actors storing the data, in classifying the data, appraising the quality of the data and readying it for dissemination. The classification and the quality appraisal information can then be used in selecting among the data.

4.2.2 Usage scenario

The specific design is designed towards use with the ZEN ICT architecture and the ZEN pilot project. In this scenario, different actors may want find specific ZEN KPI data. One of the most basic specification of the data, is the type of data, the location where the data is from, and the time period when the data was created. In addition to that, in order to find the most suitable data, the actor should be able to specify different criteria with the data, to get the most suitable place to get the data from.

4.2.3 Specified requirements

From the previous requirements, and the usage scenario, these are the requirements for the design:

- Be able to find data according type, location, and time generated.
- Be able to specify preferred qualities with data or the provider of the data, to get the most suitable data.
- Be flexible with which types of KPIs can be used to describe a data provider or the data.
- Be able to find data as soon as it is accessible.
- Facilitate open data dissemination.
- Vendor independence.

4.3 Architecture

The architecture is based on the underlying ZEN ICT architecture, which is a distributed-to-centralized architecture for data management, and provides the context this architecture is designed in. It is itself a distributed design, where different instances of what in this case is called ”Control units” are hosted in Fog-Layer-2 and in the Cloud-Layer and are able to communicate with each other. The idea behind the design is that you should be able to query any control unit, and get a response, but being in a city where a Fog-layer-2 control unit is hosted and trying find data from that city should be better. In this case, the control unit is kept separate from the data repositories, but could just as easily be implemented as a part of the data storage. In the scenario outlined for the ZEN, there might be different actors within the same city, like an energy provider, a municipal entity or a research team that themselves keep data from the city that might be accessed. The architecture is shown in figure 4.1, with two control units in Fog-Layer-2, representing Trondheim and Bergen, and a control unit is the cloud. The architecture supports adding more units in Fog-Layer-2, but for simplicity, only two are included in the design. The solid arrows represents the communication between control units, while the stippled arrow represent communication between control units and data repositories.

The control unit consist of three major components. The first being a request handler that handles the incoming queries, from either a user or another control unit, and is responsible for forwarding the request to the other components and responding to the query once the other two components are done. The second component is the ”cost model”, which is responsible for filtering and calculating a ranking between all viable alternatives that is found, based on the criteria given by the user in the query. It can either return the ranked results or just the highest ranked alternative to the user. The last component is the ”router”, which depending on the on some of the search parameters, queries the location that might hold the type of data for metadata about the data and their service properties. In addition to the control unit, a GUI is designed as the point where the user interact with the system, consisting of a configuration component to specify a criteria for the search, as well as search component that search for specific data and displays the results. In figure 4.2, the internal architecture and how the different components interacts, is shown, where

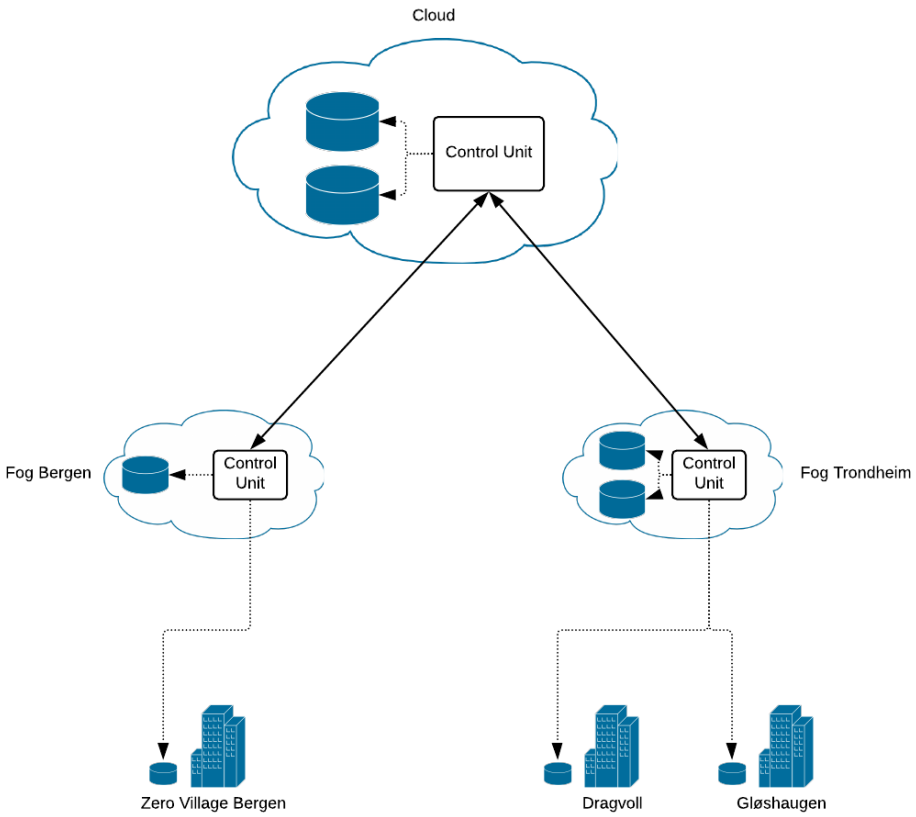


Figure 4.1: The top level of the architecture of the cost model design

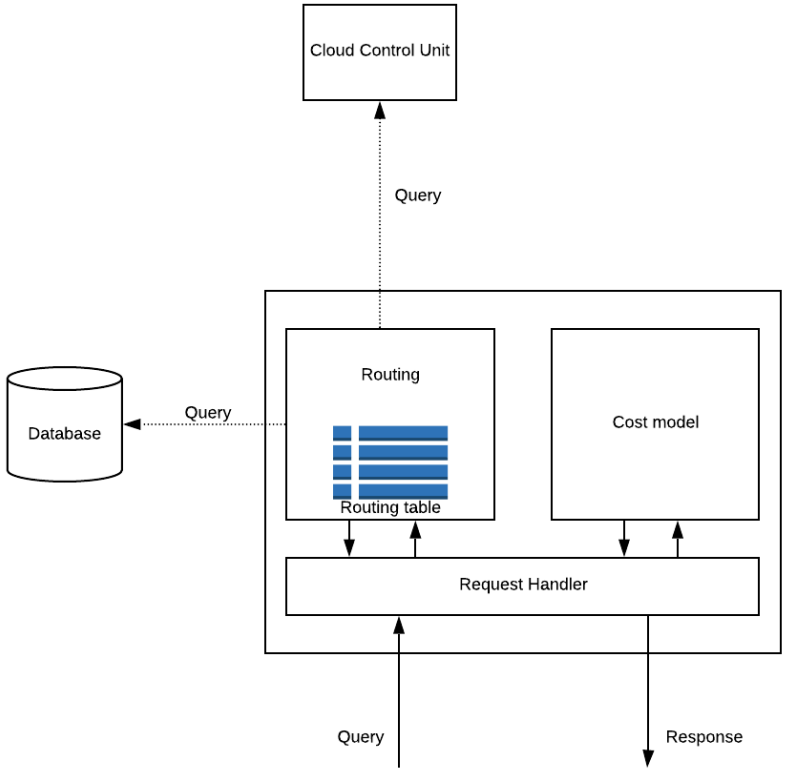


Figure 4.2: Internal architecture of a control unit

solid arrows show internal control flow of the control unit and request and response to the control unit, and the stippled arrows show the queries that the routing component might make. How the control flow is from a query to the control unit, to its response, is shown in the sequence diagram in figure 4.3.

4.3.1 Request handler

The request handler acts as the interface to the control unit. There are two types of communications that would access the request handler, the first being user requests in the form of a search query together with a description of the cost model parameters, and returns a ranked list of all the suitable alternatives. The other is a request from another control unit, that needs information about data repositories that the control unit knows about, in the form of a search query without a cost model configuration, and returns all results that match the query. The search query consists of the location where the data was created, the time it was created, and the type of data it is, like temperature measurements or power usage. The cost model parameters are explained in section 4.3.3.

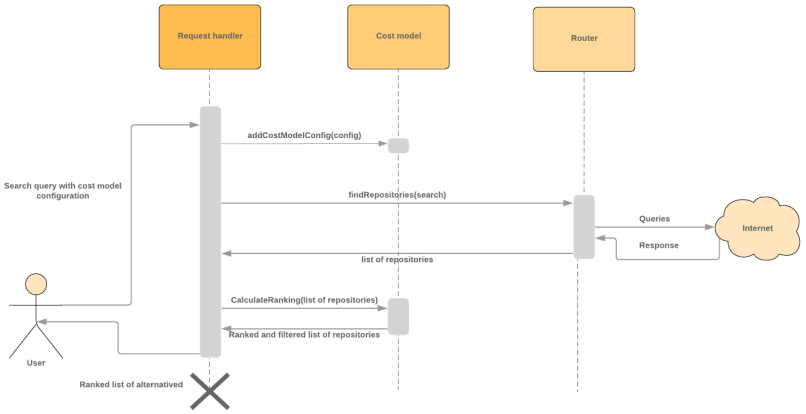


Figure 4.3: Control flow for a user query

4.3.2 Routing

The routing component is responsible for querying all the places that might have the data the user is looking for, specified by the search query sent to the request handler and passed to the routing. It consists of a routing table and a set of query algorithms. The idea behind this approach is based on the distributed hash tables containing information on where to find data or files presented in section 2.8.2. The approach allows the different nodes of the system to hold partial information about the all data stored in the system and knowing where to forward the queries if it does not hold that information itself. This means that a control unit in the fog only needs to hold information about the data stored for the fog area, limiting the amount of information kept and managed by it, and that communication can go to control unit placed locally, instead of through the cloud control unit, lowering response times. Any query to it for data from another fog area is forwarded to the cloud control unit, that will forward it to the correct fog.

The approach used in this design differs from the ones presented in section 2.8.2, in that it uses the underlying tree structure of the ZEN ICT architecture presented in section 2.10.1, and is likely very static as it is tied to the underlying infrastructure of buildings. The biggest difference from the presented approaches in section 2.8.2, is that it does not only attempt to reach a single node, but every node that might have the appropriate data, making it like a hybrid approach between the flooding used in Gnutella [26], where a query is flooded through the overlay network with some limited scope, but also keeping routing tables to avoid forwarding it to nodes that does not have the data.

The reason behind this approach, is that we to some can assume that data generated in a building can only be found at certain places, at the Fog-Layer-1 near where it was generated as real-time data, at Fog-Layer-2 as last recent data kept in the IoT-Hub or by other actors in the same city, or in the cloud as historical data. This could also be extended to be done together with type and time, as part of the routing entries, but is not done for this implementation, focusing solely on location.

Entry	Repository	Queried
”/”	Cloud	Yes
”/bergen/”	Fog-Layer 2 in Bergen	Yes
”/trondheim/”	Fog-Layer 2 in Trondheim	No
”/bergen/zeroVillage”	Fog-Layer 1 in Bergen	Yes

Table 4.1: How a query with location ”/bergen/” would query repositories from a control unit

Routing table

The routing component needs to hold its own table, that is used to query different locations based on the search query given. For a given location sent in a search query, it makes sense the routing table has enough information to know which places to query for data, this could also be done for the type, if the routing table have knowledge of which locations hold which type of data, and even for time, if it knows that some data age means that it could only be with the historic data as an example. Based on this, information from the different location is queried. The approach for describing data location is based upon the REST [44], for location only. A location for data might be described as ”/bergen/zeroVillage/2/4”, where the first parameter is the city, the next the neighborhood, followed by building number and sensor number. A repository in the fog in Bergen might hold data that begins with ”/bergen/”, which might contain any data generated in Bergen. That means that when looking for data specifically from ”/bergen/zeroVillage/2/4”, we would want to query that repository. ”/bergen/” is then the key for a table entry in routing table for that repository, and if the key can be a prefix for the location in the query, we use the associated IP-address to query that repository to find out if it holds data from ”/bergen/zeroVillage/2/4”.
Similarly, we also want to be able to query more than just a specific sensor, but data from several locations. In a case where we would want data from Bergen as a whole, we would search for ”/bergen/”, and if there exists repositories for data from a single neighborhood with an entry longer than ”/bergen/”, like ”/bergen/zeroVillage/”, we would also want to query this location as it holds data from Bergen. To do this, we check if the query can be a prefix to the routing table entry, we query the associated IP-address.
We then end up with the logic of querying any entry in the table, where the routing table entry can be a prefix of the query, or the query can be a prefix of the routing table entry. In other words, if we know that a repository does not contain the data we are looking for, we do not query it. As en example, the table 4.1 shows how the location part of a query ”/bergen/” interact with the routing table, with blue color meaning a prefix match, and the red color meaning a prefix mismatch.

Query algorithms

Using the information from the routing table, the query algorithms are responsible for querying the different locations that match the search query. As these are multiple http requests, there is a possibility that some of the requests might fail or timeout, and after the queries is done, the control unit still needs to respond before the initial request that was sent to the control unit times out. This means that there needs to be window where

responses are collected, returning those that succeed. On the other hand, if all requests succeeds or fails quickly, it should return as fast as possible. As this process is can be forwarded, there might be a need to decrease the waiting time on each following query, which might be a working approach as the tree structure is limited in depth. These request can be directly to a repository, or to another control unit, that returns the collection of data descriptions it found.

4.3.3 Cost model

The cost model designed for this is based on that MAUT approach to selection, discussed in section 2.6.7. The reason behind using this is that it a fairly intuitive approach that allows to create fairly detailed ranking function depending on the amount of intervals done, and being a utility based comparison makes it easier to target specific performance values than by using comparison to relative performance of all alternatives. It is not very computationally demanding, as it only need one iteration through all alternatives to find the best score. It also seemed easy to make an intuitive interface for as well. But as it does not have strict constraints in it, filtering results on strict constraints will be added as part ranking and selection. This is something that should give a variety of ways for a user to specify their needs and wants for the results. So, the cost model configuration consists of a set of hard constraints that filters out alternatives based on some KPIs, and a utility functions in the form of utility intervals, as presented in section 2.6.7, and their weights. The final result set consists of all the alternatives given scores. A pseudo code implementation of the cost model is presented in algorithm 2. After gathering all the alternatives in $alternatives$, it iterates through them. For each alternative a, it first iterates through the strict constraints in $filter$, if the alternative breaks a constraint, it sets add to false, and it is not added to the final results. If it not filtered out, it iterates through the criteria for the ranking, calculating a score with the weight and utility function in each criteria c, adding it to the total score, before it is appended to the list of $results$ together with its score.

```
filters: a list of strict constraints;
critera: a list of criteria to rank with;
alternatives: a list of alternatives;
result = [];
for a in alternatives do
    add = true;
    for f in filters do
        if a breaks constraint f then
            add = false;
        end
    end
    if add == true then
        score = 0;
        for c in criteria do
            score = score + (c ranking for a)
        end
        result.append({score: score, data: a})
    end
end
```

Algorithm 2: The cost model calculation

This is not an adaptive solution, as the result is calculated for a single query, based on KPI data either queried from the data repositories, or measured, and given to the user requesting it. It does not include an component for measuring the changing performance of a chosen data repository, which likely should placed at the user utilizing the data repository. This could perhaps be updated to the design by implementing the ACPS algorithm presented in section 2.6.9 with the user.

4.3.4 User interface

The most basic user interface might be through the API provided by the control unit, which takes the form of a JSON data in a HTTP POST request, with the search query and the cost model configuration. But a design is also made for a graphical user interface (GUI), to see how intuitively it can be made for users. The design consists of two pages, one where the search terms can be entered and that displays the results, and another page for specifying the cost model configuration so that it could be used for multiple search. The search pages specifies search queries in terms of data type, data location, and data age, while the cost model configuration uses different KPIs to filter and rank alternatives.

Chapter 5

ZEN Cost Model Prototype

This chapter introduces the prototype created based on the design presented in chapter 4. The creation of the prototype is part of the research methodology presented in chapter 3, where the part of the methodology includes the creation of an artifact to evaluate it as part of answering the research questions, specifically RQ2. The prototype includes all the components presented in the design, in addition to mocked data repositories to test the prototype.
The prototype is limited in scope in terms of the KPIs included in the cost model, and is using KPIs given by the mocked data repositories instead of measuring some of the data itself.

5.1 Preliminaries

5.1.1 Requirements and implementation

The requirements identified in section 4.2.3 needs to be taken into consideration in the implementation of the prototype.

1. Be able to find data according type, location, and time generated.
2. Be able to specify preferred qualities with data or the provider of the data, to get the most suitable data.
3. Be flexible with which types of KPIs can be used to describe a data provider or the data.
4. Be able to find data as soon as it is accessible.
5. Facilitate open data dissemination.
6. Vendor independence.

Most of these are taken into consideration in the design, and by implementing the design, will be satisfied, this will include the requirement 1, 2, 4, 5, 6. But the flexibility needed to satisfy requirement 3 is dependent on how the implementation of the design makes it

easy to change or add KPIs to the system. It is necessary to design the scheme by how the KPIs are handled and structured in a way that makes it flexible.

5.1.2 Scope and limitations

The scope of this prototype is in many way formed by the limitations. The largest limitation is that the underlying ZEN ICT architecture only exists as an architecture design, which means that in order to test and run the prototype in the context, some aspects of the architecture needs to be mocked. This is done through mocking data repositories. There is also some limitations to what data exists, and in what format it is. There exists data sets collected at some of the pilot locations, but not as being processed and managed through the decentralized-to-central data management architecture. But the data itself is not what is of interest when selecting, but the description or metadata of both the data and the data repository that is used in the cost model for selection. This means that a set of metadata describing data sets is made up for the sake of testing the prototype, and specifying the format of KPI descriptions for a data repository. The KPIs selected for this is further explained in section 5.2.
The scope of the prototype involves implementing the design specified with a set of few KPIs, and simple utility functions based on numerical intervals, as shown in the example of MAUT in section 2.6.7. It is hosted locally, and on several location in the cloud, that all communicate with each other and can be accessed through the implementation of GUI that runs in a web browser. The mocked data repositories are small web servers the responds to a search query of the same format as used for the control units, with a list of metadata for data sets that match the search query.

5.2 KPI and data description

The implementation of the format of both search queries, and the description of how KPI and data descriptions from the mocked data repositories is formatted is described in this section. As Nodejs and javascript is used, it makes sense to also use the JSON format for the descriptions, instead of an XML format. We can divide the descriptions into two parts, the search query descriptions and the response detailing the data.

Search query

The search query consists of three parts, the first part specifies the difference between a control unit query and a user quest, which does not include a cost model for the control unit query. The second part is the data that is being searched for, and the third part is the cost model configuration. Below is the format shown, implemented in json, with ”<” and ”>” around the input:

```
{
    "query": "user",
    "search": {
        "type": "<data type>",
        "location": "<location string>",
```

```
        "start": "<data after this date string>",
        "end": "<data before this date string>",
    },
    "criteria": {
        "criteriaList": "<list of different KPI for cost model>",
        "intervalList": "<list of utility intervals>"
    }
}
```

Response

Only a subset of the different KPIs is included in the implementation of the model. Mostly because adding every KPI to the implementation would not give any benefit in testing the implementation. The limitations with using mocked data repositories means that this information that is created for the purpose of testing. Part of that simplification has been to treat the data as sets with start and end times of recording. In reality, data can be represented in other manners as well. Those chosen KPIs are ones that have been found to be relevant for the task selecting data. The different KPIs is shown in the response format below, some tied to the data repository and some to the data itself:

```
{
    "score": "<ranking of the data>",
    "server": {
        "address": "<path to the server>",
        "bandwidth": "<bandwidth>",
        "resourceCost": "<cost for resources used"
    },
    "data": {
        "key": "<some key to access the data>",
        "timeStart": "<start date for data set>",
        "timeEnd": "<end date for data set>",
        "cost": "<access cost for data set>",
        "location": "<exact location of origin>",
        "type": "<data type>",
        "samplingRate": "<granularity of data>",
        "dataSize": "<storage size of data>",
        "accuracy": "<accuracy of the data>"
    }
}
```

This forms the API to interact with control units through HTTP POST messages, and could allow other applications to interact with the control units besides the GUI created.

To address requirement 3, this should be a very flexible manner to implement the description of data and service, as you can just add a new KPI to the server description or the data description, and if it is added to the list criteria in the query, it can be used as part of the cost model, if the KPI is a numerical value. There is some lack in the flexibility, as there is not support for non-numerical KPIs in the cost model.

5.3 Technologies

5.3.1 Programming language and run-time environment

The implementation technologies are all built upon Node.js[1], which involves both the control unit and the GUI, and the decision for programming language and platform was based mostly upon the fact that I am most comfortable with my knowledge in the JavaScript and TypeScript[2] programming language and Node.js, together with express.js framework to create web servers. By using Node.js you also have to use JavaScript, or a language like Typescript that compiles into JavaScript. This could have likely been done in any other web framework for different languages and run-time environments.

5.3.2 Frameworks

Express.js

I used the Express.js[3] web framework to create the server. This was done for the same reason as mentioned above, that it was web framework that I was familiar with, but could also likely be replaced by most other web server frameworks. It is a simple way to quickly create a web server that handles and returns http requests.

React

For the GUI, I used React[4] to quickly create a graphical user interface to specify a configuration for the cost model and to issue requests to the control unit. The reason behind using react is also the fact that I have most experience with user interfaces through web browsers created by html, css and javascript, and react seemed like a simple way to quickly create that. Any other platform for creating an UI that could also send http requests would be equally good.

5.3.3 Cloud computing

As part running the application up on the cloud, and at different places in the cloud was also needed, which made it natural to test several of the cloud providers. The selection of cloud providers was based on being some of the larger ones, and providing free cloud hosting. For this I used virtual machines on Microsoft Azure and at Google Cloud. The virtual machines in both instances ran Ubuntu 18.04 as an operating system, but that does not matter as Node.js runs cross-platform on most common operating system. Using an a cloud platform was important as this is part of the ZEN ICT architecture.

[1] Node.js - https://nodejs.org/en/
[2] TypeScript - https://www.typescriptlang.org/
[3] Express.js - https://expressjs.com/
[4] React - https://reactjs.org/

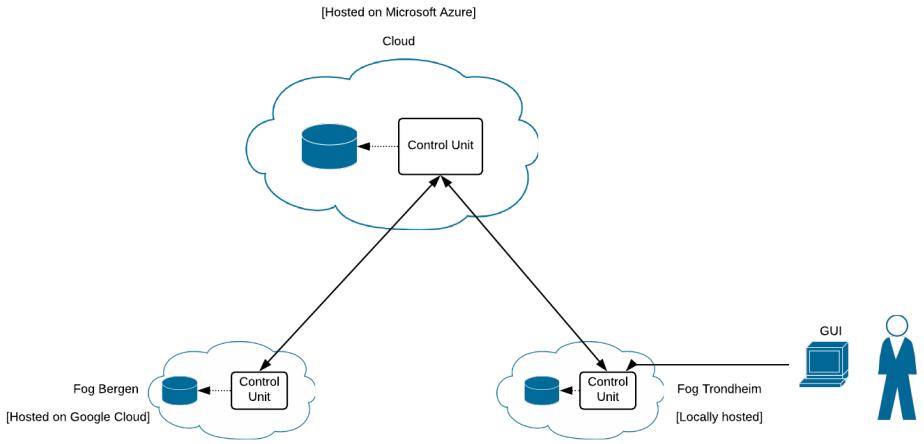


Figure 5.1: The implementation of the architecture mapped to hosting locations

5.4 Implementation architecture

It is important for the implementation to be somewhat close to the design in order to test it. As mentioned in the limitations, much of the underlying infrastructure does not exist in implemented form, but by hosting the control units locally, and on different cloud location, the testing and evaluation can be done in a context that is somewhat close to the intended one.

The final product consist of the control unit, the mocked data repository and the web browser based GUI. To run different scenarios, which will be explained more of in section 6.2.1, and test and evaluate in the implementation, it is attempted to implemented like it would be in the ZEN ICT architecture.

Figure 5.1 shows the architecture of the implementation. The Trondheim fog control unit is hosted locally on my own desktop computer, together with the mocked data repository. The GUI is running on a laptop locally that can access the desktop through the local area network, and the control units that are host on two different cloud locations. The cloud control unit is hosted on Microsoft Azure's virtual machines, together with another virtual machine that holds the mocked data repository. This is located on the North Europe region of of Microsoft Azure's regions, which is located in Ireland. The fog control unit for what is supposed Bergen in the scenario is located on Google Cloud's virtual machines in Finland, as this was the closest cloud computing provider to Norway that I found. It is hosted on a virtual machine there, and another virtual machine in the same location is used for the mocked data repository. The implemented control units have configuration files, that is used to set the routing tables such that they would communication as shown in the figure, when appropriate queries are received. The internal components of the control units is implemented as described in the architecture design in section 4.3, and more specifics about how it was implemented is discussed in the next section.

5.5 Development

This section contains information about how the different components was implemented, and the technologies used, which are almost the same for all the different components, all are based on TypeScript/JavaScript and node, but two uses the Expressjs framework, and one the Reactjs framework. The actual code for these components can be found in Appendix B.

5.5.1 Control Unit

The control unit developed with JavaScript/TypeScript and nodejs. I would say that in hindsight, typescript would not have been necessary here as the program didn't end up very large in size, and I opted to work with JSON objects as JavaScript objects, instead of converting into types. It was built upon the Expressjs framework for creating web applications on Nodejs, allows to quickly create a web server and receiving and sending HTTP requests. This would be much of the intended functionality for the request handler, and might be considered part of it. The rest of the internal architecture with the different components remained true to the design.

Request handler

A lot of the functionality intended for this component was handled by the framework, that makes it easy to start a server, defined access point, that on HTTP request can call functions and return responses. The rest of the functionality of the request handler was in this case to handle the control flow from a request was received and between the components. In the case of a user query, it would initiate the cost model with the given configuration, make the route request data from the appropriate locations, give that data to the cost model to calculate rankings and send it back as a HTTP response. In the case of control unit querying it, it would skip the steps with the cost model, and only return data data.

Routing

The routing component consist of the a routing table as in the design, and some query functions that send requests data from different locations and returns the collection of the responses. The routing table works as explained in 4.3, in the a matching prefix of a entry in the routing table with the location stated in the search query would trigger a request to the data repository listed together with the entry in the routing table.

The way the query algorithms is implemented, might in some ways be affected by the constructs of the programming language and platform. It is based on the use of promises. To explain quickly how a promise works in JavaScript, when waiting for data to set the value of a variable, we can treat it as a promise of a value that either can have the state of undecided, resolved or rejected. If it is undecided, it has not gotten a value yet, it is resolved if it has a value, and rejected if some error causes it to not get its value. When dealing with several HTTP requests as promises in this case, where we want to component to return all results as fast as possible if they all resolve or reject, but if some requests takes too long time, we want to return those that resolved and skip the others after some time.

So we wrap all requests in a promise with a timer, so that they can race against it, and once every request is resolved, return the whole set of results.

Cost Model

The cost model is implemented as in the design, allowing it to set a set of hard constraints in the form of filters, and weights and utility functions as utility intervals for the ranking of the alternatives. The implementation of this is fairly simple, it receives a set of different alternatives in the form of metadata describing the KPIs of different data sets at different data repositories, iterates through them, and for each alternative, first checks if it should be removed due to the filtering of the hard constraints. If it is not filtered, it check for each KPI specified in the cost model, what utility score the measurement is equal to and multiplies it with the weight of the KPI, adding together the total score of the alternative. This is sorted from highest score, to lowest and returned with the alternative's metadata.

5.5.2 GUI

The GUI is implemented using React, which is a framework for creating web browser-based GUIs. React is a framework for Node.js and JavaScript, but also includes some of its own programming language concepts JSX, mixing HTML into the JavaScript. In this case, it use to create a web service that serves the GUI to the web browser, which itself is a stand alone GUI.

The implementation of the GUI consists of main components, representing two pages in the GUI. The first is the search and results page, which allows the user to specify a search query for data, specifying type of data, data location, and start and end time for when the data was collected. It also renders a list of the results of the query response from the control unit below the search input. The second component is the cost model configuration page, which allows the user to add hard constraints, and KPIs to cost model, setting their weight and adding utility intervals. The state of this configuration is kept when switching back to the search and results page, and the configuration data is sent with the query to the control unit.

5.5.3 Mocked Data repository

This also a simple web server built upon Node.js, written in TypeScript/JavaScript, using the Express.js framework. Since this is not a part of the designed solution, but a necessary component for testing the implementation, everything about the functionality is explained here. The data repositories stores different lists of metadata for data and information about its own KPIs in the format explained in section 5.2. It has very simple search functionality, that would be expected from the as data repository, returning filtering results solely based on the search query, but not any of the KPIs. Which is the location, type and date of the data sets. It returns a list metadata of all the data sets that matches the search query together with information about itself.

Chapter 6

Results

This chapter presents the results of the implementation of the ZEN Cost Model Prototype, in presenting part of the solution and some testing done for certain scenarios.

6.1 ZEN Cost Model Prototype

The results of the development, is the implementation of the control units that implement the proposed design in section 4, hosted locally and two different cloud locations. And a web server that serves a GUI used to send queries to the different control units. It is hard to present much of the more ”invisible” parts of the system, being the control units, the focus on this section will be show how the system is presented to a user through the GUI, and then present some of the results of using it for several use case scenarios, together with measurements of response times.

6.1.1 GUI

The final result of the GUI is shown by the two pages in the GUI and an explanation of how it works. The design is rather simple, and not very focused on looking good, but demonstrating the functionality of the whole system.

Cost Model Configuration

The cost model configuration page is the place where the user is able to create the inputs of different criteria for the cost model. The final result is shown in figure 6.1. The two top buttons, allows navigation between the ”Search” page and the current ”Configuration” page. The ”Add Criteria” button allows to add either a ”soft criteria”, which is a criteria that is used as part of the MAUT portion of the cost model, with weights and utility functions used for ranking, or a ”hard constraint” that acts as filter for solution. In the figure, there is a hard constraint on cost, that specifies a max cost of 10, and two soft criteria on

Figure 6.1: The cost model configuration page in the GUI

bandwidth and sampling rate, with their utility intervals below them, listed under the buttons. The ”+” button is used to add utility intervals to the soft criteria. This configuration is used when searching for data in on search page, demonstrated next.

Search and results

The ”Search” page also have the same two navigation buttons at the top as shown in figure 6.1. Beneath it is the search input for the data search, the search parameters are the data type, the location the data was generated, and the time period which it was generated. Below the search input, is where the result from a search is listed. There is no other interaction with this other then seeing the different matching data sets, the ranking of it and other related information. This is shown in figure 6.2, with three matching results when searching for what is assumed to be a specific sensor or room in a building at that location. The results are shown below, which includes the score to the left, and some of the overall information that is sent back, based in the response shown in section 5.2. If the sum of the weights specified in the cost configuration is 100, and all utility values are between 0 and 100, then the alternatives are also ranked between 100 and 0, with 100 being the best.

6.1.2 Functionality

Having looked how the different pages of the GUI, an example how this works is presented here. In this case, using mocked data sets, results for a query is presented and how the result might vary dependent on the cost model configuration. The data we are looking at

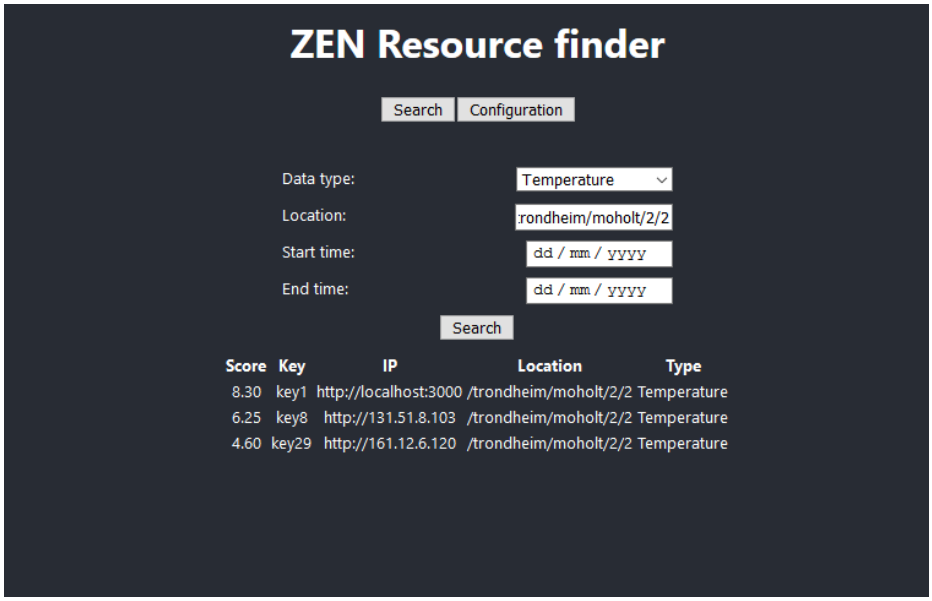


Figure 6.2: The cost model configuration page in the GUI

Type	Temperature
Location	/trondheim/moholt/1/1
Start time	2018-06-03
End time	2018-06-03

Table 6.1: Search query

in this example is stored distributed in the fog and as well as centralized in the cloud, from a single sensor from one day. The search parameters for the query is shown in table 6.1.

Query 1

The first query is done with focus on low cost and high bandwidth in the cost model configuration. The cost model configuration for this query is shown in figure 6.3. The result of this query gave the data set stored in cloud a score of 87.5, while the data set stored in the fog got a score of 17.5, meaning that the data set in the cloud is the best alternative according to the specified needs.

Query 2

The second query is done with focus on high sampling rate and a bit on cost. The cost model configuration for this query is shown in figure 6.4. The result of this query gave the data set stored in the fog a score of 82, while the data set stored in the fog got a score

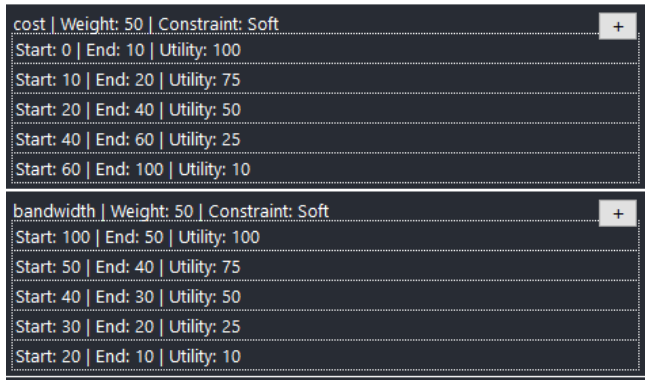


Figure 6.3: The cost model configuration for query 1

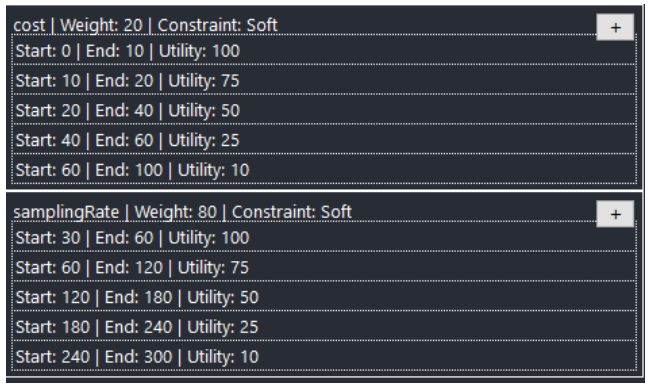


Figure 6.4: The cost model configuration for query 2

of 23, meaning that the data set stored in the fog is the best alternative according to the specified needs.

As we can see by these results, for certain preferences and needs, data stored in the distributed part of the system might better than the one stored in the cloud. And by using the cost model implementation, it can help decide when using data from the cloud is best, and when using data from the distributed repositories is the best alternative.

6.2 Measurements

Part of evaluating the implementation and design, in addition to see if it fulfill the requirements, is to do some measurements on the performance, specifically in relation to the decentralized-to-centralized KPIs presented in section 2.7.2, that might not only be used on the basis to pick data and service, but also evaluate this system itself. We will

be looking at latency measurements for several scenarios, that are specified in the next section.

6.2.1 Scenarios

This section describes the different scenario for testing the response times of the system. Within a short time period, 100 request were made for each to produce the resulting measurements.

Scenario 1: Getting data from the local fog and the cloud

In this scenario, we are querying for data generated locally, that might exist in the cloud as well, as historical data. Looking at figure 5.1, we are querying the local control unit, which then queries the cloud and local data sources. This would be the normal query for local data in our implementation, as there exists no way to know if data has been sent to the cloud or not in the design.

Scenario 2: Getting data generated in another fog area

In this scenario, we query data generated in another fog area, and using the figure 5.1, this would the data stored in Bergen or then Google Cloud. We query or local fog control unit, which queries the cloud, which in turn queries the control unit in that fog area. This would likely be the normal scenario for getting data generated in another fog area, by using the local control unit.

Scenario 3: Querying the cloud directly

In this scenario, we don't use the local fog control unit to query for data, we instead query the cloud directly. There is one issue here, since the locally hosted fog control unit is stuck behind a NAT router without my control, I am unable to query it from the cloud, so in this case, I have to query the other fog control unit. When reasoning about the difference querying the cloud for local data, or querying the local fog node, we can use this data in conjunction with the difference in response time between the two fog control units and the cloud control unit.

Scenario 4: Only getting data locally

In this case, we are looking at the performance if were able to only query the local fog control unit, due to it know what type of data already exists in the cloud. Does not work with the current implementation of routing, that needs to query the cloud to know if it has data or not.

Scenario 5: Only querying the cloud

In this scenario, we can compare to a centralized implementation, where the cloud holds all the data, or cases where the cloud knows that it is the only place were the data is held.

This does not work for the current implementation of routing, which does not now what data the lower layers holds.

6.2.2 Response times

We choose to look at the median and the mean, because most of the measurements deviate very little, but have a few outliers that are several times as large as the median. The median gives us the picture of the expected response time, but the mean with the standard deviation can give us an image of how the response time deviates in the different scenarios. The response times is given in milliseconds

Scenario	Median	Mean	Std
Scenario 1	132 ms	145 ms	75 ms
Scenario 2	247 ms	323 ms	160 ms
Scenario 3	182 ms	214 ms	77 ms
Scenario 4	26 ms	32 ms	13 ms
Scenario 5	66 ms	70 ms	14 ms

As mentioned in scenario 3, we have an issue comparing getting local data by querying a local control unit that also query the cloud, to querying the cloud which then queries the local fog. Since we are able to measure the delay between the fog nodes and the cloud, we can instead use the difference between the fog nodes to estimate what response time querying local data from the cloud would be. The fog node placed in Finland have a median response time of 94 ms to the cloud, where the local fog node has a median response time of 66 mm, which means that if we subtract 26 ms from the time to query the cloud for data in Finland, which is the difference between two fog nodes to the cloud, from the time to query the cloud for data in Finland in scenario 3, we get an estimate of 156 ms if we were to query the cloud directly for data in the local fog node. With that estimate, it seems there is a benefit to query the local fog, instead of the cloud in those cases, as it has a median response time of 132 ms.

Looking at the response time from scenario 4, which is on local area network and the travel time for the signal should be near instantaneous, means that most of the delay is from the program itself. Even if it were to be hosted somewhere in Trondheim, the latency should be extremely small as well, compared to the delay shown here. Which highlights the importance when creating a several programs that send data in a chain to each other, that they need to be quick.

Chapter 7

Discussion and Evaluation

The chapter contains the discussion about the process of this thesis, and an evaluation of the resulting artifact. The evaluation is done with the requirements identified in section 4.2.3, and the measurements presented in chapter 6. It also contains a small comparison of the former thesis [158], and a discussion on how the findings relate to ZEN.

7.1 Discussion

The start of this thesis, started with the literature review aimed at gaining knowledge on how solve the problem described in the problem description, finding services and data in the context of the distributed-to-centralized data management architecture for a smart city. In the beginning, I researched materials related to the smart city and the distributed-to-centralized, before moving on to the areas of KPI, selection methods, registry and description within the context of the architecture. This was aimed at being part of answering the first research questions, as well as being the grounding in creating a design to solve the problem. As mentioned in the research methodology in section 3.1, it uses an ad hoc approach to the literature review, which allowed to cover what was deemed as the necessary domains of knowledge. However, some areas have gotten different amounts of attention, which shows especially with the blockchain part, having much more material than other alternatives.

During the design, the choices made is based on the literature review. For the cost model, the choice ended on the MAUT-approach as it allows user to specify their needs in terms of utility and weights, allowing for a lot of fairly specific specification, but at the same time in a way that I found intuitive. How intuitive the GUI and approach is, was not tested. The only downside to the approach is that it lacks a strict constraints, so that was added to the design. Another consideration when deciding a cost model that attempts to help be cost efficient, is how to represent the relationship between utility and cost. In the MAUT approach, it is treated as an utility equal to other KPIs. Instead, an approach like in DEA in section 2.6.4, which divides the performance with the score, could be used to attempt

to get the most cost effective solution. But there are some issues with dividing by the cost when there might be free alternatives and you can not really specify the importance of cost in terms of weight either.
How to store and represent the data in the design, is in many ways shaped by the underlying data management architecture, so also using a distributed-to-centralized approach was chosen here. The blockchain, while having the benefit of decentralizing trust, and a hard to tamper with data structure, did not make sense in the context where a few actors using and storing the data are know to each other, with the downside the large computational overhead and growing data structure. And adaptive approach was not developed, because of the extra work of creating monitoring of services or data repositories with changing performance, when these did not exist. In order to make the process simple, it used static information given by the data repositories on their performance, instead measuring and benchmark testing, or using a trust system.

There were several different approaches for the design that were considered. One decision was where to have the cost model calculation. In a centralized schema, keeping it only in the cloud would have worked here, with issues accompanying centralized architecture. Another alternative was to have it locally with the user, which might been beneficial if an adaptive approach like the ACPS algorithm from section 2.6.9, used the cost model to calculate the change in performance. But the current solutions allows anyone to use cost model by just using the API, and letting the control unit which have a fair bit of computational resources, which a users device might not have, do the computation. Another decision was how to store and find information about the different data repositories. In the design an implementation, this was done through querying the data repositories. This seems beneficial in a scenario where queries are not happening that often, but the state of the repositories changes often, with the cost of some more delay. The other alternative is to let the control units keep the metadata about the data and the data repositories in the control units themselves. The bonus of this approach is that queries would be faster, but the downside is that amount of data traffic needed to keep an updated state of the data repositories and their data if they change often relative to queries. This could be done through periodical queries from the control unit, or the data repositories pushing updates to the control unit. Periodical queries need a frequency high enough to ensure the represent the current information, and pushing data on every change might result in a lot of unnecessary traffic. The best solution is likely based on the amount of queries and how often the states of the data repositories change.

From the results, we see a large benefit in response time by allowing a query for local data that does not query the cloud. And if it needs to query the cloud for local data, there seems to be a small benefit of querying the local control unit first, instead of the cloud. Another observation is that when querying the control unit, there is some delay in the processing in the control unit that quickly adds up when querying multiple of them, which is likely somewhat related to the technology used to implement it, and should be considered for a real implementation.

We also see that this allows for selection of data outside the centralized cloud, in cases where the needs of the users are such that a distributed repository is better. And in other cases, the needs of the user is such that the cloud is the best alternative. Helping the users

Requirement	Fulfilled
R1: Be able to find data according type, location, and time generated.	Yes
R2: Be able to specify preferred qualities with data or the provider of the data, to get the most suitable data.	Yes
R3: Be flexible with which types of KPIs can be used to describe a data provider or the data.	Yes
R4: Be able to find data as soon as it is accessible.	Yes
R5: Facilitate open data dissemination.	To some degree
R6: Vendor independence.	Yes

Table 7.1: The requirements for the design and implementation and their fulfilment

find the most suitable data through the cost model, is the main contribution of this paper and directly related to the problem description.

The testing was done by hosting the system two places in the cloud and locally on desktop computer. A better solution might have been to find hosting two cities where the pilot projects, and host the fog control units there, while having one in the cloud, to get more realistic results. Especially accessing the locally hosted over LAN compared to having it hosted in the same city, and the issue with the locally run control unit not being able to access queries from outside of the router, meaning the comparison using the other cloud hosted fog is a bit tenuous to use to claim that it would also hold for the locally hosted one. The reason for using this, was an issue of time and being familiar with the cloud technologies used, and accepting that it would be good enough, as the latency within a city is very small as well. These results can be used as part of the evaluation of the solution in the next section.

How intuitive and the ability to be able to specify preferences in the cost model should also have been tested by different users, to verify that view that this methods is actually intuitive and allows to specify as much of the user needs as possible, as this is one of the reasons that made it stand out from the other approaches.

7.2 Evaluation

The evaluation of the artifact created as part of this thesis, is done by looking at the requirements defined for it during the design, as well looking at the performance during testing.

Requirements

R5 is only partially fulfilled. It is a vague requirement in itself, but the current solution does not help explore and discover data as easily on its own, as it is dependent on knowledge of what that might exist to search for it and how to formulate the correct search terms.

R3 is fulfilled, as any new description of KPIs can easily be added and selected for as long it is specified as a numerical value. To allow the more flexibility, allowing other types of values, like nominal values, to be specified as utility functions would increase this flexibility as well, and should be easy to do with the current design.
R6 as this is a design that should could be implemented with a lot of different programming languages, programming platforms and web server frameworks, it is completely vendor independent.

Performance

The measurements shows that not having to directly access to cloud to query have a small response time benefit if the local control unit has to query the cloud for information about historical data, and a large one if it does not. Which means that a solution that does not need to use the cloud that often, can be preferable to a centralized cloud solution in terms of response time. It also shows that in situations when asking for information from another fog area, chaining queries through control units adds to the response time, which means that it is important that the control units are as fast as possible, which might exclude the current technology used to implement it.

7.3 Comparison with the centralized data management architecture

Looking at the former master thesis that creates a centralized architecture for data management for ZEN in [158], there have been some differences in terms of scope and what was done. In the former thesis, collecting, processing and analysis of data was part of the centralized architecture, together with dissemination of it. In this thesis, the scope was smaller, as the underlying ZEN ICT data management architecture would be responsible for gathering, processing and storing the data. The visualization and use of data would be done by other tools, like the tools in the ZEN toolbox presented in section 2.10.2. The task of connecting these two parts is the essence of this thesis, where the tools would be able to use the proposed design in their discover and selection of data. Instead of having everything in a single place, you have multiple, distributed components that are able to connect and use each other to create the same services.

7.4 ZEN contribution

The contribution of the thesis for ZEN is in creating a design for system that can help use the data managed by the ZEN ICT data management architecture. By using this system on top of the underlying data management system, third party applications and services, as well as the different tools in the ZEN toolbox can have the selection of data done automatically according to their needs of the data. It also includes a description of different KPIs that can used to select among the different data repositories, for a distributed-to-centralized context.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

The goal of this thesis was to solve the problem of finding and selecting services and data within a distributed-to-centralized data management architecture in the context of the smart city. The focus has been to do this for ZEN and the ZEN ICT architecture. It uses a literature review to find relevant KPIs to be used when selecting in this context, as well as other information that is used when designing the cost model. It then goes on to design and implement this cost model for the ZEN ICT architecture. As part of the evaluation of design and implementation, requirements for it is defined before the design which the implementation is tested against to see if it fulfills, as well as a testing its performance in terms of response time for different use case scenarios. It fulfilled the requirements set for it, and showed the benefit of a hosting parts of the implementation near local fog areas in the cities as part of the design, in cases where it does not have to contact the cloud, as well as indications of a small benefit when it does have to contact the cloud as well. And allowed selection of data from both the distributed part of the data management architecture, as well as the centralized one, depending on the user needs. The ZEN ICT architecture and similar distributed-to-centralized architectures can also be used for other smart cities, which makes the same cost model for it relevant to use in those cases as well.

The thesis follows a design science research methodology, where the contribution is from both the literature review in answering what methods and approaches can be used to implement the different parts of the design, which KPIs to use in this context, and the design artifact together with the evaluation of it within the application context of it.

8.2 Research Questions

RQ1: How can a cost model be defined for service and data selection in a distributed-to-centralized architecture for smart cities?

This research question consists of three sub-question that is part of answering the question. In the start it was identified that three things were needed to answer the overall question, which is answered below before answering this research question.

RQ1.1: Which KPI and criteria should be used with the cost model in this context?

To answer this question, it has been looked at KPIs from cloud computing, since it is the centralized component in the centralized-to-distributed architecture; KPIs identified specifically for the distributed-to-centralized schema; and for data selection in this context, also data quality KPIs have been identified. The following KPIs have been identified for cloud computing:

- Availability
- Instance Performance
- Network Performance
- Stability
- Security
- Elasticity and Scalability
- Cost
- Reliability
- Usability
- Management of Services
- Features
- Reputation and Trust

From the distributed-to-centralized schema, there are some overlap between the ones found in the cloud computing KPIs. These are as following:

- Processing load balance
- Energy consumption
- Latency
- Volatility and reliability
- Network load and traffic
- Security

For data quality, the following KPIs have been identfied:

- Accessibility
- Accuracy
- Amount of data
- Believability
- Completeness
- Concise representation

- Consistency
- Ease of manipulation
- Granularity
- Interpretability and Understandability
- Precision
- Relevancy and value-added
- Security
- Timeliness

These have varying degrees of specific ways to measure them. Some KPIs have very well defined measurement methods, and other have limited amount of work done to define them, so it is not straightforward to implement measures for every KPI. These are all the different KPIs that can be used, but which one that should be used amongst them depends on the needs of the user.

RQ1.2: What approaches and methods should be used to define a cost model in this context?

In this case, it has been looked at a set of different approaches to multi-criteria selection, and adaptive approaches that have been used for selection of services, mostly in the context of cloud service selection. Among these, MAUT from section 2.6.7 was chosen because it allowed to specify the needs of the user very specifically through utility functions and weights of importance, with the only drawback being the lack of being able to specify strict constraints, which was added to it in the cost model design. The other alternatives should also work for the same purpose with different pros and cons. There are also several other multi-criteria selection methods which might work just as good, that has not been covered.

RQ1.3: How should information about services be stored and shared in this context?

For this, three approaches was identified on how store the information, the first being the centralized approach, a decentralized approach, and a decentralized approach using blockchain. On top of the underlying architecture, it seems a decentralized approach works best because of the possible low response time together with lowering the traffic and processing in the cloud. The decentralized trust and immutability of the blockchain does not weight up for the downsides of it. But there are probably other contexts where these approaches might be better.

The complete cost model design and implementation can be defined by using the KPIs identified in RQ1.1, a cost model approach from RQ1.2, and an approach to store descriptions of the service or data from RQ1.3. This is what is used in the design and at implementation of the cost model for ZEN presention in chapter 4 and chapter 5, showing that it is possible to define and create using the knowledge gained from answering these three subquestions.

RQ2: How can an implementation of this cost model contribute to finding data in the ZEN ICT architecture?

The first thing it does, is help a user find where data is in the underlying data management architecture, regardless of where it is stored, abstracting away the complexity of the differ-

ent locations where data are stored. This can allow a service, user or application through this cost model treat the complete system as if it is only stored in one location. The next thing it does, is to select among the different locations to find the most suitable data for the user. In its current form, it only provides a list of the different places that stores data together with a ranking of how suitable the data and service is, but a ranking could quickly be made to a selection by choosing the best alternative, and automatically fetching the data from best alternative.

8.3 Future Work

To expand on the work done in this thesis, there are many things that can be approved upon. The first one is that more work is needed on defining ways to measure some of the KPIs identified. Some of them have clearly defined measurements, while others lack this. Related to the thesis specific implementation that uses measurement provided by the data repositories, future work can include how to let the system do these measurements themselves through benchmark testing, instead of relying on the information given to it. In a similar way, implementing a trust and reputation-based system shared by all user is also a possible future direction of work. Part of implementing ways to let the system itself implement measuring of performance, can also include adding adaptivity to the system, attempting to always change to the best solution. Another future direction could be to take an interest in the overall performance of the system when ranking and selecting, taking into consideration the network load or the processing load in different parts of the system. This could mean that it could be considered beneficial for a user to choose an alternative that is not the best alternative if the benefit is helping the overall system through avoiding processing nodes or part of the network that is currently near it capacity in terms of processing load or traffic.

The solution also lacks ways to explore the data, as you need knowledge of what data is out there to use the correct search terms to find it. Adding interfaces for exploration of data is also a future direction that can help with the facilitation of open data dissemination, which one of the requirements identified for the design.

On large drawback with the current design with the routing tables that was used, was that it needed to query the cloud for data, as it held no knowledge of what data might have been sent to the cloud as historic data. The measurements performed show a large benefit in response time by not querying the cloud, so looking into ways this can the case when querying local data, through storing that information locally, or knowing if data is not sent to the cloud yet, is also a possible direction of future work.

Lastly, a natural future direction is to try to implement the design or something similar with the actual underlying architecture, where it either query different actual locations for data, or is part of the data management system that stores the data.

Bibliography

[1] Abawajy, J., Nov 2011. Establishing trust in hybrid cloud computing environments. In: 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 118–125.

[2] Abbadi, I. M., Martin, A., 2011. Trust in the cloud. information security technical report 16 (3-4), 108–114.

[3] Achim, O., Pop, F., Cristea, V., Sep. 2011. Reputation based selection for services in cloud environments. In: 2011 14th International Conference on Network-Based Information Systems. pp. 268–273.

[4] Agarwal, S., Yadav, S., Yadav, A. K., 2016. An efficient architecture and algorithm for resource provisioning in fog computing. International Journal of Information Engineering and Electronic Business 8 (1), 48.

[5] Al-Dhuraibi, Y., Paraiso, F., Djarallah, N., Merle, P., March 2018. Elasticity in cloud computing: State of the art and research challenges. IEEE Transactions on Services Computing 11 (2), 430–447.

[6] Albino, V., Berardi, U., Dangelico, R. M., 2015. Smart cities: Definitions, dimensions, performance, and initiatives. Journal of urban technology 22 (1), 3–21.

[7] Alrawais, A., Alhothaily, A., Hu, C., Cheng, X., Mar 2017. Fog computing for the internet of things: Security and privacy issues. IEEE Internet Computing 21 (2), 34–42.

[8] Baird, L., 2016. The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep.

[9] Ballou, D. P., Tayi, G. K., 1999. Enhancing data quality in data warehouse environments. Communications of the ACM 42 (1), 73–78.

[10] Barnaghi, P. M., Bermudez-Edo, M., Tönjes, R., et al., 2015. Challenges for quality of data in smart cities. J. Data and Information Quality 6 (2-3), 6–1.

[11] Bass, L., J. L., Jain, S., 2013. Software architecture in paractice. Addison-Wesley Professional.

[12] Bauer, K., 2004. Kpis-the metrics that drive performance management. Information Management 14 (9), 63.

[13] Behzadian, M., Otaghsara, S. K., Yazdani, M., Ignatius, J., 2012. A state-of the-art survey of topsis applications. Expert Systems with applications 39 (17), 13051–13069.

[14] Bentov, I., Lee, C., Mizrahi, A., Rosenfeld, M., 2014. Proof of activity: Extending bitcoin's proof of work via proof of stake. IACR Cryptology ePrint Archive 2014, 452.

[15] Berander, P., Andrews, A., 2005. Requirements prioritization. In: Engineering and managing software requirements. Springer, pp. 69–94.

[16] Bonomi, F., Milito, R., Natarajan, P., Zhu, J., 2014. Fog computing: A platform for internet of things and analytics. In: Big data and internet of things: A roadmap for smart environments. Springer, pp. 169–186.

[17] Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog computing and its role in the internet of things. In: Proceedings of the first edition of the MCC workshop on Mobile cloud computing. ACM, pp. 13–16.

[18] Boussofiane, A., Dyson, R. G., Thanassoulis, E., 1991. Applied data envelopment analysis. European Journal of Operational Research 52 (1), 1–15.

[19] Bramas, Q., 2018. The stability and the security of the tangle.

[20] Buchanan, J. T., Sheppard, P. J., Vanderpooten, D., 1999. Project ranking using ELECTRE III. Department of Management Systems, University of Waikato.

[21] Budish, E., Cramton, P., Shim, J., 2015. The high-frequency trading arms race: Frequent batch auctions as a market design response. The Quarterly Journal of Economics 130 (4), 1547–1621.

[22] Buyya, R., Yeo, C. S., Venugopal, S., Sep. 2008. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In: 2008 10th IEEE International Conference on High Performance Computing and Communications. pp. 5–13.

[23] Caragliu, A., Del Bo, C., Nijkamp, P., 2011. Smart cities in europe. Journal of urban technology 18 (2), 65–82.

[24] Castro, M., Liskov, B., et al., 1999. Practical byzantine fault tolerance. In: OSDI. Vol. 99. pp. 173–186.

[25] Chabridon, S., Laborde, R., Desprats, T., Oglaza, A., Marie, P., Marquez, S. M., 2014. A survey on addressing privacy together with quality of context for context management in the internet of things. Annals of telecommunications-annales des télécommunications 69 (1-2), 47–62.

[26] Chawathe, Y., Ratnasamy, S., Breslau, L., Lanham, N., Shenker, S., 2003. Making gnutella-like p2p systems scalable. In: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications. ACM, pp. 407–418.

[27] Chen, J., Micali, S., 2016. Algorand. arXiv preprint arXiv:1607.01341.

[28] Chen, M., Mao, S., Liu, Y., Apr 2014. Big data: A survey. Mobile Networks and Applications 19 (2), 171–209.
URL https://doi.org/10.1007/s11036-013-0489-0

[29] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., et al., 2001. Web services description language (wsdl) 1.1.

[30] Cocchia, A., 2014. Smart and Digital City: A Systematic Literature Review. Springer International Publishing, Cham, pp. 13–43.
URL https://doi.org/10.1007/978-3-319-06160-3_2

[31] Cooper, B. F., Silberstein, A., Tam, E., Ramakrishnan, R., Sears, R., 2010. Benchmarking cloud serving systems with ycsb. In: Proceedings of the 1st ACM symposium on Cloud computing. ACM, pp. 143–154.

[32] D. Baer, I. Andresen, 2018. ZEN pilot projects: Mapping of the pilot projects within the Research Centre on Zero Emission Neighbourhoods in Smart Cities. ZEN Report 10.

[33] Dastjerdi, A. V., Gupta, H., Calheiros, R. N., Ghosh, S. K., Buyya, R., 2016. Fog computing: Principles, architectures, and applications. In: Internet of things. Elsevier, pp. 61–75.

[34] Deb, K., 2014. Multi-objective optimization. In: Search methodologies. Springer, pp. 403–449.

[35] Dejun, J., Pierre, G., Chi, C.-H., 2010. Ec2 performance analysis for resource provisioning of service-oriented applications. In: Dan, A., Gittler, F., Toumani, F. (Eds.), Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 197–207.

[36] Demchenko, Y., Grosso, P., de Laat, C., Membrey, P., May 2013. Addressing big data issues in scientific data infrastructure. In: 2013 International Conference on Collaboration Technologies and Systems (CTS). pp. 48–55.

[37] Deng, R., Lu, R., Lai, C., Luan, T. H., June 2015. Towards power consumption-delay tradeoff by workload allocation in cloud-fog computing. In: 2015 IEEE International Conference on Communications (ICC). pp. 3909–3914.

[38] Devillers, R., Bédard, Y., Jeansoulin, R., 2005. Multidimensional management of geospatial data quality information for its dynamic use within gis. Photogrammetric Engineering & Remote Sensing 71 (2), 205–215.

[39] Elmaghraby, A. S., Losavio, M. M., 2014. Cyber security challenges in smart cities: Safety, security and privacy. Journal of advanced research 5 (4), 491–497.

[40] Ethereum community, c2016. Ethereum homestead documentation. `http://ethdocs.org/en/latest/`, accessed: 2010-01-21.

[41] Ezbakhe, F., Prez-Foguet, A., 07 2017. Improved prioritisation tool for local decision-making in the water, sanitation and hygiene sector.

[42] Feng, J., Zhang, L., Lu, J., Xu, B., 2017. Cpe: a cloud server performance evaluation model. In: Proceedings of the 11th International Conference on Ubiquitous Information Management and Communication. ACM, p. 60.

[43] Fiedler, M., Hossfeld, T., Tran-Gia, P., 2010. A generic quantitative relationship between quality of experience and quality of service. IEEE Network 24 (2), 36–41.

[44] Fielding, R. T., Taylor, R. N., 2000. Architectural styles and the design of network-based software architectures. Vol. 7. University of California, Irvine Doctoral dissertation.

[45] Figueira, J., Mousseau, V., Roy, B., 2005. Electre methods. In: Multiple criteria decision analysis: State of the art surveys. Springer, pp. 133–153.

[46] Fleming, P. J., Wallace, J. J., 1986. How not to lie with statistics: the correct way to summarize benchmark results. Communications of the ACM 29 (3), 218–221.

[47] Furht, B., 2010. Cloud computing fundamentals. In: Handbook of cloud computing. Springer, pp. 3–19.

[48] Furht, B., Escalante, A., 2010. Handbook of cloud computing. Vol. 3. Springer.

[49] Ganesh, A., Sandhya, M., Shankar, S., Feb 2014. A study on fault tolerance methods in cloud computing. In: 2014 IEEE International Advance Computing Conference (IACC). pp. 844–849.

[50] Gao, W., Hatcher, W. G., Yu, W., 2018. A survey of blockchain: Techniques, applications, and challenges. In: 2018 27th International Conference on Computer Communication and Networks (ICCCN). IEEE, pp. 1–11.

[51] Garg, S. K., Versteeg, S., Buyya, R., Dec 2011. Smicloud: A framework for comparing and ranking cloud services. In: 2011 Fourth IEEE International Conference on Utility and Cloud Computing. pp. 210–218.

[52] Guptill, S. C., Morrison, J. L., 2013. Elements of spatial data quality. Elsevier.

[53] Habib, S. M., Ries, S., Muhlhauser, M., Nov 2011. Towards a trust management system for cloud computing. In: 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications. pp. 933–939.

[54] Hang, C.-W., Singh, M. P., 2010. From quality to utility: Adaptive service selection framework. In: International Conference on Service-Oriented Computing. Springer, pp. 456–470.

[55] Hasan, R., Anwar, Z., Yurcik, W., Brumbaugh, L., Campbell, R., April 2005. A survey of peer-to-peer storage techniques for distributed file systems. In: International Conference on Information Technology: Coding and Computing (ITCC'05) - Volume II. Vol. 2. pp. 205–213 Vol. 2.

[56] Hashem, I. A. T., Chang, V., Anuar, N. B., Adewole, K., Yaqoob, I., Gani, A., Ahmed, E., Chiroma, H., 2016. The role of big data in smart city. International Journal of Information Management 36 (5), 748–758.

[57] Herbst, N. R., Kounev, S., Reussner, R., 2013. Elasticity in cloud computing: What it is, and what it is not. In: Proceedings of the 10th International Conference on Autonomic Computing ({ICAC} 13). pp. 23–27.

[58] Hevner, A., Chatterjee, S., 2010. Design science research in information systems. In: Design research in information systems. Springer, pp. 9–22.

[59] Hinchey, M., Coyle, L., March 2010. Evolving critical systems: A research agenda for computer-based systems. In: 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems. pp. 430–435.

[60] Hwang, C.-L., Yoon, K., 2012. Multiple attribute decision making: methods and applications a state-of-the-art survey. Vol. 186. Springer Science & Business Media.

[61] IOTA Foundation, 2018. Coordinator. part 1: The path to coordicide. https://blog.iota.org/coordinator-part-1-the-path-to-coordicide-ee4148a8db08, [Online; accessed 18-May-2019].

[62] Jekosch, P., 2005. Voice and speech quality perception: Assessment and evaluation-ute.

[63] Julien Paulou et al., 2014. Financing the energy renovation of buildings with CohesionPolicy funding. Tech. rep.

[64] Karpak, B., Kumcu, E., Kasuganti, R. R., 2001. Purchasing materials in the supply chain: managing a multi-objective task. European Journal of Purchasing & Supply Management 7 (3), 209–216.

[65] Khatoun, R., Zeadally, S., 2016. Smart cities: concepts, architectures, research opportunities. Commun. Acm 59 (8), 46–57.

[66] Krause, M., Hochstatter, I., 2005. Challenges in modelling and using quality of context (qoc). In: International Workshop on Mobile Agents for Telecommunication Applications. Springer, pp. 324–333.

[67] Larsen, I. K., Småstuen, M., Johannesen, T. B., Langmark, F., Parkin, D. M., Bray, F., Møller, B., 2009. Data quality at the cancer registry of norway: an overview of comparability, completeness, validity and timeliness. European journal of cancer 45 (7), 1218–1231.

[68] Lee, Y. W., Strong, D. M., Kahn, B. K., Wang, R. Y., 2002. Aimq: a methodology for information quality assessment. Information & management 40 (2), 133–146.

[69] Luan, T. H., Gao, L., Li, Z., Xiang, Y., Wei, G., Sun, L., 2015. Fog computing: Focusing on mobile users at the edge. arXiv preprint arXiv:1502.01815.

[70] Luna, J., Taha, A., Trapero, R., Suri, N., 2017. Quantitative reasoning about cloud security using service level agreements. IEEE Transactions on Cloud Computing 5 (3), 457–471.

[71] Luna Garcia, J., Langenberg, R., Suri, N., 2012. Benchmarking cloud security level agreements using quantitative policy trees. In: Proceedings of the 2012 ACM Workshop on Cloud computing security workshop. ACM, pp. 103–112.

[72] Lundy, M., Mees, A., 1986. Convergence of an annealing algorithm. Mathematical programming 34 (1), 111–124.

[73] M. K. Wiik, S. M. Fufa, J. Krogstie, D. Ahlers, A. Wyckmans, P. Driscoll, H. Bratteb, A. Gustavsen, 2018. Zero Emission Neighbourhoods in Smart Cities - Definition, key performance indicators and assessment criteria. ZEN Report 7.

[74] Manuel, P., Oct 2015. A trust model of cloud computing based on quality of service. Annals of Operations Research 233 (1), 281–292.
URL https://doi.org/10.1007/s10479-013-1380-x

[75] Manzoor, A., Truong, H.-L., Dustdar, S., 2008. On the evaluation of quality of context. In: European Conference on Smart Sensing and Context. Springer, pp. 140–153.

[76] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al., 2004. Owl-s: Semantic markup for web services. W3C member submission 22 (4).

[77] Masip-Bruin, X., Marín-Tordera, E., Alonso, A., Garcia, J., 2016. Fog-to-cloud computing (f2c): The key technology enabler for dependable e-health services deployment. In: 2016 Mediterranean ad hoc networking workshop (Med-Hoc-Net). IEEE, pp. 1–5.

[78] Masip-Bruin, X., Marn-Tordera, E., Tashakor, G., Jukan, A., Ren, G., October 2016. Foggy clouds and cloudy fogs: a real need for coordinated management of fog-to-cloud computing systems. IEEE Wireless Communications 23 (5), 120–128.

[79] Mell, P., Grance, T., et al., 2011. The nist definition of cloud computing.

[80] Menditto, A., Patriarca, M., Magnusson, B., Jan 2007. Understanding the meaning of accuracy, trueness and precision. Accreditation and Quality Assurance 12 (1), 45–47.
URL https://doi.org/10.1007/s00769-006-0191-z

[81] Moradian, E., Håkansson, A., 2006. Possible attacks on xml web services. IJCSNS International Journal of Computer Science and Network Security 6 (1B), 154–170.

[82] Morton, A., Claise, B., 2009. Packet delay variation applicability statement. Tech. rep.

[83] Nam, T., Pardo, T. A., 2011. Conceptualizing smart city with dimensions of technology, people, and institutions. In: Proceedings of the 12th annual international digital government research conference: digital government innovation in challenging times. ACM, pp. 282–291.

[84] Naskos, A., Gounaris, A., Sioutas, S., 2016. Cloud elasticity: A survey. In: Karydis, I., Sioutas, S., Triantafillou, P., Tsoumakos, D. (Eds.), Algorithmic Aspects of Cloud Computing. Springer International Publishing, Cham, pp. 151–167.

[85] Neirotti, P., De Marco, A., Cagliano, A. C., Mangano, G., Scorrano, F., 2014. Current trends in smart city initiatives: Some stylised facts. Cities 38, 25–36.

[86] Nguyen, D. K., Lelli, F., Taher, Y., Parkin, M., Papazoglou, M. P., van den Heuvel, W.-J., 2011. Blueprint template support for engineering cloud-based services. In: European Conference on a Service-Based Internet. Springer, pp. 26–37.

[87] Ning, Z., Huang, J., Wang, X., February 2019. Vehicular fog computing: Enabling real-time traffic management for smart cities. IEEE Wireless Communications 26 (1), 87–93.

[88] Noor, T. H., Sheng, Q. Z., 2011. Credibility-based trust management for services in cloud environments. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H. R. (Eds.), Service-Oriented Computing. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 328–343.

[89] Noor, T. H., Sheng, Q. Z., 2011. Trust as a service: A framework for trust management in cloud environments. In: Bouguettaya, A., Hauswirth, M., Liu, L. (Eds.), Web Information System Engineering – WISE 2011. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 314–321.

[90] Noor, T. H., Sheng, Q. Z., Alfazi, A., July 2013. Reputation attacks detection for effective trust assessment among cloud services. In: 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications. pp. 469–476.

[91] Olson, J. E., 2003. Data quality: the accuracy dimension. Elsevier.

[92] Osmani, M., O'Reilly, A., 2009. Feasibility of zero carbon homes in england by 2016: A house builder's perspective. Building and environment 44 (9), 1917–1924.

[93] P4Titan, 2014. Slimcoin a peer-to-peer crypto-currency with proof-of-burn.

[94] Panarello, A., Tapas, N., Merlino, G., Longo, F., Puliafito, A., 2018. Blockchain and iot integration: A systematic survey. Sensors 18 (8), 2575.

[95] Pawar, P. S., Rajarajan, M., Nair, S. K., Zisman, A., 2012. Trust model for optimized cloud services. In: Dimitrakos, T., Moona, R., Patel, D., McKnight, D. H. (Eds.), Trust Management VI. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 97–112.

[96] Pipino, L. L., Lee, Y. W., Wang, R. Y., 2002. Data quality assessment. Communications of the ACM 45 (4), 211–218.

[97] Plaxton, C. G., Rajaraman, R., Richa, A. W., 1999. Accessing nearby copies of replicated objects in a distributed environment. Theory of computing systems 32 (3), 241–280.

[98] Popov, S., 2016. The tangle. cit. on, 131.

[99] Prat, N., Madnick, S., Jan 2008. Measuring data believability: A provenance approach. In: Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008). pp. 393–393.

[100] Prodan, R., Ostermann, S., Oct 2009. A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In: 2009 10th IEEE/ACM International Conference on Grid Computing. pp. 17–25.

[101] Prodan, R., Sperk, M., Ostermann, S., March 2012. Evaluating high-performance computing on google app engine. IEEE Software 29 (2), 52–58.

[102] Qu, C., Buyya, R., May 2014. A cloud trust evaluation system using hierarchical fuzzy inference system for service selection. In: 2014 IEEE 28th International Conference on Advanced Information Networking and Applications. pp. 850–857.

[103] Ramgovind, S., Eloff, M. M., Smith, E., Aug 2010. The management of security in cloud computing. In: 2010 Information Security for South Africa. pp. 1–7.

[104] Ramírez, W., Masip-Bruin, X., Marin-Tordera, E., Souza, V. B. C., Jukan, A., Ren, G.-J., de Dios, O. G., 2017. Evaluating the benefits of combined and continuous fog-to-cloud architectures. Computer Communications 113, 43–52.

[105] Ratnasamy, S., Stoica, I., Shenker, S., 2002. Routing algorithms for dhts: Some open questions. In: International Workshop on Peer-to-Peer Systems. Springer, pp. 45–52.

[106] Recommendation, E., 2008. 800: Definitions of terms related to quality of service. International Telecommunication Unions Telecommunication Standardization Sector (ITU-T) Std.

[107] Ren, L., Devadas, S., 2016. Proof of space from stacked expanders. In: Theory of Cryptography Conference. Springer, pp. 262–285.

[108] Repschlaeger, J., Zarnekow, R., Wind, S., Turowski, K., et al., 2012. Cloud requirement framework: Requirements and evaluation criteria to adopt cloud solutions. In: ECIS. p. 42.

[109] Romero, C., 2014. Handbook of critical issues in goal programming. Elsevier.

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

[111] Ruiz-Alvarez, A., Humphrey, M., 2011. An automated approach to cloud storage service selection. In: Proceedings of the 2nd international workshop on Scientific cloud computing. ACM, pp. 39–48.

[112] Saldana, J., Suznjevic, M., 08 2015. QoE and Latency Issues in Networked Games.

[113] Saravanan, K., Kantham, M. L., 2013. An enhanced qos architecture based framework for ranking of cloud services. International Journal of Engineering Trends and Technology (IJETT) 4 (4), 1022–1031.

[114] Saroiu, S., Gummadi, K. P., Gribble, S. D., 2003. Measuring and analyzing the characteristics of napster and gnutella hosts. Multimedia systems 9 (2), 170–184.

[115] Sartori, I., Napolitano, A., Voss, K., 2012. Net zero energy buildings: A consistent definition framework. Energy and buildings 48, 220–232.

[116] Schmoldt, D., Kangas, J., Mendoza, G. A., Pesonen, M., 2013. The analytic hierarchy process in natural resource and environmental decision making. Vol. 3. Springer Science & Business Media.

[117] Schwartz, D., Youngs, N., Britto, A., et al., 2014. The ripple protocol consensus algorithm. Ripple Labs Inc White Paper 5.

[118] Sebastiao, R., Gama, J., 2009. A study on change detection methods. In: Progress in Artificial Intelligence, 14th Portuguese Conference on Artificial Intelligence, EPIA. pp. 12–15.

[119] Sengupta, S., Garcia, J., Masip-Bruin, X., 2018. An architecture for resource management in a fog-to-cloud framework. In: European Conference on Parallel Processing. Springer, pp. 275–286.

[120] Sengupta, S., Garcia, J., Masip-Bruin, X., 2018. Essentiality of resource and service-task characterization in the coordinated fog-to-cloud paradigm. In: 2018 International Conference on Smart Communications in Network Technologies (SaCoNeT). IEEE, pp. 249–254.

[121] Sengupta, S., Garcia, J., Masip-Bruin, X., 2018. Taxonomy and resource modeling in combined fog-to-cloud systems. In: Proceedings of the Future Technologies Conference. Springer, pp. 687–704.

[122] Shaikh, R., Sasikumar, M., 2015. Trust model for measuring security strength of cloud computing service. Procedia Computer Science 45, 380–389.

[123] Shawky, D. M., Ali, A. F., Aug 2012. Defining a measure of cloud computing elasticity. In: 2012 1st International Conference on Systems and Computer Science (ICSCS). pp. 1–5.

[124] Sinaeepourfard, A., Garcia, J., Masip-Bruin, X., Marín-Torder, E., 2016. Towards a comprehensive data lifecycle model for big data environments. In: Proceedings of the 3rd IEEE/ACM International Conference on Big Data Computing, Applications and Technologies. ACM, pp. 100–106.

[125] Sinaeepourfard, A., Garcia, J., Masip-Bruin, X., Marin-Tordera, E., 2018. Data preservation through fog-to-cloud (f2c) data management in smart cities. In: 2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC). IEEE, pp. 1–9.

[126] Sinaeepourfard, A., Krogstie, J., Petersen, S. A., 2018. A big data management architecture for smart cities based on fog-to-cloud data management architecture.

[127] Sinaeepourfard, A., Krogstie, J., Petersen, S. A., Gustavsen, A., Oct 2018. A zero emission neighbourhoods data management architecture for smart city scenarios: Discussions toward 6vs challenges. In: 2018 International Conference on Information and Communication Technology Convergence (ICTC). pp. 658–663.

[128] Song, W., Su, X., May 2011. Review of mobile cloud computing. In: 2011 IEEE 3rd International Conference on Communication Software and Networks. pp. 1–4.

[129] Souidi, M., Souihi, S., Hoceini, S., Mellouk, A., 2015. An adaptive real time mechanism for iaas cloud provider selection based on qoe aspects. In: 2015 IEEE International Conference on Communications (ICC). IEEE, pp. 6809–6814.

[130] Souihi, S., Souidi, M., Mellouk, A., 2015. An adaptive qoe-based network interface selection for multi-homed ehealth devices. In: International internet of things summit. Springer, pp. 437–442.

[131] Souza, V., Masip-Bruin, X., Marín-Tordera, E., Sànchez-López, S., Garcia, J., Ren, G.-J., Jukan, A., Ferrer, A. J., 2018. Towards a proper service placement in combined fog-to-cloud (f2c) architectures. Future Generation Computer Systems 87, 1–15.

[132] Souza, V. B., Masip-Bruin, X., Marin-Tordera, E., Ramirez, W., Sanchez, S., Dec 2016. Towards distributed service allocation in fog-to-cloud (f2c) scenarios. In: 2016 IEEE Global Communications Conference (GLOBECOM). pp. 1–6.

[133] Souza, V. B. C., Ramrez, W., Masip-Bruin, X., Marn-Tordera, E., Ren, G., Tashakor, G., May 2016. Handling service allocation in combined fog-cloud scenarios. In: 2016 IEEE International Conference on Communications (ICC). pp. 1–5.

[134] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., Balakrishnan, H., 2001. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review 31 (4), 149–160.

[135] Strong, D. M., Lee, Y. W., Wang, R. Y., 1997. Data quality in context. Communications of the ACM 40 (5), 103–110.

[136] Sun, L., Dong, H., Hussain, F. K., Hussain, O. K., Chang, E., 2014. Cloud service selection: State-of-the-art and future research directions. Journal of Network and Computer Applications 45, 134–150.

[137] Takeda, H., Veerkamp, P., Yoshikawa, H., 1990. Modeling design process. AI magazine 11 (4), 37–37.

[138] Tanenbaum, A. S., Van Steen, M., 2007. Distributed systems: principles and paradigms. Prentice-Hall.

[139] Tang, B., Chen, Z., Hefferman, G., Wei, T., He, H., Yang, Q., 2015. A hierarchical distributed fog computing architecture for big data analysis in smart cities. In: Proceedings of the ASE BigData & SocialInformatics 2015. ACM, p. 28.

[140] Tayi, G. K., Ballou, D. P., 1998. Examining data quality. Communications of the ACM 41 (2), 54–57.

[141] Truong, H.-L., Dustdar, S., 2010. Composable cost estimation and monitoring for computational applications in cloud computing environments. Procedia Computer Science 1 (1), 2175–2184.

[142] u. Rehman, Z., Hussain, F. K., Hussain, O. K., June 2011. Towards multi-criteria cloud service selection, 44–48.

[143] United Nations, Department of Economic and Social Affairs, 2018. 2018 Revision of World Urbanization Prospects.
URL https://www.un.org/development/desa/publications/2018-revision-of-world-urbanization-prospects.html

[144] United Nations, Department of Economic and Social Affairs, Population Division, 2017. World Population Prospects: The 2017 Revision, Key Findings and Advance Tables.
URL https://population.un.org/wpp/Publications/Files/WPP2017_KeyFindings.pdf

[145] Van Dinh, D., Yoon, B., Le, H. N., Nguyen, U. Q., Dang Phan, K., Dinh Pham, L., Feb 2018. Ict enabling technologies for smart cities. In: 2018 20th International Conference on Advanced Communication Technology (ICACT). pp. 606–611.

[146] Varshney, P., Simmhan, Y., May 2017. Demystifying fog computing: Characterizing architectures, applications and abstractions. In: 2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC). pp. 115–124.

[147] Voas, J., Zhang, J., March 2009. Cloud computing: New wine or just a new bottle? IT Professional 11 (2), 15–17.

[148] Von Winterfeldt, D., Fischer, G. W., 1975. Multi-attribute utility theory: models and assessment procedures. In: Utility, probability, and human decision making. Springer, pp. 47–85.

[149] Walnum, H.T., K. Srnes, M. Mysen, .L. Srensen, and A.-J. Alms, 2017. Preliminary toolkit for goals and KPIs, in PI-SEC Planning Instruments for Smart Energy Communities.

[150] Wang, R. Y., Strong, D. M., 1996. Beyond accuracy: What data quality means to data consumers. Journal of management information systems 12 (4), 5–33.

[151] Wang, S., Wei, J., Sun, L., Sun, Q., Yang, F., Dec 2013. Reputation measurement of cloud services based on unstable feedback ratings. In: 2013 International Conference on Parallel and Distributed Systems. pp. 474–479.

[152] Watkins, C. J. C. H., Dayan, P., 1992. Q-learning. In: Machine Learning. pp. 279–292.

[153] Weber, A., Herbst, N., Groenda, H., Kounev, S., 2014. Towards a resource elasticity benchmark for cloud environments. In: Proceedings of the 2nd International Workshop on Hot Topics in Cloud service Scalability. ACM, p. 5.

[154] Whaiduzzaman, M., Gani, A., Feb 2014. Measuring security for cloud service provider: A third party approach. In: 2013 International Conference on Electrical Information and Communication Technology (EICT). pp. 1–6.

[155] Wiik, M. K., Fufa, S. M., Baer, D., Sartori, I. Andresen, I., 2018. The ZEN definition a guideline for the ZEN pilot areas. Version 1.0. ZEN Report 11.

[156] Woods, R., Reme, K. S., Hestnes, A. G., Gustavsen, A. (eds.) , 2019. Annual Report 2018. ZEN Report 15.

[157] Yi, S., Hao, Z., Qin, Z., Li, Q., Nov 2015. Fog computing: Platform and applications. In: 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). pp. 73–78.

[158] Ystmark, A. M., 2018. Building a Data Management Architecture for Zero-Emission Neighbourhoods in Smart Cities. Master's thesis, NTNU, Trondheim.

[159] Zhao, B. Y., Kubiatowicz, J., Joseph, A. D., et al., 2001. Tapestry: An infrastructure for fault-tolerant wide-area location and routing.

[160] Zheng, Z., Xie, S., Dai, H.-N., Wang, H., 2016. Blockchain challenges and opportunities: A survey. Work Pap.–2016.

[161] Zhou, B., Dastjerdi, A. V., Calheiros, R. N., Srirama, S. N., Buyya, R., June 2015. A context sensitive offloading scheme for mobile cloud computing service. In: 2015 IEEE 8th International Conference on Cloud Computing. pp. 869–876.

[162] Zhu, C., Nicanfar, H., Leung, V. C. M., Yang, L. T., Jan 2015. An authenticated trust and reputation calculation and management system for cloud and sensor networks integration. IEEE Transactions on Information Forensics and Security 10 (1), 118–131.

Appendix A

Appendix A contains the installation and user guide for the programming artifacts of this thesis. The installation guide first explains how to install the control unit, before explaining how to install the GUI. The control unit and mocked data repository is in its own project folder, while the GUI also has its own separate from this. So the installation guide is divided in two for each of the project folders.

Control Unit and Mocked Data Repository Installation

1. **Install Node.js and NPM**

   Both the GUI and the control unit program is written in TypeScript/JavaScript for Node.js, and is dependent on Node.js to run. The version of Node.js used is version 10.14.0. The software needed to install imported modules and build the program is called Node Packet Manager (NPM), and version 6.9.0 was used for with the program. Using different versions might cause issues.

   NPM is most likely included in the Node.js installation.

2. **Install the required modules for the control unit program**

   From the root directory of the project folder, run ”npm install” in the terminal or command line. The root project directory contains the package.json file which is needed this command to work. This will install all the required modules for this project.

3. **Install TypeScript**

   This is needed to build the program. It compiles TypeScript into runnable JavaScript. Install this by running ”npm install typescript” from the same directory as in step 2. The version of TypeScript used for this project is version 3.2.1.

4. **Compile the TypeScript into JavaScript**

   By using the command ”npm run build”, also from the root project directory, the TypeScript files found in the ”lib” folder is compiled into JavaScript found in the ”dist” folder in the root project directory. If anything is changed in the TypeScript files, they need to be compiled before running the program, for the changes to take effect.

5. **Configurate the config files**

   By this point, both the control unit and the mocked data repository, have their dependencies installed, is compiled to JavaScript. The last step before running either of the programs, is to make sure that their simple config files are as they should be.

   (a) *Control Unit Config:*

   The control unit config file is found in the root of the project directory, named ”config.json”. It has two entries, the first being its own IP-address, and the second being the entries into the routing table. Its own IP-address is very important, as this part of the mechanism that stops an endless loop of queries between two control units from happening, which translates into monetary loss in a cloud environment. This helps by attaching the IP-address to each forwarded query, which stops the next control unit from forwarding the query to the source. And it also means that endless cycles can happen if the network organized by the routing table contains cycles as well. So structure of network formed by the routing tables is important, maintain the tree structure in the ZEN ICT architecture.

   The routing table contains entries that map data location to different data repositories and control unit. How this works is explained in chapter 4, but the short version is that if an entry in the routing table’s key matches as a prefix to the query’s location, the associated data repository or control unit is queried.

   (b) *Mocked Data Repository Config*

   The config of the mocked data repository is also found in the root of the project directory, named ”dataConfig.json”. The first entry is its own IP-address, which is not needed, as no logic is dependent on it, besides displaying it in the GUI. The next entry, ”dataSource”, decides which JSON file it imports as mocked data set descriptions. These data sets are found in the ”mockedData” folder in the root project directory. The two next entries, ”bandwidth” and ”resourceCost”, is KPIs used by the cost model for selection. At last, the ”delay” entry adds delay to the response of mocked data repository, set in milliseconds.

6. **Run the program**

   Finally, the programs can started. From the root of the project directory, ”npm run startc” starts a control unit using port 3001, while ”npm run startd” starts a mocked data repository using port 3000. Ctrl-c in terminal or command line to stop the program running.

Contents of the control unit project folder

In this part, a short explanation and overview of the different content of the control unit project is presented. At the root of the project folder, is the different config files. ”package.json”, ”package-lock.json”, and ”tsconfig.json” is configuration files for a Node.js

project with TypeScript, while the ”dataConfig.json” and ”config.json” is config files for the program itself. The ”node_modules” folder contains all the installed dependencies

The ”lib” folder contains the program files. Within it, is a folder named ”controlUnit” take contains all the code for the control unit, and the ”dataApp” folder, which contains all the code for the mocked data repository. In the ”mockedData” lies several mocked data set descriptions that are used by the mocked data repositories as part of the mocking.

GUI installation

The GUI also depends on Node.js, so if it has not been installed, look at step 1 in the installation guide for the control unit and mocked data repository, and perform it before any steps in this installation guide.

1. **Install the required modules for the control unit program**

   From the root of the project folder (folder containing ”package.json”), open terminal or command line and run the command ”npm install”.

2. **Change the hard coded control unit IP-address**

   The ip-address the queries are sent to is hard coded in the file ”app.js”, found in the ”src” folder. To change the IP-address the queries are sent to, the IP-address string in the ”handleClick” function must be changed.

3. **Run the program**

   Run the program opening a terminal or command line in the root of the project directory and running the command ”npm start”, this will open a new window in the browser containing the GUI. All state that is stored in the GUI, like cost model configuration, disappears when refreshing the page. Ctrl-c in the terminal or command line running the program stops it.

Contents of the GUI project folder

The program is based on the React’s own initialized project template. The only files that are change and that are of interest, is in the ”src” folder, with ”app.js”, that contains all the logic of the search page, and ”config.js” that contains all the logic of the cost model configuration page.

Appendix B

Appendix B contains the code from the three components of the control unit, the request handler, the router, and the cost model, as these can be considered the most important parts of the design and prototype.

Request Controller Implementation

```
export class RequestHandler {

    public router: Router;

    constructor() {
        this.router = new Router();
    }

    public async handle(req) {
        console.log(req);
        if (req['query'] === 'user') {
            //build ranking stuff here
            const rankingFunction = new RankAndSelector();
            rankingFunction.addUtilityFunctionsOrFilters(req['criteria']['criteriaList'], req['criteria']['intervalList']);
            //fetch all the different data sources
            const results = await this.router.findRepositories(req['search'], "user");
            //rank the results according to ranking stuff
            const flattenResults = this.flattenDeep(results);
            const ranking = rankingFunction.calculateRanking(flattenResults);
            //send the results back
            return {results: ranking};


        //By providing the source of the request, it should not query it, to avoid circular query chains
        } else if (req['query'] === 'controlunit') {
            console.log("controlUnit");
            const results = await this.router.findRepositories(req, req['source'], +req['timeout']);
            const cleanedUp = this.removeRejectsAndTimeouts(results);
            return cleanedUp;
        } else {
            return Promise.reject({error: "Must specify query type"});
        }
    }

    //cleans up rejected or timed out requests.
    private removeRejectsAndTimeouts(arr) {
        const results = [];
        arr.forEach(element => {
            if (element !== "rejected" && element !== "timed out") {
                results.push(element);
            }
        });
        return results;
    }

    //recursive flattening of array, borrowed from Mozilla Developer Network.
    private flattenDeep(arr1) {
        return arr1.reduce((acc, val) => Array.isArray(val) ? acc.concat(this.flattenDeep(val)) : acc.concat(val), []);
    }


}
```

Router

```
import * as rp from "request-promise";
import * as fs from "fs";

export class Router {

    private table: Map<string, string>;
    private timeout: number = 6000
    public address: string;

    constructor() {
        this.table = new Map<string, string>();
        const configData = fs.readFileSync("config.json").toString('utf-8');
        const config = JSON.parse(configData);
        this.address = config['address'];
        config['routing'].forEach(route => {
            this.table.set(route['key'], route['address']);
        });
    }

    //takes in a request object and returns a list of places where the data can be found and metadata
    //the only accessible function outside this class.
    public findRepositories(searchParameters, source: string, timeout: number = this.timeout) {
        let finalResults = new Promise((resolve, rejects) => {
            this.requestMetaData(searchParameters, source, timeout).then((results: any[]) => {
                console.log(results);
                let finalResults = [];
                results.forEach(element => {
                    if (typeof element === "string") {
                        try {
                            finalResults.push(JSON.parse(element))
                        } catch (err) {
                            console.log(err.message);
                        }
                    }
                });
                resolve(finalResults);
            });
        });
        return finalResults;
    }

    //Makes a series of http requests to all places that might have the data that is queried
    //once all requests are resolved/rejected/timed out, the promise that it returns, returns a list of results or errors.
    private requestMetaData(searchParameters, source: string, timeout: number) {
        const formDataInput = {
            query: "controlunit",
            location: searchParameters['location'],
            start: searchParameters['start'],
            end: searchParameters['end'],
            type: searchParameters['type'],
            timeout: timeout, //- reduce the timeout for each req. (Not actually used)
            source: this.address
        };
        console.log(formDataInput);

        const options = {
            method: "POST",
            uri: null,
            form: formDataInput,
            headers: {
                'content-type': "application/json"
            },
            time: true,
            resolveWithFullResponse: true
        };

        const promiseArray = [];
        this.table.forEach((value, key) => {
            if (searchParameters['location'].startsWith(key) && source !== value) {
                console.log("request");
                options.uri = value + "/query";
                const dataRequestPromise = this.resolveRejects(rp(options), timeout, value);
                promiseArray.push(dataRequestPromise);
            }
        });

        return Promise.all(promiseArray);
    }

    //wrapper for data request promises
    //this lets us treat the request-promise rejects as resolves, so that a reject or timing out doesn't stop promise.all
    //The intended behavior is that whenever a promise is done or the time is over, the ones that resolves will be used.
    //handle the error from requests elsewhere
    private resolveRejects(promise, timeout, address) {
        const resolves = new Promise((resolve, reject) => {
            setTimeout(() => resolve("timed out"), timeout);
            promise.then((response) => {
```

```
                console.log("Delay: " + response.elapsedTime + "ms from " + address);
                resolve(response.body);
            })
            .catch((err) => {
                console.log(err);
                resolve("rejected");
            })
        })
        return resolves;
    }

}
```

Cost Model

```
interface IntervalValue {
    iStart: number;
    iEnd: number;
    value: number;
}

//Represent a strict constraint for a KPI
class Filter {
    public type: string;
    public direction: string;
    public limit: number;

    public keep(measure: number): boolean {
        return this.direction === "Max" ? measure <= this.limit : measure >= this.limit;
    }
}

//Represents a criteria for the cost model, with a type, a weight of importance,
//and a utility function in the form of a collection of intervals.
//calculateScore() return the score for a single KPI given a measurement of it.
//A criteria can be considered as the utility function plus its weight for a single KPI in the cost model
class UtilityFunction {
    public type: string;
    public weigth: number;
    public valueFunction: IntervalValue[];

    constructor() {
        this.valueFunction = [];
    }

    public calculateScore(measure: number): number {
        let value = 0;
        console.log(measure);
        this.valueFunction.forEach(interval => {
            if ((measure >= interval.iStart && measure <= interval.iEnd) || (measure <= interval.iStart && measure >= interval.iEnd)) {
                if (value < interval.value) {
                    value = interval.value
                }
            }
        });
        return value * (this.weigth / 100);
    }
}

//Main cost model class, holds a collection of filters and cost model criteria and methods to interact with it
export class RankAndSelector {

    private utilityFunctions: UtilityFunction[];
    private filters: Filter[];

    constructor(){
        this.utilityFunctions = [];
        this.filters = [];
    }

    //Adds filters and criteria to the cost model
    public addUtilityFunctionsOrFilters(critieraList, intervalList) {
        let count = 0;
        critieraList.forEach(criteria => {
            if (criteria['constraintType'] === "Soft") {
                const c = new UtilityFunction();
                c.type = criteria['criteria'];
                c.weigth = +criteria['weight'];
                intervalList[count].forEach(interval => {
                    const i: IntervalValue = {iStart: +interval['start'], iEnd: +interval['end'], value: +interval['utility']};
                    c.valueFunction.push(i);
                });
                this.utilityFunctions.push(c);
            } else {
                const f = new Filter();
                f.type = criteria['criteria'];
                f.limit = +criteria['limit'];
                f.direction = criteria['direction'];
```

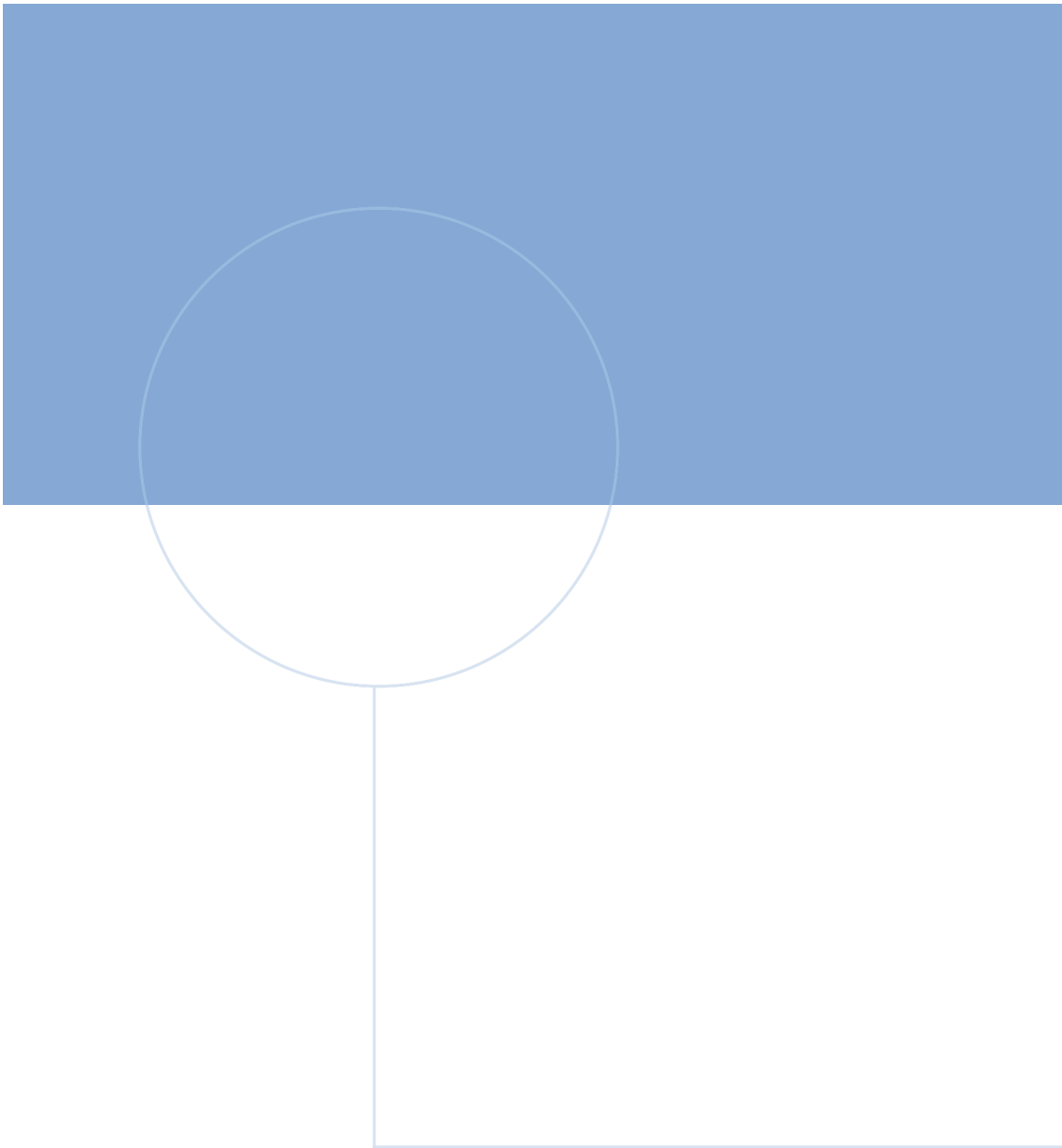
```
                this.filters.push(f);
            }
            count++;
        });
    }

    //Takes in a list of alternatives, and returns a list with all alternatives either ranked or filtered out
    public calculateRanking(results) {
        console.log(results);
        const result = [];
        results.forEach(serverData => {
            serverData['data'].forEach(metadata => {
                if (this.filter(serverData, metadata)) {
                    const score = this.calculateScore(serverData['server'], metadata);
                    result.push({score: score, server: serverData['server'], data: metadata})
                }
            });
        });
        result.sort((a, b) => {
            return b.score - a.score;
        });
        return result;
    }

    //calculates the score for a single alternative
    private calculateScore(serverData, metadata): number {
        let score = 0;
        this.utilityFunctions.forEach(element => {
            //if it is cost, we calculate total cost
            if(element.type === "cost") {
                score += element.calculateScore(this.calculateCost(serverData, metadata));
            } else {
                if (serverData.hasOwnProperty(element.type)) {
                    score += element.calculateScore(serverData[element.type]);
                } else if (metadata.hasOwnProperty(element.type)) {
                    score += element.calculateScore(metadata[element.type]);
                }
            }
        });
        return score;
    }

    //returns false if an alternative breaks a strict constraint
    private filter(serverData, metadata): boolean {
        let include = true;
        this.filters.forEach(element => {
            //if it is cost, we calculate total cost
            if (element.type === "cost") {
                if(!element.keep(this.calculateCost(serverData, metadata)))
                    include = false;
            } else {
                if (serverData.hasOwnProperty(element.type)) {
                    if (!element.keep(serverData[element.type])) {
                        include = false;
                    }
                } else if (metadata.hasOwnProperty(element.type)) {
                    if (!element.keep(metadata[element.type])) {
                        include = false;
                    }
                }
            }
        });
        return include;
    }

    //calculates total cost for an alternative, using access cost and estimated resource cost
    private calculateCost(serverData, metadata) {
        let cost = 0;
        if (metadata.hasOwnProperty("cost"))
            cost += +metadata["cost"];
        if (serverData.hasOwnProperty("resourceCost") && metadata.hasOwnProperty("dataSize"))
            cost += (+serverData["resourceCost"] * +metadata['dataSize']);
        return cost;
    }
}
```



NTNU
Norwegian University of
Science and Technology