Software Tactics for Sustainable Cloud : An Empirical Study

Lappeenranta–Lahti University of Technology LUT

Master of Science (Tech.) in Software Engineers for Green Deal, Master's thesis

2023

Himanshu Pandey

Examiners: Professor Jari Porras (LUT University)

Professor Henry Muccini (University of L'Aquila)

Professor Patricia Lago (Vrije Universiteit Amsterdam)

With the support of the
Erasmus+ Programme
of the European Union

This thesis has been accepted by partner institutions of the consortium (619839-EPP-1-2020-1-FI-EPPKA1-JMD-MOB).

Successful defence of this thesis is obligatory for graduation with the following national diplomas:

- Master of Computer Science (University of L'Aquila)
- Master of Science in Technology (LUT University)
- Master of Computer Science (Vrije Universiteit Amsterdam)

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

LUT School of Engineering Science

Software Engineering

Himanshu Pandey

Software Tactics for Sustainable Cloud: An Empirical Study

Master's thesis

2023

68 pages, 20 figures, 16 tables

Examiners: Professor Jari Porras (LUT University)
Professor Henry Muccini (University of L'Aquila)
Professor Patricia Lago (Vrije Universiteit Amsterdam)

Keywords: Software tactics, Energy consumption, Cloud computing, API proxy, Response time

One of the ways for reducing carbon emissions in cloud computing involves reducing the energy consumption of the software applications running on cloud data centers. This master's thesis focuses on examining the impact of a software tactic on the energy consumption of a software application in the context of cloud computing. The objective is to provide data on the energy-related implications of software tactic adoption, along with the associated trade-offs. Moreover, the study assesses the potential influence of a specific tactic, namely the cloud-native API gateway, on the response time of cloud-hosted software applications. To gather this data, the study employs an empirical experiment featuring a single factor (application type) and two treatments (implementation without the software tactic and implementation with the software tactic). The study spans four distinct user workloads to observe diverse application behaviors. From the results, it can be concluded that since the energy consumption for the two different implementations is based on different energy metrics, a direct comparison is not feasible, therefore it is concluded that there is no significant difference in energy consumption. In relation to response time, there significant difference observed for some workloads and it is crucial to consider this aspect when adopting the software tactic.

ACKNOWLEDGMENTS

I extend my sincere appreciation to my esteemed professors, Jari Porras, Henry Muccini, and Patricia Lago, for their unwavering guidance and support throughout this enriching two-year journey. I am also profoundly grateful to my professor and supervisor, Patricia Lago, for her invaluable mentorship in guiding me through the thesis process. The knowledge and insights gained from this thesis will undoubtedly serve me well in all aspects of my life. I would like to extend a special thank you to Susanna Koponen, whose unwavering support and guidance have been indispensable in making this journey possible.

I express my heartfelt gratitude to Markus Funke for his unwavering support and continuous assistance throughout the thesis. Additionally, I extend my thanks to Ivano Malavolta for providing valuable technical assistance in relation to the empirical experiment. Their contributions have been invaluable to the success of this research.

Finally, I extend my deepest appreciation to my mother and friends, whose unwavering support has been instrumental in my journey to complete my master's. Their encouragement and belief in me have been invaluable, and I am truly grateful for their presence throughout this fulfilling endeavor.

TABLE OF CONTENTS

ABSTRACT **iii**

ACKNOWLEDGMENTS **iv**

1 Introduction **5**

2 Theoretical Background and Related Works **9**

2.1 Cloud Platforms and Cloud Market 9

2.2 Energy 11

2.3 Environmental sustainability and Carbon Emissions 12

2.4 Software Tactics 14

2.5 Related Works 15

2.5.1 Optimizing Data Center Energy Consumption 16

2.5.2 Software Architecture Optimization for Energy Efficiency 17

3 Study Design **19**

3.1 Premise 19

3.2 Study Design 20

3.3 Feasibility Study 22

3.3.1 Inputs 22

3.3.2 Feasibility Results 24

3.3.3 Potential tactic 25

4 Empirical Experiment **28**

4.1 Tactic selection 28

4.2 Research Goal and Questions 29

4.2.1 Goal 29

4.2.2 Questions 30

4.2.3 Metrics 30

4.2.4 GQM-Tree 30

4.3 Subject selection 31

4.4 Variable selection 32

4.5 Experiment Hypothesis 33

4.6 Experiment design 33

4.7 Infrastructure and Configuration 34

4.8 Load Testing 36

4.9 Energy Measurements 37

4.10 Initial tests . . . 37
4.11 Execution . . . 38
4.12 Data Collection . . . 39

5 Results **40**
5.1 Energy Consmption (RQ1) . . . 40
5.1.1 Data Exploration . . . 40
5.1.2 Normality Testing . . . 42
5.1.3 Hypothesis Testing . . . 45
5.2 Response Time (RQ2) . . . 46
5.2.1 Data Exploration . . . 46
5.2.2 Normality Testing . . . 48
5.2.3 Hypothesis Testing . . . 51

6 Discussion **53**
6.1 Energy Consumption (RQ1) . . . 53
6.2 Response time (RQ2) . . . 54

7 Threats To Validity **56**
7.1 Internal Validity . . . 56
7.2 External Validity . . . 56
7.3 Construct Validity . . . 57
7.4 Conclusion Validity . . . 58

8 Conclusion **59**

REFERENCES **60**

LIST OF FIGURES

1 Global Cloud revenue (in billion dollar), 2019-2023 [21] 9
2 Infrastructure as a Service (IaaS) Cloud segment market percentage share for the year 2020 [11] 11
3 Worldwide Energy Consumption by Energy Sources [30] 14
4 Overview of the study design 21
5 GQM Tree [49] 31
6 Software System with and without software tactic implementation 32
7 Sample Locust Workload for 300 users (duration: 1 hour) 36
8 Overview of the experiment execution process for RQ1 and RQ2 38
9 Box-plot: Energy consumption for *implementation A* (without tactic) at different workloads (RQ1) 42
10 Box-plot: Energy consumption for *implementation B* (with tactic) at different workloads (RQ1) 42
11 Histogram: Energy consumption for *implementation A* (without tactic) at various workloads (RQ1) 43
12 QQ plot: Energy consumption for *implementation A* (without tactic) at various workloads (RQ1) 44
13 Histogram: Energy consumption for *implementation B* (with tactic) at various workloads (RQ1) 44
14 QQplot: Energy consumption for *implementation B* (with tactic) at various workloads (RQ1) 45
15 Boxplot: Response Time per workload for *implementation A* (without software tactic) (RQ2) 47
16 Boxplot: Response Time per workload for *implementation B* (with software tactic) (RQ2) 47
17 Histogram: Response Time per workload for *implementation A* (without tactic) (RQ2) 48
18 QQ plot: Response Time per workload for *implementation A* (without software tactic) (RQ2) 49
19 Histogram: Response Time per workload for *implementation B* (with tactic) (RQ2) 50
20 QQ plot: Response Time per workload for *implementation B* (with software tactic) (RQ2) 50

LIST OF TABLES

1 Anonymized employee references 24

2 Goal definition [49] 29
3 Overview of experiment trials 34
4 Infrastructure details of Azure Web App (App Service) 35
5 Infrastructure details of Azure API Management 35
6 Infrastructure details of Azure Virtual Machine 35
7 Infrastructure details of Azure Load Balancer 35

8 Descriptive statistics on energy consumption in watt-Hours for *implementation A* (without software tactic) 41
9 Descriptive statistics on energy consumption in watt-Hours for *implementation B* (with software tactic) 41
10 Shapiro-Wilk test: p-values for datasets of workload 0, 100, 200, and 300 in *implementation A* (without tactic) (RQ1) 43
11 Shapiro-Wilk test: p-values for datasets of workload 0, 100, 200, and 300 in *implementation B* (with tactic) (RQ1) 45
12 Descriptive statistics on response time in milliseconds for *implementation A* (without software tactic) 46
13 Descriptive statistics on Response Time in milliseconds for *implementation B* (with software tactic) 47
14 Shapiro-Wilk test: p-values for data sets of workload 100, 200 and 300 for *implementation A* (RQ2) 48
15 Shapiro-Wilk test: p-values for data sets of workload 100, 200 and 300 for *implementation B* (with tactic) (RQ2) 49
16 Mann-Whitney U test: U-val and p-values for data sets of workload 100, 200, and 300 (RQ2) 51

1 Introduction

As we move into the 21st century, Information and Communication Technology is increasingly integrating into our daily practice [1], mainly through the widespread adoption of smart devices [2]. The smart device category is expanding day by day, and it comprises but is not limited to smartphones, wearables, and health-tech devices, among others [2]. As per the findings of Lee and Circella [1], a rising trend of increased smart device utilization is being observed that co-relates to the growth of the smart devices market. This trend is observed to be an upward trend across Generations. The trend shows that Generation Z tends to devote more time to using these devices than the predecessor generation commonly referred to as Millennials [1] and Millennials more than their predecessor. Furthermore, this trend is projected to persist with future generations.

One of the significant factors behind this trend is the increase in computing capabilities of these smart devices. The low-end devices of today are more powerful than the high-end devices of two decades ago. One of the main reasons behind this is Cloud computing. Cloud computing has substantially impacted the rapid growth of smart devices' computing capabilities while keeping costs low.

Moreover, Cloud computing provides multiple other benefits to these smart devices. It enables these devices to gain remote access to multiple functionalities such as computation, storage, networking, and advanced analytics capabilities, among others. It establishes a pathway that avoids relying solely on physical resources within the device itself. It also paves the way for remote provisioning of these resources. This approach helps overcome challenges related to the cost and efficiency of providing physical resources [3]. By leveraging the Cloud, smart devices can tap into powerful computing resources at a lower cost, increasing their accessibility to the broader population. Taking smartphones as an example, they have become almost as powerful, if not more so, than high-end computers from the past decade [4]. The reason is that today, smartphones can access remote resources required for heavy computation over the internet via the Cloud and provide users with more functionalities at a low price point.

Thus, given the immense potential of Cloud computing, it is also the focus of this research study. Hence, it is crucial to establish a foundational definition of Cloud computing that will serve as a reference point throughout this research. For this purpose, the definition by the American standardization center NIST (National Institute of Standards and Technology)[5] will be utilized throughout this study. According to NIST

> ”Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computer resources (e.g., networks, servers, storage systems, applications, and services) that can be provisioned rapidly and released with minimum management effort or service provider interaction” [5]

According to the previously mentioned definition, the fundamental aspect of Cloud computing lies in the remote provision of computing resources. Furthermore, according to Sunyaev [6], Cloud computing encompasses various features that enable users to leverage shared computing resources over the internet. These resources include services like storage, computing, networking, and more.

In the computing sphere other than that of smart devices, Cloud computing, in general, offers multiple benefits over traditional on-premise computing [7]. One of the significant advantages of Cloud computing is attributed to the scalability and flexibility that Cloud computing offers to its users [8]. Businesses can use Cloud resources to readily adapt and respond to the fluctuations and transitions in user traffic without having to invest in expensive on-premise infrastructure [9]. Cloud computing similarly offers cost savings by reducing capital expenditure and applicable charges associated with managing traditional IT infrastructure [10] by themselves.

Furthermore, Cloud providers typically offer a pay-per-use model, enabling businesses to pay only for the resources they have provisioned for a specific duration or task. Considering its efficient resources utilized, Cloud computing also emerges as a financially viable choice suitable for businesses of any scale. Other benefits of Cloud computing, like improved collaboration and accessibility, enhanced security, backup, and recovery [7] make it a growing industry within the Information and Communication Technology (ICT) segment. Thus, it is no surprise that Cloud computing has an influence on a significant portion of the overall ICT industry, not just the smart device division.

Another sphere that shows the increasing popularity of Cloud computing is its financial market cap. According to a report by Gartner [11] just, the worldwide infrastructure as a service (IaaS) Cloud market segment is estimated to be approx 60 billion dollars in the year 2020. Thus, it establishes Cloud computing as a Multi billion-dollar industry. Moreover, many businesses are migrating to the Cloud every year. According to another report by [12], the majority (95%) of new digital workload is predicted to be hosted on the Cloud by 2025.

As mentioned above, Cloud computing is being adopted as an industry practice at an ever-increasing rate. This adaptation rate presents a significant potential for the growth of the Cloud financial market. However, running all the worldwide Cloud services also requires many physical computing assets. Being scaled up to a billion-dollar industry, these Cloud

assets consume much energy and are responsible for significant carbon emissions. According to the Greenpeace organization, the carbon footprint of cloud computing is increasing every year specially with the help of big oil companies [13]. Moreover, adding to the already existing climate crisis, the carbon emissions generated due to Cloud computing can no longer be ignored. There is an ever-emerging need to make the whole ICT industry more sustainable, and Cloud computing is no exception to it. Though sustainability encompasses many dimensions such as environmental, economic, and societal [14] for the scope of this study, we will focus solely on the environmental dimension, which deals with carbon emissions and energy consumption because they are more quantifiable and directly related to global warming.

Adding to the emissions, the research of Xu and Buyya [15] shows that the primary carbon emission sources due to Cloud computing come from the data centers hosting these Cloud assets. Cloud data centers consume a lot of energy and water to keep their operations up and running [16]. The energy consumed here comes from different production sources, this may come from different types of resources with different carbon intensities. It may include renewable sources of energy in countries like Sweden, Norway, and Finland or may include coal plants and natural gas in countries like Mongolia, Poland, and Indonesia (Electricity Maps[1] for more information).

Therefore, as global warming intensifies worldwide, it becomes crucial for the Cloud computing industry, including the data centers, cloud providers, and its consumers, to address and reduce Clouds carbon emissions actively. The escalating global warming phenomenon [17], which is leading to severe weather phenomena worldwide, highlights the urgency of taking sustainable measures in various sectors, including Cloud computing moving away from energy sources such as oil and coal while at the same optimizing its current energy consumption. Generally, there are two primary ways to reduce Cloud computing energy consumption and carbon emission. The first is to make the data centers more energy efficient [18] and utilize greener energy sources [19]. The second is architecting the software application that runs on these Cloud computing data centers to utilize the Cloud resources more efficiently and consume less energy.

This study primarily focuses on the second method, which involves architecting software applications to enhance their efficiency on the hardware they utilize and reduce energy consumption during operations. Nevertheless, it is crucial to acknowledge that to truly impact the energy consumption of the entire data center, all software applications running on that data center must be made more sustainable, and making adjustments to software architecture to decrease energy consumption is merely the initial step towards reducing the overall energy consumption of global data centers. This can be done by utilizing architectural tactics that

[1] `https://app.electricitymaps.com/zone/MN`

can be leveraged as a guideline to architecting energy-efficient software [20]. In this study, we will explore whether these tactics can have a significant impact on the energy consumption of software applications in the Cloud. Thus, we structure our study as an empirical study to provide quantifiable results that can encourage the adoption of software tactics for architecting cloud-based applications. Moreover, this study involves a collaborative partnership with a financial institution based in the Netherlands, referred to as the ”Financial Institution” throughout this report. The institution's contribution is of paramount importance as it supplies essential information regarding various software applications and software tactics for potential use in the empirical study. However, to maintain confidentiality, the financial institution has requested to remain anonymous.. The question we look to answer here is, What are the implications of employing software tactics on Cloud-based software applications in terms of energy consumption and carbon emissions, and are there any trade-offs related to this employment?

The remaining thesis report is structured in multiple chapters. Starting with Chapter 2, it first provides a theoretical background for ensuring the definitions of essential concepts used in this study are understood by all the readers. Then it goes over some of the research already present in the literature that acts as a reference guide for our study. Chapter 3 will go over our study design, establish the study's premise. Then this chapter will also go over feasibility research done to investigate the feasibility of conducting an experiment in the Financial Institution Cloud environment for studying the energy consumption of different software applications. Chapter 4 will provide details on the experiment setup, variables, different platforms, and tools used during the experiment planning and execution. We will also go over the details of the process of conducting the experiment in this chapters. Furthermore, finally, the results from the experiment chapter are discussed in Chapter 5, and their implications are further discussed in Chapter 6 with Chapter 7 providing an overview of a report on threats to validity. At last, Chapter 8 concludes the study.

2 Theoretical Background and Related Works

This chapter is structured in two parts. The first part will provide the requisite theoretical underpinnings so that readers from diverse backgrounds are aligned in their understanding of the base concepts of the study. The structure of this chapter is as follows the section 2.1 will first go over the basics of Cloud computing. Then we will establish the conceptual framework for energy and its related terminology used in this study in section 2.2. This section will also cover concepts like energy efficiency and energy consumption in Cloud computing. Following that, section 2.3 will explore sustainability and carbon emissions in the context of ICT and Cloud computing, and finally, section 2.4 will provide background information on software tactics for the sustainable cloud. Finally, in the second part of this chapter, we will provide more information related to the existing literature on software tactics in the section 2.5.

2.1 Cloud Platforms and Cloud Market

Today the global Cloud computing market is worth billions, as seen in figure 2. This market share is projected to keep growing in the upcoming years. Considering the widespread adoption of the cloud, studying its energy consumption and impact on the climate is critical. Nevertheless, before we do this, a common understanding of Cloud computing must be established.

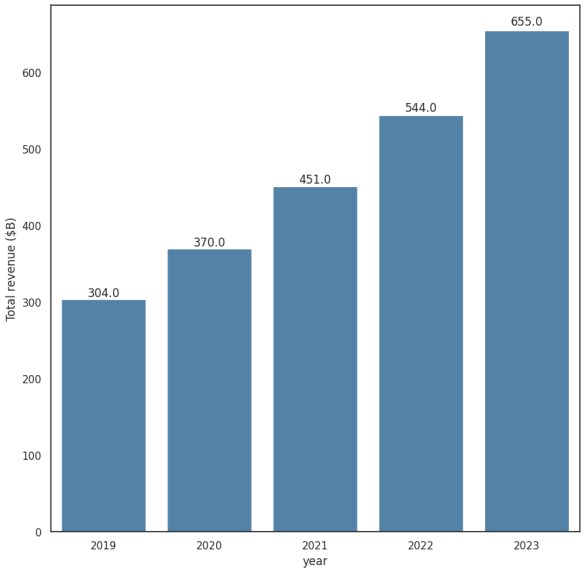

Figure 1: Global Cloud revenue (in billion dollar), 2019-2023 [21]

As previously stated in the introduction, the definition of Cloud computing considered for this study is the one by the American standardization center NIST (National Institute of Standards and Technology) [5]. According to them

> ”Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computer resources (e.g., networks, servers, storage systems, applications, and services) that can be provisioned rapidly and released with minimum management effort or service provider interaction” [5]

Adding on the definition, Cloud computing is commonly employed in public or private Cloud forms [22]. The public cloud refers to a model where third-party Cloud providers offer clients Cloud resources and services. These resources are usually provided over the Internet, and the infrastructure here is usually shared by multiple users who provision these resources according to their needs. Whereas the private cloud is designed to be used solely by one organization, providing greater control, security, and privacy. Both public and private Clouds run on three different service models: Infrastructure-as-a-service, Platform-as-a-service, and Software-as-a-services [23].

Each of these service models addresses specific user requirements and provides distinct benefits. IaaS offers the necessary infrastructure for hosting software applications, typically providing virtualized computing resources like virtual machines, storage, and networking over the Internet. It empowers users to efficiently manage and govern the underlying infrastructure without acquiring and maintaining specific physical hardware [24]. Conversely, PaaS provides a comprehensive Cloud-based environment for development and deployment. It provides users with a platform to deploy effortlessly and test applications without the burden of managing the underlying infrastructure, including virtual machines (VMs) [25]. Lastly, SaaS facilitates the delivery of software applications through subscription-based models, enabling users to access and utilize the software without needing installation or maintenance, as the responsibility for hosting and management lies with the Cloud provider [26].

Each of these service models offers a diverse level of control to the users and thus has different overall energy consumption. In addition, they have different Infrastructure setups under the umbrella of Cloud computing. As a result, the energy consumption of software running on the cloud varies depending on the service provider, platform, and service model. Moreover, it is also important to mention that although there is a diverse range of service offerings in the cloud, only a handful of companies hold the majority market share in the Cloud market. For example, in the Infrastructure as a Service (IaaS) segment, Amazon and Microsoft have established themselves as dominant forces, as indicated by the market share illustrated in Figure 2. When considering the overall market, Amazon, Microsoft, and Google emerged as the leading Cloud providers holding the majority market share in 2020, according to Gart-

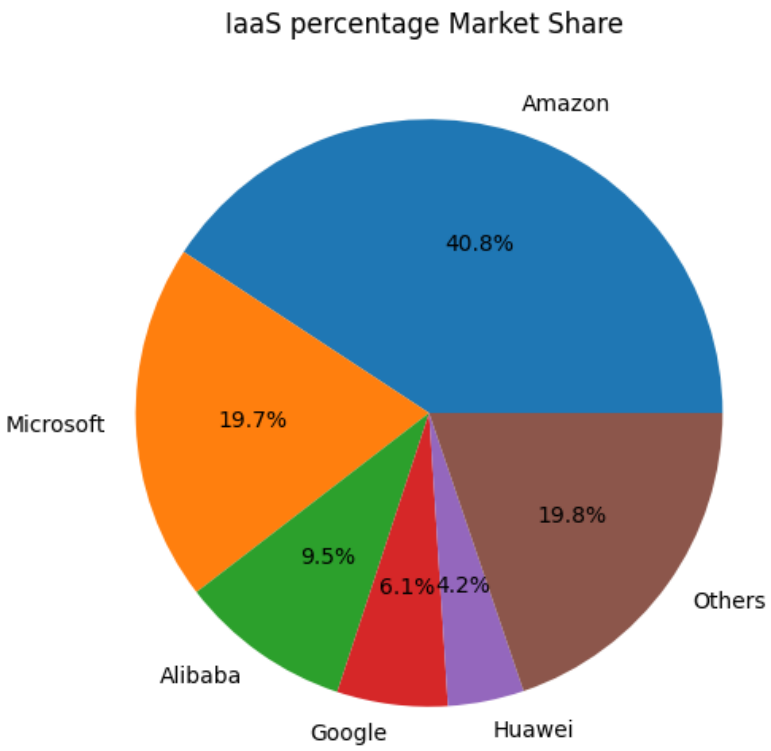

Figure 2: Infrastructure as a Service (IaaS) Cloud segment market percentage share for the year 2020 [11]

ner [11]. This demonstrates that these companies have a significant presence and impact within the Cloud computing energy consumption landscape. Therefore, less competition in the cloud market can lead to less transparent information about performance metrics, carbon emissions and energy consumption. As an example, the dominance of a few companies, such as Amazon and Microsoft, may lead to less transparency in energy consumption data as these companies might not disclose all relevant information due to a lack of competition pressuring them to do so. As such, gaining a comprehensive understanding of the cloud market and its energy consumption landscape is essential for making informed decisions regarding the optimal cloud service provider for hosting software applications.

Coming to the scope of this study, we aim to investigate the energy usage of software applications that utilize Cloud computing as their base. It is essential to note that in chapter 3 we will explore the feasibility of conducting the experiment in Financial Institution X's Cloud environment.

2.2 Energy

This section will go over definitions of energy, energy consumption, and why optimizing a software application's energy consumption is essential.

Every operating machine inevitably consumes energy, which is essential for its functioning, regardless of its power source. Energy, commonly measured in Joules or kilowatt-hours (kWh), represents the amount of work that can be performed by the machine [27]. In this context, energy refers to electrical power consumed over time and measured in watts (W). Notably, 1 kilowatt (kW) is equivalent to 1000 watts [27].

Furthermore, in the context of an empirical study, it is more practical to consider 'energy consumption' as the metric instead of just 'energy.' Energy consumption encapsulates an object's total electrical power during a specified duration [28]. This paper focuses on software applications operating in the cloud, with the duration being their runtime. The preferred unit of measurement is kilowatt-hours (kWh), signifying the energy consumed when 1 kilowatt of power is utilized for an hour. For instance, an appliance with a power rating of 1000W will consume 1 kWh of energy if operated for an hour. To further illustrate the relevance of energy consumption, consider that the average per-capita energy consumption in the Netherlands in 2014 stood at 6713 kWh, as reported by the World Bank [29].

In the context of software, it is essential to note that these software applications do not consume energy directly. They require specialized hardware to run, which consumes energy when hosting these software applications. In the context of Cloud computing software run with the specialized hardware is located in data centers spread across different Continental and regional zones.

In the past years, the energy consumption of these data centers has increased rapidly. According to the EU Commission study [4] in 2018, The consumption of energy by data centers located within the boundaries of the European Union (EU) amounted to 76.8 terawatt-hours (TWh). Projections indicate that by 2030, this figure is anticipated to increase to 98.52 TWh, representing a 28% growth. As a comparison, Croatia consumed 99 TWh in 2021 [30]. Thus it is essential to monitor and design our software applications to consume less energy and be more energy efficient.

2.3 Environmental sustainability and Carbon Emissions

As per the Brundtland Commission of the United Nations [31], sustainability is defined as

> "Meeting the needs of the present without compromising the ability of future generations to meet their own needs [31]."

This definition provides insight into the efficient use of natural, technological, or human resources while maintaining a balance or equilibrium. Using the resources efficiently and in a limited context also gives the resources time to replenish themselves naturally.

In terms of software engineering, this can refer to the technologies related to the software applications, human resources required during the software life cycle, and the energy consumed by these applications. The catalog defined in [32] expresses Software sustainability into five dimensions: social, individual, environmental, economic, and technical.

While the five dimensions have equal importance, we will only consider environmental sustainability within the scope of this research. Environmental sustainability focuses on preserving nature and its natural resources while maintaining the delicate ecological balance of our planet. It encompasses practices and actions that aim to protect and conserve the environment, ensuring its long-term health and viability for current and future generations. The fundamental principles in environmental sustainability are the conservation of resources, ecosystem protection, pollution, and environmental degradation mitigation, climate change mitigation and adaptation, and sustainable land and resource management.

One of the critical aspects of environmental sustainability is managing the emissions of greenhouse gases, including carbon dioxide (CO2), which are inherently present in our ecosystem and play a crucial role in maintaining the Earth's temperature. Under natural circumstances, there is a delicate equilibrium as plants and algae actively absorb CO2 for photosynthesis, which is essential for their growth [33]. However, with the advent of the industrial era, human activities, particularly the combustion of fossil fuels for electricity generation, have significantly increased the concentration of greenhouse gases in the atmosphere. This escalated release of greenhouse gasses has disrupted the natural balance, as the rate at which it is emitted surpasses the rate at which it can be absorbed through natural processes. Consequently, this has led to an enhanced greenhouse effect, contributing to global warming and the associated climatic changes [34].

A significant portion of carbon emissions stems from energy production and consumption, as reported by Yoro et al. [35]. Presently, the majority of energy consumption relies heavily on the combustion of oil, gas, and coal, as depicted in Figure 3. This figure highlights that in the year 2022, fossil fuels constituted the primary energy source, making them the leading contributors to global CO2 emissions and greenhouse gases.

Hence, any device or establishment that relies on energy generated from these resources inevitably adds to global carbon emissions. The ICT industry, including cloud computing, is no exception. Research cited in [36] indicates that the ICT industry accounts for approximately 2.5% to 3.7% of global greenhouse emissions, comparable to the aviation sector. Within the ICT industry, cloud computing plays a role in releasing its fair share of carbon emissions. Consequently, it becomes imperative to prioritize the optimization of software efficiency within data centers to mitigate its environmental impact by either using renewable resources or being more energy efficient.

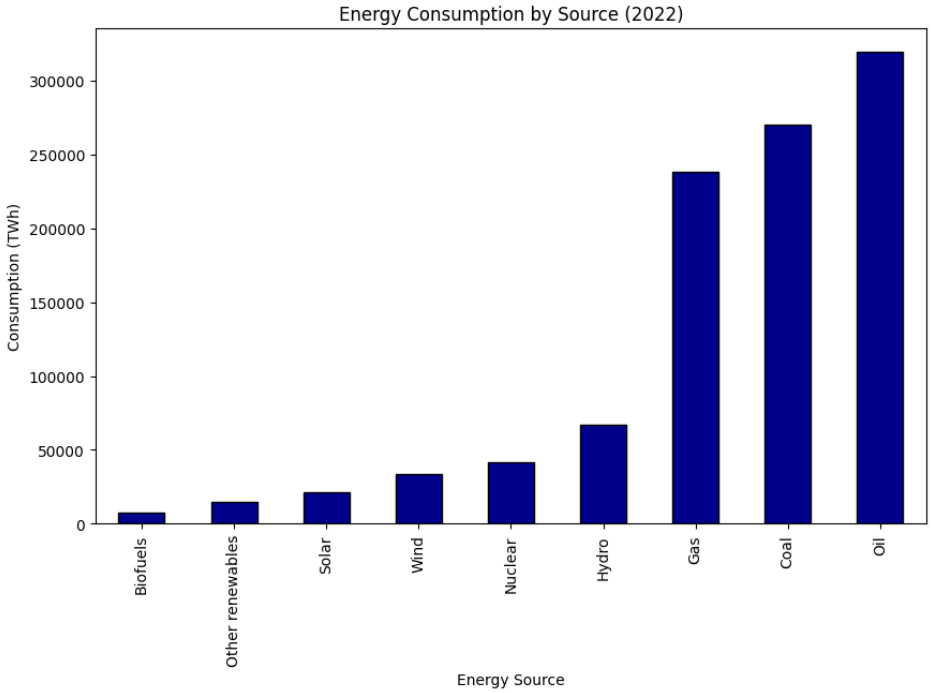

Figure 3: Worldwide Energy Consumption by Energy Sources [30]

2.4 Software Tactics

In their research, Procaccianti *et al.* [20] has established a formalized set of recurrent design solutions, referred to as software tactics, to enhance the energy efficiency of Cloud-based software applications. The tactics are also further defined by them as ”design decisions that influence the achievement of a quality attribute response” [37], the quality attribute here being energy consumption. For the scope of this study, we will focus on architectural tactics that positively affect the energy efficiency of Cloud-based software applications.

We also look at the work done by Vos *et al.* [38] for setting an information base for exploring and understanding software tactics in this study. The authors identified 18 different architectural tactics for Cloud-based software applications, and following the work of Procaccianti *et al.* [20], they have grouped them into three categories that are Resource Monitoring, Resource Allocation, and Resource adaptation.

The first category of tactics incorporates monitoring the Cloud workloads to find areas that need change or optimization. This includes metering energy usage of Cloud components like Virtual Machines and containers used to run Cloud-based applications. This category, while it does not directly affect the application’s energy efficiency, is the baseline for the tactics in other categories, as measurement and identification is the first step to optimizing a problem.

The second and third category deal with the tactics that influence application workload, task assignments, and optimizing the hardware resources utilized by the Cloud applications. Many

resources like virtual machines and containers in the cloud are often not scaled down according to the workload or deallocated when not in use. Frequently, these resources tend to be underutilized; however, due to their provisioning, they continue to consume energy even when idle. The same goes for other Cloud resources such as API gateway, serverless hosting, and security groups, among others. They need to be right-sized such that they are using energy and resources effectively. The most common reason behind this is that businesses have critical requirements like availability and response time which they want to stay above a specific limit even when the demand is low. Thus scaling down is seen as a potential risk even when it is not. Hence this is where software tactics can act as a thorough guide to improve energy consumption while minimizing business risks.

Thus it is essential to establish the impacts of these software tactics on Cloud-based software applications through empirical studies. This can result in solidifying their benefits and help in their widespread adoption. In the next section, we will go over these tactics in more detail.

2.5 Related Works

This section explores various studies centered around investigating the energy measurement of cloud applications. As Cloud computing continues to thrive economically, it becomes critical to address its environmental impacts. One of the pressing concerns in this context is mitigating carbon emissions and energy consumption associated with cloud-based software systems, especially in light of the escalating global climate change. This makes the shift from mere performance optimization to encompass energy consumption reduction even more necessary.

Studies, such as those by Ali *et al.* [39] and Harrison and Avgeriou [40], reveal that historically, Cloud research has primarily been focused on performance optimization, often overlooking the energy dimension. Traditional performance optimization has predominantly been centered around fulfilling business requirements and optimizing software quality metrics such as response time, which led to energy consumption being set as a noncritical requirement for Cloud applications.

Thus to fill this gap, emergent research is beginning to explore the intersection of performance optimization and energy consumption in Cloud applications. However, this body of research is still developing, with a majority being theoretical research and relatively few empirical studies examining the implications of software tactics on energy consumption.

Considering this, we look to analyze some of the studies that incorporate empirical research aimed at optimizing energy consumption in Cloud applications. This section includes two subsections. Subsection 2.5.1, investigates studies not directly related to software tactics for

software architecture and applications but engages in optimizing energy consumption through algorithms or resource optimization strategies within data centers.

In contrast, subsection 2.5.2 focuses on studies that experiment with software tactics for software applications in the cloud as the primary goal of reducing software energy consumption. This section also looks to explore if there are some trade-offs of software applications being more energy efficient.

2.5.1 Optimizing Data Center Energy Consumption

The adoption of Cloud computing has revolutionized computing by offering it as a utility and enabling various applications in scientific, consumer, and business domains. However, this implementation raises significant concerns regarding energy consumption, carbon emissions, and associated costs. This section investigates a few studies that explore the energy consumption of the data centers' resource allocation through different algorithms and methods.

The study Uchechukwu *et al.* [41] looks to explore the concept of Green Cloud Environments (GCE). The research here focuses on providing a generic solution for optimizing Cloud computing data centers while maintaining both dynamic and static components of the cloud as per the business standards. The authors here explore different patterns and find that it is feasible to achieve a 20% reduction in energy consumption in Cloud data centers. The findings of this study can be integrated into Cloud computing systems to monitor energy consumption and facilitate system-level optimization for static and dynamic components. The main outtake from this study is that there are multiple ways of using dynamic and static Cloud components. Through constant monitoring, we can identify the pattern that can reduce energy consumption in Cloud data centers. This technique can also relate to Cloud applications by revealing software design decisions that can reduce the application's energy consumption.

Moving further Subirats and Guitart [42] is another study that investigates energy reduction in cloud data centers. Here the author addresses the interest of IaaS providers in optimizing infrastructure energy efficiency. They propose a mathematical formulation for VM placement algorithms so that they can become more energy efficient via the CPU utilization estimator to assess energy efficiency at different levels. While this study presents results related to energy consumption reduction, there is a lot of dependency on the CPU utilization estimation parameters in this study which convolutes the results related to energy reduction. The takeaway from this study in terms of software application is that applications can be set to use resources such as virtual machines such that they are utilized at a rate that is the most optimum in terms of resources and energy consumption.

Another such study Kumar and Peddoju [43] explores the problem of data centers consuming a significant amount of energy for computing resources and cooling systems. They propose a method to reduce energy consumption in Cloud servers by using Dynamic Voltage Frequency Scaling and adjusting voltage levels to reduce operating frequency during slack time. Though this study demonstrates energy reduction, unlike the studies above, the results here cannot be translated to software applications. In contrast, the paper by Lee and Zomaya [44] focuses on the energy consumption of under-utilized resources in a Cloud environment. The author leverages efficient resource allocation and task consolidation to improve the energy efficiency of the Cloud servers. They emphasize that server energy consumption is directly related to resource utilization and task consolidation. Here the results can be used to understand that through proper optimization, software applications can be made to utilize cloud resources such that these resources can consume less energy.

From the studies explored in this section, it is clear that optimizing resource usage in the context of data centers is a feasible option for reducing energy consumption. And some of the techniques used for the data centers can also be used for software applications.

2.5.2 Software Architecture Optimization for Energy Efficiency

This subsection will explore studies that directly focus on software tactics for reducing the energy consumption of software applications in the cloud. In most of the studies, we found this optimization is done through the means of recurrent design patterns called Cloud Patterns [45].

Cloud patterns are similar to software tactics in terms that they both address recurrent design solutions set on optimizing software quality factors. While software tactics focus on energy consumption as the quality factor, Cloud patterns focus on Cloud service quality factors such as service availability, response time, and throughput, among others. It is important to note that though Cloud patterns theoretically do not affect the energy consumption of an application directly, they do work on optimizing resource efficiency, which may result in more reasonable energy consumption by the software application.

Khomh and Abtahizadeh [46] explores the implications of Cloud patterns on the energy consumption of software applications. The authors consider six Cloud patterns and measure their implication on energy consumption using Power-API. Their study demonstrates that Cloud patterns can effectively decrease energy consumption in Cloud-based applications, although this impact may vary depending on the workload and the application type. The author also observed a trade-off between response time and energy consumption. Another significant result found here is that migrating to micro-service-based architecture can reduce the energy

consumption of an application. This result is similar to one of the software tactics mentioned in [38]: ”Apply granular scaling.”

The paper by Bani *et al.* [47] focuses on the energy consumption of Cloud patterns related to different databases available for Cloud-based applications. The author argues here that despite their widespread use and research on multiple service quality factors, the energy footprint of these databases still needs to be explored. The databases included here for the study are MySQL, PostgreSQL, and MongoDB databases. The results show that the choice of the database significantly impacts the energy consumption of an application. Their findings emphasize the importance of considering energy efficiency in database selection for sustainable Cloud computing.

Another study Abtahizadeh *et al.* [48] tries to investigate Cloud patterns with respect to their energy consumption. The author argues that the digital industry today consumes 10% of the world's electricity. Thus, it is essential to test Cloud tactics not just for optimizing performance but also in terms of energy consumption. The study explores three different Cloud patterns (Local Database proxy, Local Sharding Based Router, and Priority Queue). The author's main results here are similar to the other studies that Cloud patterns can help reduce energy consumption. However, this reduction comes at the cost of software quality aspects such as response time.

As mentioned in the section 2 section 2.4, the work done by Vos *et al.* [38] is used as an information base for this study. The author also presents a case study of edge versus Cloud computing in the context of IoT monitoring systems. The result observed by the author here shows that a 96% decrease in energy consumption can be observed when we move the workload from the edge to the cloud. Our goal in this study is to get similar results from the application of a Cloud tactic. Considering all the information gathered in this chapter, next we will explain the study design and the goals of this study.

3 Study Design

This chapter lays the foundation for the empirical research central to this thesis. In order to maintain a level of consistency throughout the research process, we will utilize a study guide. Here, the study's design is instrumental in ensuring that the research questions addressed throughout the study remain consistent and are backed by reliable and valid results. In this chapter, we outline the context in which the study is conducted and elaborate on the various phases of the study design. This methodical approach is vital for navigating the complexities of examining the sustainability of cloud computing.

This chapter will first lay out the premise of the study design in section 3.1. Following that the actual study design is defined and explained in the section 3.2. Finally the section 3.3 will provide the inputs and the results of the feasibility study done to explore the viability of conducting the experiment in the cloud environment of the financial institution.

3.1 Premise

This master's thesis was done in collaboration with a Financial Institution located in the Netherlands. The financial institution has an ambition to be more environmentally sustainable and to do this they have joined Net Zero Banking Alliance. (NZBA)[1]. They also have an ambition for reaching the goal of net zero carbon emission by 2030 for their own operations. In this context, the financial institution is actively exploring various strategies to enhance the sustainability of its financial and day-to-day operations. As part of this endeavor, the institution is also focusing on its IT department and conducting research to identify opportunities for decreasing its carbon emissions.

In the same context, the software also plays a pivotal role in facilitating the day-to-day operations of the financial institution. It is an integral component that enables crucial tasks such as financial transactions, mortgage approvals, customer verifications, and more. The seamless functioning of these operations heavily relies on the efficiency and reliability of the software systems in place. To this extent, their IT portfolio ranges from homegrown AI and machine learning-based market analysis software tools to COTS like Microsoft Teams. These software applications can be hosted either on-premise, within an organization's own infrastructure, or off-premise, in cloud-based environments. In the context of this thesis, we will explore their cloud portfolio.

[1] `https://www.unepfi.org/net-zero-banking/`

Currently, the Financial Institution is making a significant transition towards a cloud-based environment. In their Cloud portfolio, they have incorporated providers like Amazon AWS[2] and Microsoft Azure[3]. Leveraging these cloud services, the institution hosts their software applications and integrates essential security features offered by these cloud providers to safeguard its operations and data. This shift to the cloud offers the financial institution enhanced scalability, flexibility, and accessibility, enabling them to optimize their IT infrastructure and deliver improved services to their customers.

The Financial Institution has also taken initial measures to enhance sustainability within its cloud portfolio. They have developed a dashboard to actively monitor the carbon emissions of their software applications (refer to section 3.3). Subsequently, the institution aims to identify and optimize software applications for reducing its energy consumption and carbon emissions. However, at the present moment many teams within the institutions IT department are preoccupied with their team sprints and are mostly focused on maintaining their products' business requirements. Thus with given restrictions on time and resources teams are hesitant to try to implement new software tactics that may or may not have a impact on their software applications. They require more research data on the implications of the software tactic in the cloud environment and how the tactic can impact their application (at least on a small scale). This master thesis addresses this gap by providing the financial institution with valuable data on making their software applications more sustainable. Our study will try to provide them with more data by conducting an empirical study to analyze the impact of software tactics on an existing software application in the context of cloud computing. We will also try to study the impact of these software tactics on the software performance parameters to provide the teams with more data regarding the adoption of the software tactic to help them make more informed design decisions.

3.2 Study Design

This section provides a thorough description of the goal of each phase of our study design. A graphic summary of the entire study design can be seen in Figure 4. As seen in the figure, the study is divided into two phases which are explained below.

In **Phase 1** of the study, our objective is to enhance our understanding of the theoretical grounds and the related research necessary to conduct the empirical study. The goal here is to analyze the literature and software patterns related to cloud computing and cloud energy consumption. This includes examining information related to software tactics, patterns, and studies related to measuring the energy consumption of software applications in the context

[2] `https://aws.amazon.com/`

[3] `https://azure.microsoft.com/en-us`

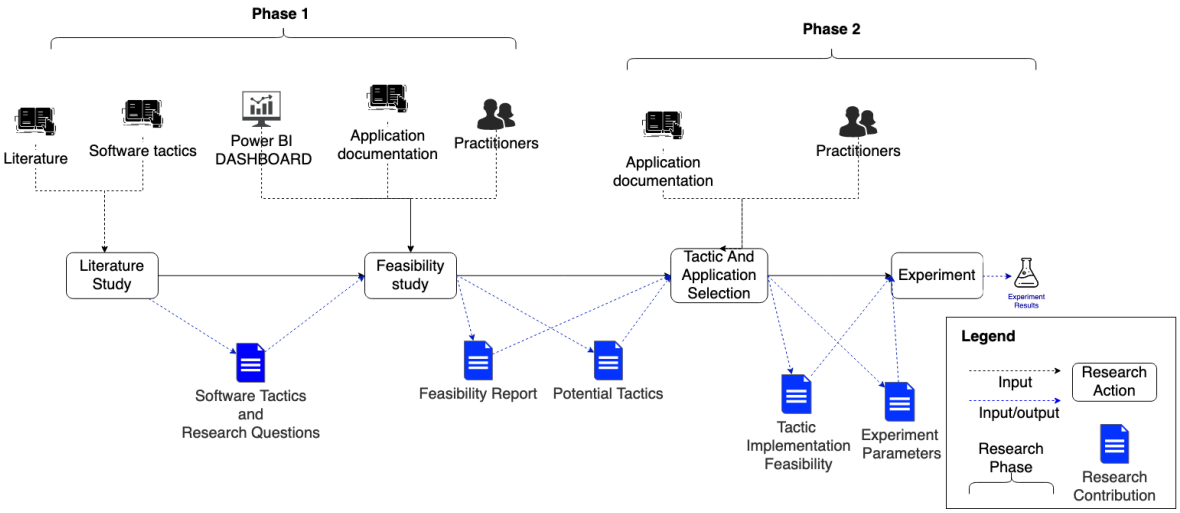

Figure 4: Overview of the study design

of cloud computing. The second half of the previous chapter 2 provides more details on the results of this step. Chapters 2 provides a knowledge base associated with studies on software tactics and energy measurement in the cloud that help us understand how a software tactic can help reduce the energy consumption of a cloud-based application.

After setting up our knowledge base, we perform a feasibility study. The feasibility study enables us to examine what is and is not possible in the context of the study in collaboration with the financial institution. To better understand the possibilities and constraints around the study, we gathered information from different sources in the financial institution, such as power bi dashboards, application documentation, and input from the software practitioners from the IT department at the institution. The feasibility study reveals that, at the time, it is not feasible to execute energy measurement experiments in the cloud environment of the financial institution due to a multitude of factors that will be discussed in the feasibility report in section 3.3. This information is of utmost importance in order to identify and establish the primary research question of this study, as explained in chapter 4. After establishing the information base and examining the experiment feasibility in Phase 1, we carry on to Phase 2 of our research.

In **Phase 2**, we look at the design and experiment related to one of the software tactics identified during the feasibility study. The goal here is to select one specific tactic present in the financial institution's cloud environment and study its effect on the energy consumption behavior of a cloud-based software application. For the experiment planning, we will try to mimic a cloud application that exists in the institution's cloud environment. As discussed in section 3.3, we do not have access to the source code of the software application. Thus, mimicking the software application is the path we will go forward with. The Chapter 4 will

provide more details on how the software application, tactic, and experiment are all set up. Finally, the results of the experiment are shown in Chapter 5 and discussed in Chapter 6

3.3 Feasibility Study

This section presents the findings of the feasibility study, which aims to assess the viability of conducting an empirical study on the energy consumption of a cloud-based software application within the financial institution's cloud environment. The study's structure is as follows: Section 3.3.1 outlines the input resources used for evaluating the experiment's feasibility. Subsequently, Section 3.3.2 analyzes these inputs, determining the study's feasibility, and identifies potential software tactics for the empirical investigation.

3.3.1 Inputs

This section describes the sources utilized as input for the feasibility study. The feasibility study considers three primary input sources for the feasibility analysis: A power-BI-based dashboard, Remarks made by practitioners during the exploratory interviews, and technical documentation of cloud-based software applications.

Dashboards: As previously mentioned, the financial institution is committed to becoming a net-zero carbon emitter. To achieve this, they have established a dashboard for monitoring their software applications, collecting and providing data related to energy consumption in kWh and carbon emissions in MtCO2e. This information encompasses all the software applications hosted on-premise, as well as those on Azure and AWS. Moving forward, the institution aims to incorporate SaaS components into the dashboard as part of their future goals.

The dashboard offers comprehensive information regarding each software application. It presents a holistic view by providing energy consumption and carbon emission data categorized by Year, platform, and department, offering an organization-wide overview. Additionally, it facilitates the visualization of metrics for individual applications.

One of its crucial functionalities is the ability to compare present energy consumption and carbon emissions with historical data, enabling users to monitor changes over time. To enhance user understanding, the dashboard also offers a contextual interpretation of the metrics, relating them to relatable aspects of everyday life, such as phone recharges and air travel. This feature provides a more tangible perspective of the data's impact on real-world scenarios.

An additional critical feature of the dashboard is its capability to display metrics for individual components that host the application, such as Virtual Machines, App Services, SQL

databases, and others. This granular view allows users to assess the energy consumption and carbon emissions of each component independently.

Furthermore, the dashboard offers a ranking system that highlights the highest consumers and emitters among these individual components. This ranking functionality aids users in identifying and focusing on the components that have the most significant impact on energy consumption and carbon emissions. It enables users to target specific areas for optimization and sustainability improvements within the cloud computing infrastructure.

To conduct the feasibility study, the financial institution granted us read-only access to their dashboard. This access was utilized to filter and explore software applications based on their energy consumption and carbon emissions. The study primarily focused on applications hosted on cloud platforms, aligning with the main focus of our research.

Two key filters were applied during the exploration process. Firstly, applications must be hosted on a cloud platform to ensure relevance to our study. Secondly, applications deemed non-business-critical were selected to enable access to documentation related to their architecture. This access is essential for designing the empirical experiment effectively, as it helps eliminate potential roadblocks that could hinder the experiment's progress.

Application Documentation: To continue our feasibility study the financial institution provided us with read-only access to different documentation platforms to support our feasibility study. We used two main platforms for our research. First, a Confluence portal that is used by teams to collaborate, document changes, make information requests and capture knowledge related to their products. Confluence provides dynamic pages, that enable seamless creation, capturing, and collaboration on various projects and ideas. The platform facilitates efficient organization, structuring, and sharing of work, granting easy access to vital information and institutional knowledge. This portal is used for research architectural documentation, and team details related to various software products. It is important to note that our access was restricted to limited information, encompassing high-level architectural documentation, team details, and certain change documentation. The second platform we were granted access to contains details about the institution's internal terms, addressing various hosting platforms, documentation, security levels, and other relevant information.

The information acquired from the portals was used in two steps; first, we used it to find out the organizational structure of an application. Using this information, we can contact IT practitioners to obtain detailed information related to a software application and tactic.

As a second step, we utilized application documentation to examine the high-level architectural details of an application, as elaborated in chapter 4. This data offered insights into

the overall architecture, deployment methods, and current workload of software applications within the financial institution.

Practitioners: The third source of input for our feasibility study consisted of practitioners whom we approached to assess the viability of our study. Table 1 displays an anonymized list of their current positions in the company (please note that this only represents their time in the financial institution in their current positions, and their overall time in the institution may vary).

Reference	Position	Time in position
Employee A	CIO Domain Expert	$\sim 1 years$
Employee B	Business Consultant	$\sim 3 years$
Employee C	Technology architect	$\sim 9 years$
Employee D	Senior Consultant	$\sim 3 years$

Table 1: Anonymized employee references

Employees A and B provided the initial input, highlighting the limitations regarding access to the source code of software applications in the financial institution's cloud environment due to business-critical information restrictions. They suggested that a more feasible approach would involve studying the existing software applications and tactics within the institution's environment and seeking assistance from cloud practitioners to implement the software tactics.

Subsequently, Employee C shared additional information, indicating that most of the IT department's sprints were already planned, and team members wouldn't be available to aid in implementing the software tactic in the financial institution's cloud environment. As a result, we conducted further discussions with Employees C and D, ultimately deciding to undertake a replication study of the software tactics already present in the institution's environment to generate data on its implementation's impact on energy consumption and software performance metrics like response time. This would help in providing practitioners additional details about a software tactic and help in better adoption of the software tactics. Subsequently, with inputs from Employees A, C, and D, we utilized the Confluence portal mentioned earlier to search for potential tactics.

3.3.2 Feasibility Results

This subsection assesses the feasibility study findings, considering the inputs discussed in the previous subsections. The evaluation aims to determine whether conducting an empirical experiment within the financial institutions' cloud environment is viable.

Based on the dashboard, portals, and input from the practitioners discussions conducted, it was found that the IT department at the financial institution is actively trying to make its cloud portfolio more sustainable, especially in terms of environmental sustainability. However, due to the heavy workloads within the department and already defined team sprints, the IT teams will not be able to provide comprehensive support for the empirical study. Without the assistance of developers that engineer and maintain the software applications of the financial institution. With no access to the applications source code the study will have to rely on the high-level architecture documentation provided by application documentation in Confluence portal.

Due to these reasons, we consider that, at this time, it is not feasible to perform an empirical study in the cloud environment of the financial institution. As an alternative, we will perform an empirical experiment in a simulation setting (in public cloud), where we will attempt to recreate the impact of a software tactic present in the environment of the financial institution by mimicking its high level architecture and analyze whether we can observe energy consumption reduction similar to the one observed by practitioners in the IT department.

3.3.3 Potential tactic

Following the findings of the feasibility study it is decided to mimic the software tactic and application present in the cloud environment of the financial institution. The financial institution provided us with 5 different software tactics for our experiment. Tactic 1 is provided by Employee D as an tactic in production in the cloud environment, 2 and 3 were found using the confluence portal and listed as potential tactics not in use. Tactic 4 and 5 were present in the confluence portal as the tactics that the software teams tried to implement but have discontinued since. (see to 3.3). Given below is the list of these 5 tactics, to give them proper representation will follow a similar notation as Vos *et al.* [38] and also map out similar tactics from their paper. The tactics found in their environment are mentioned below.

- **Tactic 1: Use Cloud-specific API Management Gateway instead of Proxy virtual machines.**
  Type: Cloud Tactic
  Category: resource-adaptation
  Mapped tactic: ”Reuse software services” [38]
  Description: When moving software applications to the cloud, it is generally better to use cloud-native solutions compared to the pre-existing non-cloud-native ones. Hence implementing a proxy gateway from scratch for each software application can be replaced by cloud-native services by the cloud providers. In this case, Microsoft’s Azure and AWS both provide cloud-native API gateways. *Relation to energy efficiency:* By

eliminating virtual machines and load balancer-based proxy implementation, each application makes use of a cloud-native application that is hosted as a shared platform. This way, the cloud-native API gateway can reduce the energy consumption of a software application in the cloud. The savings here come from sharing of resources and API gateway being cloud-native and optimized for either Azure or AWS.

- **Tactic 2: Location shifting (parts of) your application to lower carbon intensity and lower impact**
  Type: Cloud (Azure) Tactic
  Category: resource-allocation
  Mapped tactic: ”Select nearby regions with better renewable energy rates” [38]
  Description: Cloud regions with better carbon intensity can be used to host non-critical parts of a software application. *Relation to energy efficiency:* Though the tactic does not impact energy efficiency directly, it can help reduce the carbon emission of a software application.

- **Tactic 3: Time shifting of (parts of) your application to lower carbon intensity and impact**
  Type: Cloud (Azure) Tactic
  Category: resource-allocation
  Mapped tactic: ”Select nearby regions with better renewable energy rates” [38]
  Description: Every cloud region designated to host a cloud server will have different carbon intensities throughout the day. Thus heavy but non-critical workloads can be shifted to a time of the day with better carbon intensity *Relation to energy efficiency:* Though the tactic does not impact energy efficiency directly, it can help reduce the carbon emission of a software application.

- **Tactic 4: Deallocate and schedule auto down of Azure virtual machines**
  Type: Cloud (Azure) Tactic
  Category: resource-allocation
  Mapped tactic: ”Deallocate resources that are not used” [38] *Description:* More often than not, the workload at Financial Institution is time specific, i.e., it has a pattern of being high at certain times of the day and the week and the same for being low. Test and developer environments are less active on weekends and public holidays. Thus, at times of low workloads, such as night or weekends, resources such as virtual machines can be deallocated or put into auto shut down mode.*Relation to energy efficiency:* Resources allocated in the Azure cloud will consume energy even when they are not being used. Thus deallocating them or putting them in auto shutdown can decrease their en-

ergy consumption.

- **Tactic 5: Auto scale down Azure SQL database Service Level Objective (SLO) when not in use**
  Type: Cloud (Azure) Tactic **Category**: resource-allocation
  Mapped tactic: ”Continuously evaluate right sizing” [38]
  Description: SQL Database in Azure consumes energy according to the space allocated for them. Even if most of the database is underutilized, the SQL database will still consume energy according to the space allocated for it. *Relation to energy efficiency:* If a big part of the database is not used, then the allocated space for the database can be reduced or right-sized. The rightsizing here can bring down the energy consumption of the application utilizing the database.

4 Empirical Experiment

This chapter goes over the design and execution details of our experiment as a part of the empirical study. Firstly, section 4.1 will review the tactics defined in the previous chapter and explain our tactic selection process. Then section 4.2 will list our goal, research questions, and metrics in GQM format. Subsequently, the subsequent information is presented in the following manner: the experiment subject is elucidated in Section 4.3, variables are discussed in Section 4.4, hypothesis details are provided in Section 4.5. The experiment design is expounded upon in Section 4.6.

After establishing the experiment design this Chapter will explain and lay out technical details related to the cloud configuration of the experiment execution for application implementation with the software tactic and without it. Section 4.7 will provide details related to the infrastructure used for conducting the experiment. Section 4.8 and 4.9 will convey details on load testing and energy measurement, and finally, section 4.11 and 4.12 will present details of experiment execution and data collection.

4.1 Tactic selection

As mentioned in the previous chapter 3.2, during the feasibility study, we identified multiple software tactics in the financial tactic environment. However, due to the constraints of time, the empirical study can only focus on a single software tactic. In this section, we will narrow down the number of potential tactics to one.

We will narrow down the potential tactics based on criteria that involve metrics such as used in production, available information, and deployment policy. According to the given criteria and the exploratory interviews in the feasibility study 3.3, we have selected **Tactic 1** as **Tactic 2 and 3** are not currently being used in the Azure environment whether it be production, development or testing. **Tactic 4** is dependent on specific Microsoft Azure policies compared to other general software, and **Tactic 5** is highly dependent on the user workload of the financial institution replicating, which can be time consing, costly, thus currently out of the scope of the empirical study. Hence **Tactic 1** is the tactic this study will move forward for the experimentation.

As mentioned in section 3.3 **Tactic 1** deals with cloud-native API gateway. To explain further, many software applications serve as background applications with no front end. These applications are hosted as REST APIs in the cloud environment of the financial institution. Due to security reasons, clients either in the cloud or on-premise are not allowed to call

the API endpoint directly. An API proxy is added between the API consumer and the API provider to mediate this. In general, the API proxy is done via the means of proxy Virtual Machines(VMs) and Load Balancers(LB). This API proxy is application specific, and every API needs its own proxy VMs and LB. The tactic argues that the proxy being non-native can consume more energy than an API gateway that the Cloud Provider offers. The reason behind this is that the API gateway, since it is cloud-native, is optimized for the specific cloud environment that is hosting it. Also, unlike API proxy, which is specific to the application, API gateway is a shared resource that can provide service to multiple APIs simultaneously. Thus the use of an API gateway can eliminate redundant proxy VMs and LBs for multiple APIs at the same time.

4.2 Research Goal and Questions

To give our study a proper definition and interpretability, we use the GQM framework [49]. It follows a top-down approach which includes defining the study in terms of a goal, questions *(RQ1)* and *(RQ2)* , and metrics *(M1)* and *(M2)*. In GQM, the questions help in defining the goal, and the metrics are required to answer these questions. In this section, we will explain our goal, question, metric, and the relationship between them.

4.2.1 Goal

Table 2 provides the definition of our goal in GQM format.

Analyze	software tactic: using cloud-native API gateway
for the purpose of	evaluation
with respect to their	Energy Consumption
from the point of view of	Software Developers
in the context of	cloud-based software application
Result	
Analyze software tactic: cloud-native API gateway for the purpose of evaluation with respect to their energy consumption from the point of view of software developers in the context of cloud-based application	

Table 2: Goal definition [49]

4.2.2 Questions

RQ1: *What is the effect of a software tactic: cloud-native API gateway on the energy consumption of a cloud-based software application?*
RQ1 is our goal's central research question. We selected this approach due to the presence of numerous situations where understanding the impact of software tactics on energy consumption would offer significant benefits. In this case, we want to study the impact of using a cloud-native API gateway instead of non-cloud native API proxies on the energy consumption of software applications energy consumption. This information can be a handful in reducing a cloud-based application's carbon emissions and energy consumption.
RQ2: *What is the effect of a software tactic: cloud-native API gateway on the response time of a cloud-based software application?*
RQ2 is the trade-off analysis for our research question. We want to test here whether the application of the software tactic can affect the response time of a cloud application significantly or not.

4.2.3 Metrics

M1: *Energy Consumption*
As discussed in section 2.2, the metric we have decided to use for the study is Energy consumption. It will be measured in kilowatt-hours (kWh) and will determine how much energy is used for a specific workload.

M2: *Response Time*
Cloud application response time refers to the time it takes for a cloud-based application to respond to a user's request. It measures the elapsed time between the moment a user sends a request and the moment the application completes processing and provides a response and is usually measured in milliseconds.

4.2.4 GQM-Tree

Combining all the information defined above, figure 5 represents our GQM tree. It depicts the relationship between our goal, question, and metric. It also acts as a base for our study.

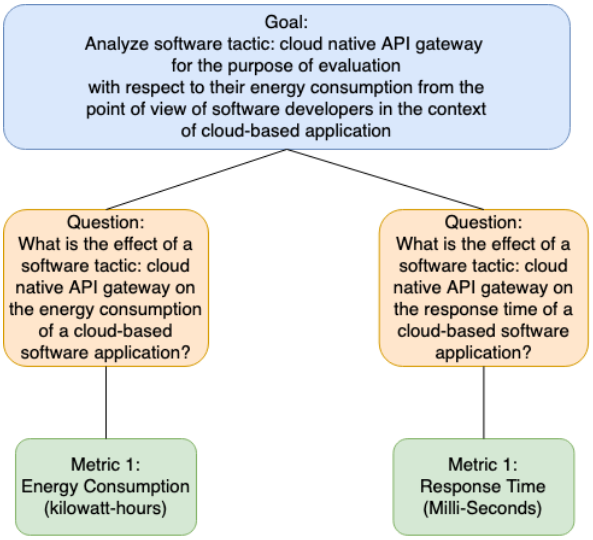

Figure 5: GQM Tree [49]

4.3 Subject selection

In the feasibility study 3.3, it has been determined that replicating a particular application of financial institution one-on-one within our controlled environment is unfeasible due to the unavailability of application code and implementation specifics. To generate a meaningful outcome for financial institution and concurrently provide a broader analysis of the implications associated with employing the software tactic: native API gateway, we will construct a software application that has the same functionality as one of the financial institutions' software application that transitioned from the proxy VMs to a cloud-native API gateway.

As mentioned earlier, the application here functions as a back-end system, exposing a Restful API endpoint. To conduct testing and examine the behavior of the application when utilizing the software tactic of a cloud-native API gateway, we will initially perform tests using the Python-based Flask[1] framework. Flask was chosen for its reputation as a lightweight and versatile web framework, enabling rapid development and prototyping of API designs. This approach aids in understanding how the application behaves in the cloud environment.

Following the initial testing with Flask, the application will be converted to the Java Spring Boot[2] framework. Java Spring-boot framework provides benefits like built-in security features along with features like health check monitoring right out of the box. This choice is influenced by the fact that the financial institution employs a Java-based framework for their REST APIs that implement the given software tactic.

[1] `https://flask.palletsprojects.com/en/2.3.x/`
[2] `https://spring.io/projects/spring-boot`

4.4 Variable selection

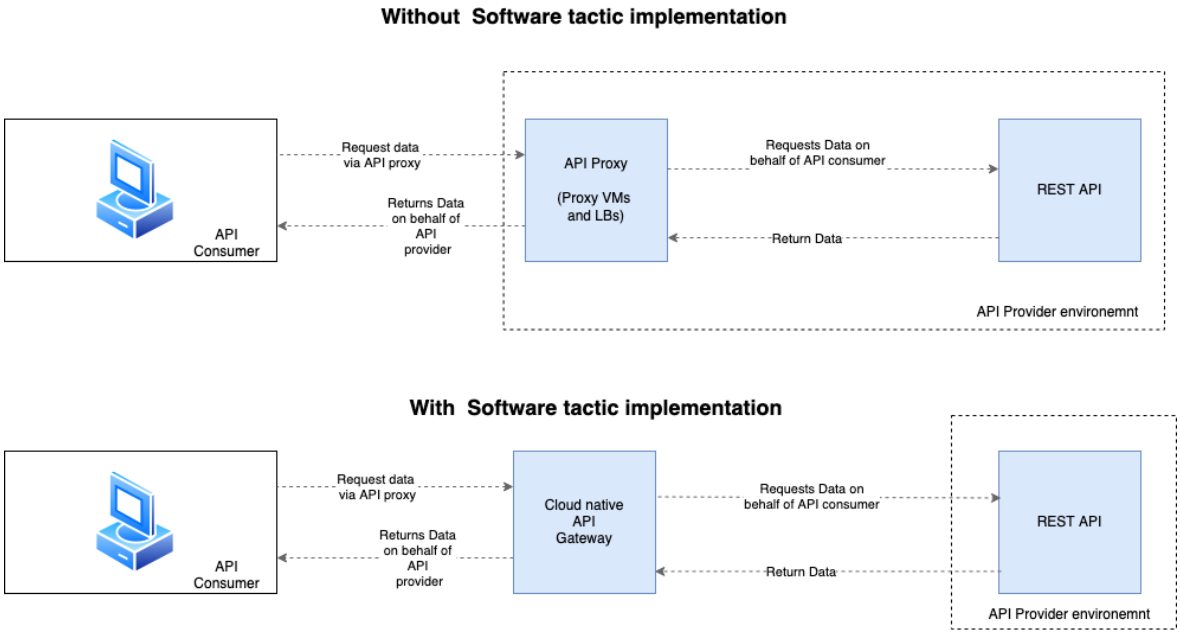

Figure 6: Software System with and without software tactic implementation

Again the goal of the experiment is to compare the energy consumption of the software application software with and without the software tactic: cloud-native API gateway. Thus we will do experiments for two different implementations, one where the API provider and consumer are mediated by API proxy in the form of proxy VMs and the second where the API provider and consumer are mediated by a cloud-native API gateway as shown in the figure 6.

The independent variable for both **RQ1** and **RQ2** is :

- **API-Proxy**: According to how the tactic is implemented, there are two ways of implementing the API proxy. The first one is to do it using a custom application hosted in the application's network using Proxy Virtual machines and a load balancer, and the second one is to use the software as a native service in the form of the API gateway. From this point forward until the conclusion of the study, we will refer to the two implementations as **"implementation A"** (system without software tactic implantation) and **"implementation B"** (system with software tactic implantation), respectively.

In order to be able to tell whether the distinct treatments of the independent variables have an impact on the energy consumption of the Azure server running the application, during the experiment, we keep track of two dependent variables for both of the research questions. The dependent variables are energy consumption and execution time:

- **Energy Consumption**: The energy consumed by the application in the Azure server on which the API and the API proxy run. It is measured using the cloud's carbon footprint. The energy consumption is measured in kilowatts per hour.
- **Response time**: The time it takes for a system or application to provide a response or output after receiving a stimulus or input. It measures the elapsed time from initiating a request or action to completing the corresponding response.

4.5 Experiment Hypothesis

In order to answer the research question **RQ1** stated in 4.2, we have formulated the following pair of hypotheses:

$$H_0 : \tau_1 = \tau_2 \qquad H_1 : \tau_1 \neq \tau_2$$

Given that τ_1 is the mean energy consumption of API proxy implementation with VMs and a LB and τ_2 is the mean energy consumption of API proxy with cloud-native API gateway, the null hypothesis states that the mean energy consumption of using VMs and LB and cloud-native API gateway does not have any significant difference. Subsequently, the alternative hypothesis states that there is a significant difference in mean energy consumption when using VMs and LB versus cloud-native API gateway.

For the research question **RQ2** also stated in 4.2, we have formulated the following pair of hypotheses:

$$H_0 : \mu_1 = \mu_2 \qquad H_1 : \mu_1 \neq \mu_2$$

Given that μ_1 is the mean response time of API proxy implementation with VMs and a LB and μ_2 is the mean response time of API proxy with cloud-native API gateway, the null hypothesis states that the mean response time of using VMs and LB and cloud-native API gateway does not have any significant difference. Subsequently, the alternative hypothesis states that there is a significant difference in response time when using VMs and LB versus cloud-native API gateway.

4.6 Experiment design

Based on Sections 6.2 and 6.3, our experiment takes the form of paired comparison design, where there is one factor with two treatments. The factor is the implementation of the API proxy (Factor-1), which relates to our **RQ1** and **RQ2**. The treatments for Factor-1 are VMs

and LB (Treatment-1.1) and cloud-native API gateway (Treatment-1.2). In addition, to answer the *RQ1* and *RQ2*, we propose one pair of hypotheses, as seen in Section 6.4.

Workload	Number of Trial	RQ	Implementaion
0 users	10	RQ1	*A+B*
100 users	10	RQ1+RQ2	*A+B*
200 users	10	RQ1+RQ2	*A+B*
300 users	10	RQ1+RQ2	*A+B*

Table 3: Overview of experiment trials

Further on, to see the effect of the factor on the dependent variable, we perform the experiment in complete order. Here *complete* means that for each subject, we measure their energy consumption for both treatments. Moreover, these trials are done ten times for four different workloads, as seen in table 3 to again mitigate noise and bias in the experiment. In this context, the workload is treated as a blocking factor, meaning that our primary objective does not focus on studying the energy consumption behavior at different workloads. Instead, our main goal lies with software tactic implantation, and workload variations will not be the primary focus of our investigation. Finally, as seen in table 3, workload 0 is not studied for RQ2 as at 0 user workload, there is no API call, and hence we cannot measure any response time at this workload.

4.7 Infrastructure and Configuration

Given that the tactic used for this study is cloud-based, the software system must be also cloud-based. Consequently, the experiment infrastructure is set up using Microsoft Azure[3] Cloud services. Azure was chosen as the cloud provider for this experiment due to the financial institution's active transition of their software stack to the Azure cloud environment. This decision ensures that Azure Cloud closely mimics the cloud environment of the financial institution, making it the most suitable choice for our study.

The main infrastructure components needed for this experiment are categorized into two groups, referred to as *implementation A* and *implementation B*, corresponding to the application setups without and with the tactic implementation, respectively. For *implementation A*, the experiment utilizes Azure Web App (App Service), Azure virtual machines, and Azure Load Balancer Conversely, For *implementation B*, the experiment runs on Azure Web App (App Services) and Azure API Management Gateway.

[3] `https://azure.microsoft.com/en-us/`

In both application setups, Azure Web App (App Service) is the Java Spring Boot Rest API hosting platform, serving as the back-end application endpoint. Extensive information about Azure App Service can be accessed in Microsoft's official documentation [50]. Additionally, to host the application, the Azure Web App is integrated with the GitHub repository's workflow using GitHub Actions. Detailed configuration information for the Azure Web App is presented in Table 4

Component	Specification
Publish	Code
Runtime Stack	Java 11
Operating system	Linux
Java Web Server Stack	Java SE
Pricing plan	Premium v3 P1V3
Region	West Europe

Table 4: Infrastructure details of Azure Web App (App Service)

Component	Specification
Region	West Europe
Pricing tier	Standard
Monitoring	Application Insights
Virtual Network	Private Endpoint
Protocol	TLS 1.0

Table 5: Infrastructure details of Azure API Management

Component	Specification
Region	West Europe
Image	Ubuntu Server 20.04
VM architecture	x64
Authentication type	SSH
OS disk type	Standard SSD
Monitoring	Boot Diagnostics

Table 6: Infrastructure details of Azure Virtual Machine

Component	Specification
Region	West Europe
SKU	Standard
type	Public
Tier	Regional

Table 7: Infrastructure details of Azure Load Balancer

In *implementation A*, the API proxy is established using Azure Virtual Machines (VMs) and Load Balancer, which are connected to the Azure Web App via the Azure CLI interface. Detailed configuration information for the VMs and Load Balancer is provided in Table 6 and Table 7, respectively. Importantly, all three components, namely Azure Web App, VMs, and Load Balancer, are hosted within the same private subnet. For further insights into Azure VMs, Load Balancers, and Subnets, Microsoft's official documentation [51], [52] offers more comprehensive information.

Finally, in *implementation B*, Azure API Management is employed to serve as the API gateway for the web app, acting as an alternative to the API proxy mentioned earlier. Azure API Management offers a user-friendly interface that connects its private endpoint to the web app's endpoint. Detailed information about this configuration can be found in Table 5. For more in-depth insights into API Management, Microsoft's official documentation [53] provides comprehensive details.

4.8 Load Testing

Cloud applications display diverse behaviors in response to different scenarios, which can involve varying user workloads, hosting tiers, and other factors. One of the primary reasons behind this variability is the limited control cloud users have over the underlying hardware. Although cloud providers offer various hosting options at different price points, most subscribers remain unaware of the specific machines executing their applications. Consequently, conducting multiple load tests becomes essential to minimize the impact of external factors on the test results, such as VM failures, network unavailability, and web service downtime.

As outlined in the earlier Chapter 4, the experiment encompasses ten trials for four distinct user workloads load levels: **0 users**, representing the baseline measurement, and **100, 200, and 300 users**, allowing for the evaluation of the Azure cloud's response under different workloads in terms of energy consumption and response time. This is done to minimize bias and test the cloud tactic under different workload scenarios.

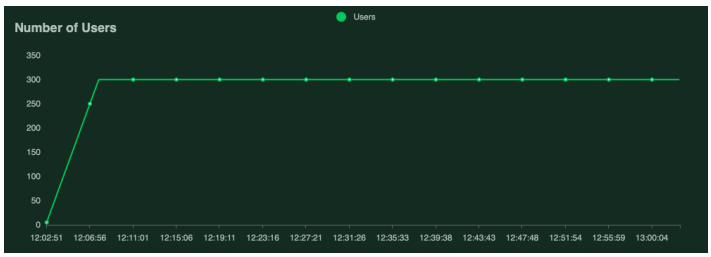

Figure 7: Sample Locust Workload for 300 users (duration: 1 hour)

In this empirical experiment aimed at load-testing Azure Cloud, we will utilize Locust[4], an open-source and scalable load-testing tool developed in Python. Locust allows users to create and customize load-testing scripts using Python, offering the ability to simulate multiple concurrent users, which meets the requirements of our experiment.

Figure 7 depicts a load test from one of the initial tests for 300 users over a duration of 1 hour. The graph illustrates a gradual increase in the number of HTTP GET requests, with

[4] `https://locust.io/`

an additional request per second, reaching a peak of 300 requests per second in 5 minutes. Following the peak, the load test maintains 300 users for the remaining duration of the test.

4.9 Energy Measurements

As discussed in the previous Chapter 4, this experiment's objective is to assess the energy consumption of a cloud-based software application. However, a significant challenge arises when it comes to measuring the application's energy consumption directly within the Azure cloud environment. Azure does not provide specific energy consumption measures for individual applications, offering only general information like CPU utilization, network activity, and other related factors.

To address this challenge, we employ Cloud Carbon Footprint (CCF)[5], an open-source project designed to empower organizations in evaluating, monitoring, and mitigating their carbon emissions resulting from cloud usage. CCF establishes connections with various cloud providers, including Azure, to comprehensively analyze an organization's carbon footprint associated with its cloud services.

This experiment uses CCF to measure the energy consumption of different application setups. The tool provides a detailed guide[6] on how to connect it to an Azure Subscription and the cloud environment. CCF offers energy consumption values on a daily basis, meaning that experiment results for a specific day will be accessible after 24 hours.

4.10 Initial tests

As mentioned in Chapter 3.2, we conducted initial testing using the Flask framework. This preliminary phase was crucial for studying the seamless integration of Locust, Cloud Carbon Footprint (CCF), and Microsoft Azure. This testing taught us that a Python-based script can effectively schedule and coordinate all the load tests. Azure offers extensive support for load testing and CCF, providing valuable insights into the energy consumption of the cloud-based software application.

In the context of CCF, obtaining energy values requires access to the Azure subscription and can provide energy consumption data for a specific day in increments of 24 hours. CCF incorporates a tagging feature to effectively manage various tests, as detailed in the documentation[7]. Additionally, CCF seamlessly integrates with a MongoDB database[8] for efficient data storage and management during the experiment.

[5] https://www.cloudcarbonfootprint.org/
[6] https://www.cloudcarbonfootprint.org/docs/azure
[7] https://www.cloudcarbonfootprint.org/docs/tagging
[8] https://www.cloudcarbonfootprint.org/docs/data-persistence-and-caching

By utilizing Flask and conducting preliminary testing, we successfully demonstrated the successful integration of Locust, CCF, and Azure. This laid a strong foundation for the experiment, enabling us to efficiently perform load testing and measure energy consumption in the Azure cloud environment.

4.11 Execution

The sequence of the experiment execution is depicted in Figure 8. Prior to commencing the execution, it is assumed that the cloud components, including both setups with and without the software tactic, have already been deployed on the Azure cloud and are operational.

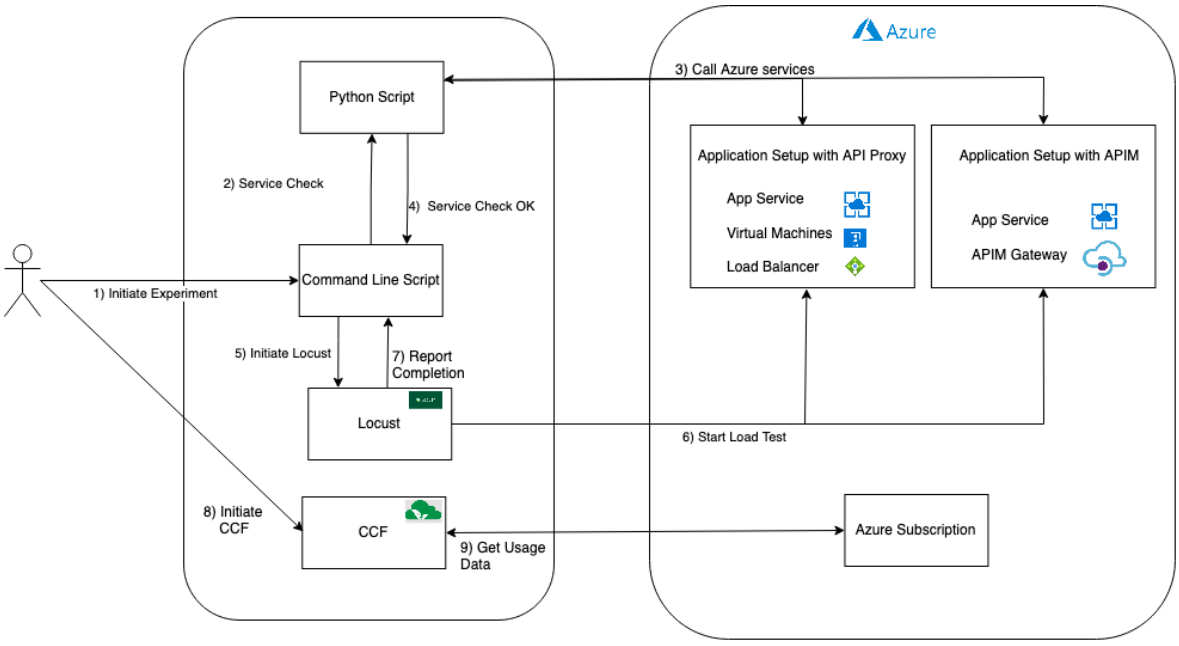

Figure 8: Overview of the experiment execution process for RQ1 and RQ2

The experiment execution begins on the local machine, where the experiment is manually triggered (1), initiating a series of 10 runs for each application implementation (with and without the software tactic) across workloads of **0, 100, 200, and 300 users** (2). For each run, a Python script is utilized to interact with the cloud application endpoints and gather updates on the application setup, accomplished through the Azure CLI Portal (3).

Once the script verifies that the cloud applications are up and running (4), it proceeds to launch the Locust load testing tool (5). Locust then initiates the load test (6) based on the specific application implementation and load test scenario. After the completion of the load test (7), the script stores the results in the form of CSV logs. This iterative process continues until the experiment reaches its conclusion.

Finally, the Cloud Carbon Footprint tool is manually activated (8). Cloud Carbon Footprint collects electricity consumption data (9) from the current Azure subscription and saves it in JSON file format for further analysis.

4.12 Data Collection

During trials 1 through 10 for all workloads, the collected data includes measurements of both energy consumption and response time. Given below are the two sources of data collection that we use in this experiment:

- **Cloud Carbon Footprint (CCF)**. Cloud Carbon Footprint (CCF) generates a comprehensive log file capturing various attributes and information related to energy consumption in cloud-based environments. This log file includes essential records such as the measured energy consumption in kilowatt-hours, the name or provider of the cloud service (in this case, Microsoft Azure), the identifier of the specific cloud service or application under study, and the geographical region where the cloud service is hosted (specifically, Western Europe). These recorded details offer crucial insights for analyzing and understanding the energy consumption patterns of cloud-based applications in the Azure environment throughout the course of the experiment.
- **Load test logs**. After each load test, Locust generates a CSV log file containing comprehensive statistics of the load test, which includes information such as the total number of calls made and the response time for each call. This log file allows for a detailed analysis of the load test results and provides valuable insights into the performance and behavior of the cloud-based software application under different workloads.

5 Results

This chapter presents the outcomes obtained from our empirical study, providing an in-depth analysis of the results. Our primary focus centers on addressing the research questions (RQs) that guided our investigation, specifically RQ1 and RQ2. Sections 5.1 and 5.2 will delve into the outcomes related to RQ1 and RQ2, respectively. Throughout this chapter, the software application implementation without the software tactic will be referred to as **implementation A**, and the implementation with the software tactic will be referred to as **implementation B**.

5.1 Energy Consmption (RQ1)

5.1.1 Data Exploration

As mentioned in Chapter 4, the energy consumption data is initially collected in kilowatt-hours (kWh). However, due to the minimal values involved, we will convert these measurements to watt-hours (Wh) for more practical data analysis. This conversion will facilitate easier comparison and interpretation of the energy consumption data throughout the analysis process. The energy consumption data in this study is obtained using the CCF tool, providing information about the energy consumed by an application hosted in the Azure cloud during a specific load test run lasting **1 hour**. In this subsection, we utilize descriptive statistics and box plots as valuable tools to explore the combined energy consumption results. By employing these methods, we aim to gain a deeper understanding of the data by examining key statistical measures and visual representations. These analytical techniques will provide valuable insights into the energy consumption patterns and variations observed during the experiment.

Using descriptive statistics, we can summarize the central tendencies, variability, and distribution of the energy consumption data, enabling us to quantify and present essential characteristics such as the mean, standard deviation, minimum, maximum, and quartiles. Table 9 and 8 provide descriptive statistics related to **RQ1**. According to these tables, the mean energy consumption for workloads 300, 200, 100, and 0 in *implementation A* is 30.927845, 20.781596, 10.899448, and 0.713500, respectively. For *implementation B*, the mean energy consumption is 11.169774, 7.862242, 4.556599, and 0.495041 for the same workloads. The statistics indicate that energy consumption increases as the workload increases for both implementations (A and B). Furthermore, it is evident that for all workloads *implementation A* consumes less energy compared to *implementation B*. These descriptive statistics provide

	Workload 300	**Workload 200**	**Workload 100**	**Workload 0**
count	10	10	10	10
mean	30.927845	20.781596	10.899448	0.713500
std	5.879529	3.643892	1.147286	0.169281
min	24.486857	14.901226	9.239831	0.552687
1st quartile	26.746582	18.722056	10.400820	0.571372
2nd quartile	28.944467	19.780636	10.835800	0.625651
3rd quartile	33.704397	24.104414	11.461385	0.896580
max	41.158460	26.358570	13.326841	0.923890

Table 8: Descriptive statistics on energy consumption in watt-Hours for *implementation A* (without software tactic)

	Workload 300	**Workload 200**	**Workload 100**	**Workload 0**
count	10	10	10	10
mean	11.169774	7.862242	4.556599	0.495041
std	0.657237	0.355761	0.386163	0.116301
min	10.407417	7.176515	3.705773	0.371143
1st quartile	10.681510	7.671730	4.495275	0.397544
2nd quartile	11.115124	7.944838	4.532046	0.506748
3rd quartile	11.484205	8.024327	4.701437	0.520306
max	12.605337	8.320028	5.254153	0.759399

Table 9: Descriptive statistics on energy consumption in watt-Hours for *implementation B* (with software tactic)

valuable insights into the energy consumption patterns under different workloads and highlight the differences between the two implementations.

We also utilize box plots to visually depict the distribution and variability of the combined energy consumption results. Figure 9 and 10 display boxplots of the energy consumption per trial for *implementation A* and *implementation B*, respectively. These boxplots provide a visual representation showing that the samples for workloads 0 and 100 exhibits a smaller spread than those for workloads 200 and 300 in both implementations *(A and B)*. Additionally, we observe that the trials for Workload and 100 contain an outlier for *implementation A*, and the trials for Workload 0 and 100 each contain an outlier for *implementation B*.

Furthermore, similar to the descriptive statistics presented in Tables 9 and 8, the boxplots confirm that the energy consumption is higher in *implementation A* compared to *implementation B*. These boxplots provide a more precise visual understanding of the energy consumption

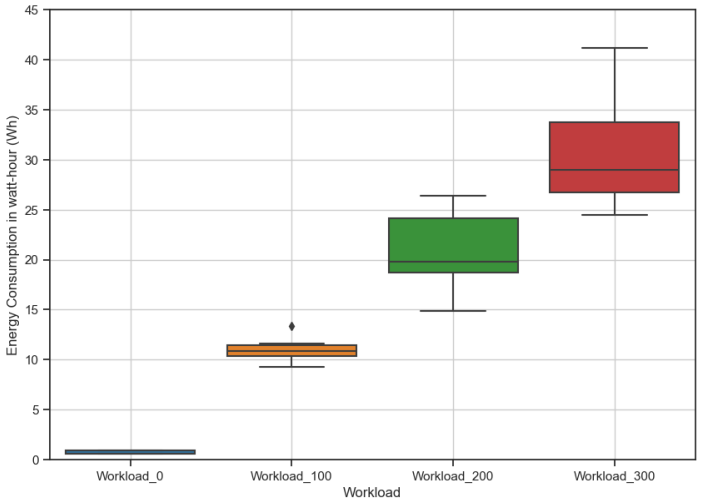

Figure 9: Box-plot: Energy consumption for *implementation A* (without tactic) at different workloads (RQ1)

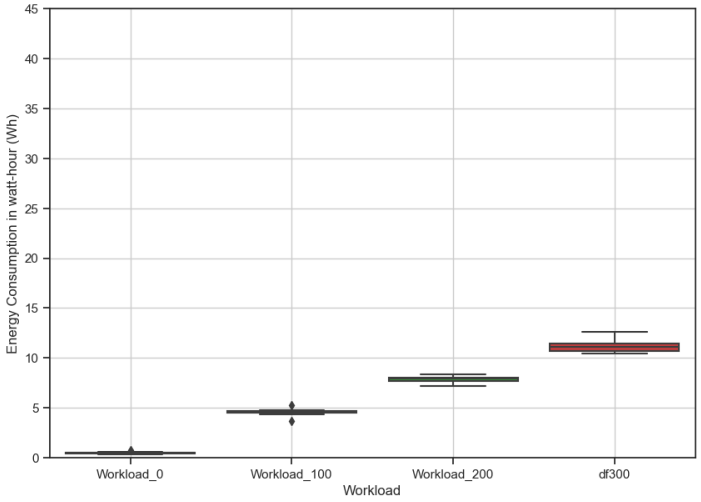

Figure 10: Box-plot: Energy consumption for *implementation B* (with tactic) at different workloads (RQ1)

patterns and variations under different workloads and implementations, corroborating the findings from the descriptive statistics analysis.

5.1.2 Normality Testing

To assess the normality of the sample data, indicating whether it is derived from a normally distributed population, we employ several methods. Firstly, we visually examine histograms and Q-Q plots of the data to assess their distributional characteristics. Additionally, we utilize the Shapiro-Wilk normality test to further support and validate our observations. These techniques collectively help us determine the degree of conformity to normality and ensure the reliability of our analysis.

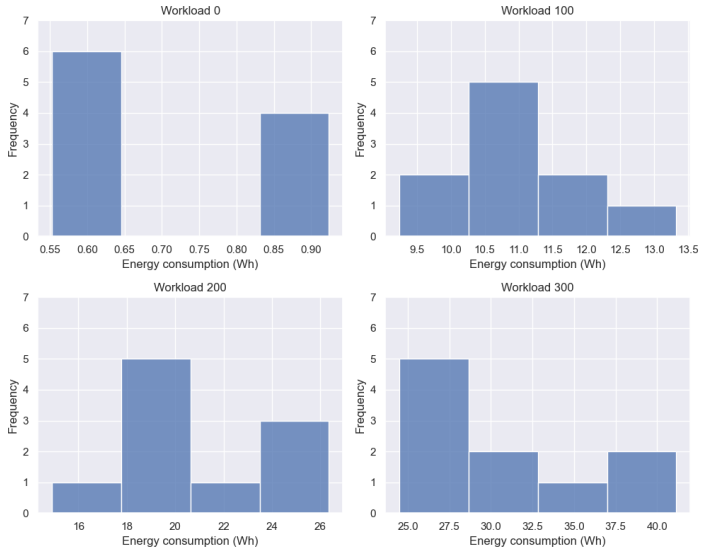

Figure 11: Histogram: Energy consumption for *implementation A* (without tactic) at various workloads (RQ1)

	p-value
Workload 0	0.004381332080811262
Workload 100	0.6375899910926819
Workload 200	0.6256417632102966
Workload 300	0.07239329069852829

Table 10: Shapiro-Wilk test: p-values for datasets of workload 0, 100, 200, and 300 in *implementation A* (without tactic) (RQ1)

For *implementation A*, Figure 13 displays histograms representing the energy consumption for workloads 0, 100, 200, and 300. We utilize the default number of bins provided by the Seaborn[1] Python library. Observing the histograms, we note that workload 0 appears to be divided into two bins, indicating a non-normal distribution. In contrast, workloads 100 and 200 show a tendency to follow a normal distribution, while workload 300 exhibits significant left skewness. Additionally, the QQ plot in Figure 14 reflects similar observations.

To further confirm our findings, we conducted the Shapiro-Wilk normality test, and the results are presented in Table 11. The test outcomes reveal that workload 0 does not conform to a normal distribution. We reject the null hypothesis for workload 0, as the p-value for this workload is less than the chosen significance level of $\alpha = 0.05$.

Figure 13 displays the histograms for *implementation B*, encompassing results for all four workloads. The histograms are generated using the default number of bins according to the

[1] `https://seaborn.pydata.org/`

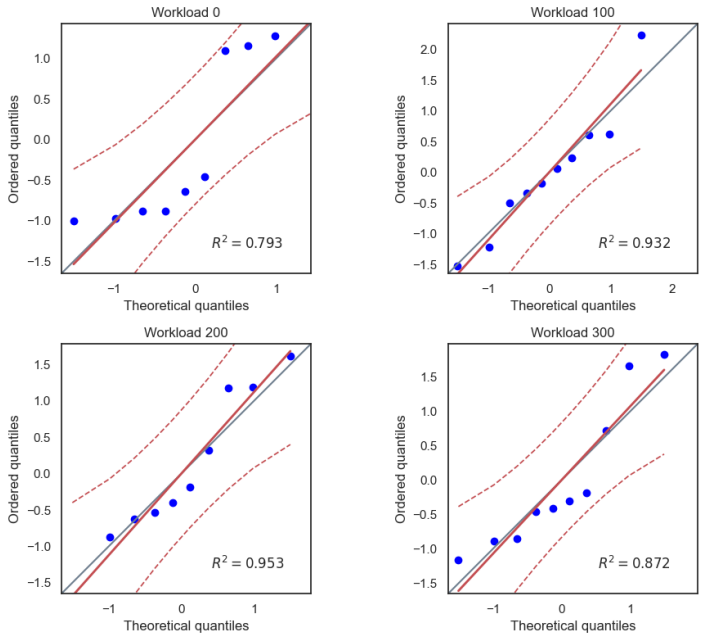

Figure 12: QQ plot: Energy consumption for *implementation A* (without tactic) at various workloads (RQ1)

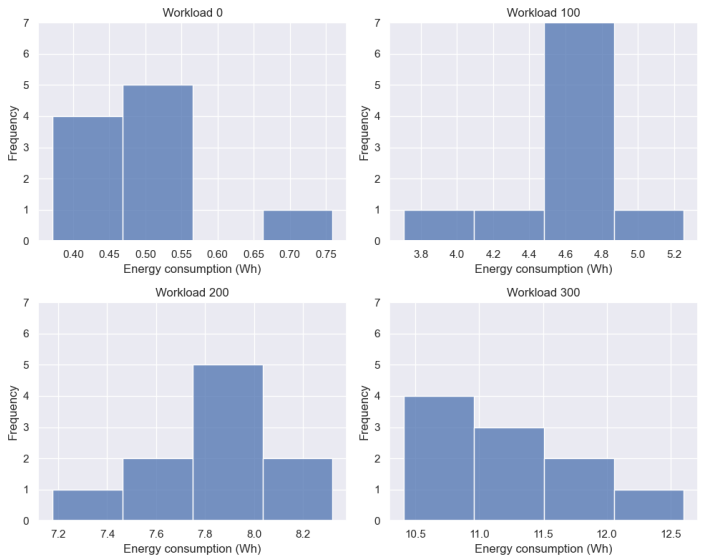

Figure 13: Histogram: Energy consumption for *implementation B* (with tactic) at various workloads (RQ1)

Seaborn[2] Python library. Observing the histograms, we find that workload 200 potentially follows a normal distribution, while workloads 0 and 300 exhibit left-skewed tendencies, and workload 100 follows a more right-skewed curve. The QQ plot in Figure 14 confirms similar results. Furthermore, according to the Shapiro-Wilk test presented in Table 11, we cannot conclude that the data from any workload is non-normal, as we fail to reject the null

[2] `https://seaborn.pydata.org/`

	p-value
Workload 0	0.07736954838037491
Workload 100	0.16774961352348328
Workload 200	0.6502904295921326
Workload 300	0.3330352008342743

Table 11: Shapiro-Wilk test: p-values for datasets of workload 0, 100, 200, and 300 in *implementation B* (with tactic) (RQ1)

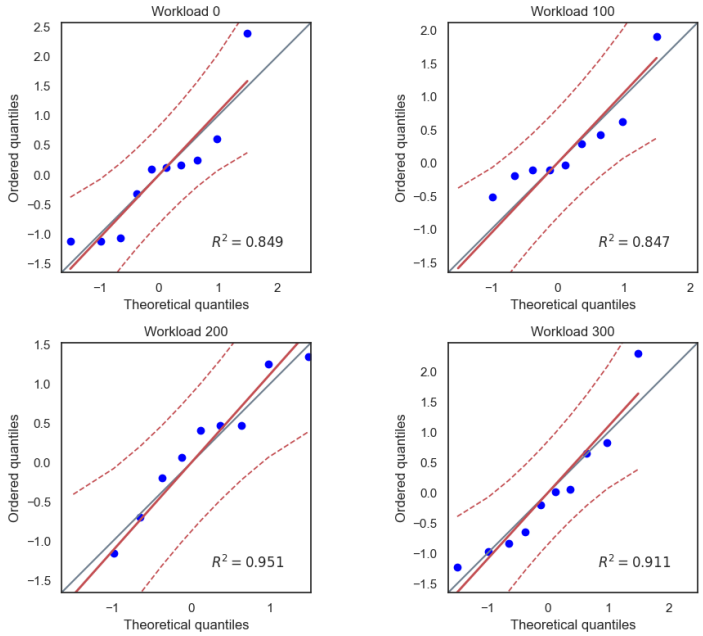

Figure 14: QQplot: Energy consumption for *implementation B* (with tactic) at various workloads (RQ1)

hypotheses that the data is normally distributed, given that the p-values for all the workloads are greater than the chosen significance level of $\alpha = 0.05$.

5.1.3 Hypothesis Testing

While collecting results from the CCF tool, we discovered that although CCF reports energy consumption in kilowatt-hours for both *implementation A and B*, the underlying units of measurement used for calculating energy consumption are different. For *implementation A*, the unit of measurement[3] includes CPU usage, Virtual CPU hours, and the number of API calls. On the other hand, for *implementation B*, the unit of measurement is solely CPU usage. As a result, the energy calculations are based on distinct units of measurement, rendering the energy consumption values incomparable between the two implementations. Due to this discrepancy, we are unable to perform any hypothesis tests on the energy consumption values

[3] `https://www.cloudcarbonfootprint.org/docs/methodology`

of *implementation A and B*. Consequently, we cannot reject our null hypothesis based on the energy consumption data from the CCF tool.

5.2 Response Time (RQ2)

5.2.1 Data Exploration

As mentioned in Chapter 6 the Response time is measured in milliseconds(ms) for RQ2 data analysis. The response time here is measured with the help of the Locust tool, and it represents the average elapsed time from a client's initial request to the receipt of the last byte of the server's response. The response time in this case is measured for a specific (0, 100, 200, or 300 user workload) load test run for a duration of 1 hour. For RQ1 it is important to note that for 0 user workload since there are no API calls being made thus there is no response time to measure. Therefore for RQ2, we will only consider 100, 200, and 300-user workloads.

	Workload 100	**Workload 200**	**Workload 300**
count	10	10	10
mean	71.773267	112.571245	111.971471
std	20.483286	117.612732	61.261481
min	55.772213	53.657279	48.727716
1st quartile	57.478284	58.761193	53.593291
2nd quartile	66.036764	68.841291	98.197887
3rd quartile	74.453275	77.590788	167.095348
max	122.912217	432.029384	201.414093

Table 12: Descriptive statistics on response time in milliseconds for *implementation A* (without software tactic)

For RQ2, similar to the data analysis conducted for RQ1, we will employ descriptive data analysis and box plots. Tables 13 and 12 display the descriptive statistics related to RQ2. These tables present key statistical measures such as mean, standard deviation, minimum, and maximum for *implementation A and B* across three different workloads (100, 200, and 300 users). The observed pattern here is not straightforward. Response time for 200 and 300 users is higher than for 100 users in both implementations, but it is challenging to establish a clear relationship between workload and response time, unlike RQ1.

For comparison between *implementation A* and *implementation B*, it is evident that the average response time of the former is consistently lower for all workloads. This suggests that *implementation A* performs better regarding response time than *implementation B* across varying workloads. The descriptive statistics and box plots provide valuable insights into

	Workload 100	Workload 200	Workload 300
count	10	10	10
mean	266.922674	514.629502	223.842104
std	250.428788	659.833656	142.820143
min	73.919313	111.188599	91.422344
1st quartile	125.369592	126.098919	131.815746
2nd quartile	225.806671	188.929209	153.356298
3rd quartile	276.215460	448.711235	293.186977
max	945.574904	1774.946926	528.712959

Table 13: Descriptive statistics on Response Time in milliseconds for *implementation B* (with software tactic)

the two implementations' response time patterns and differences under different workloads. However, further analysis and interpretation will be required to ascertain the precise relationship between workload and response time.

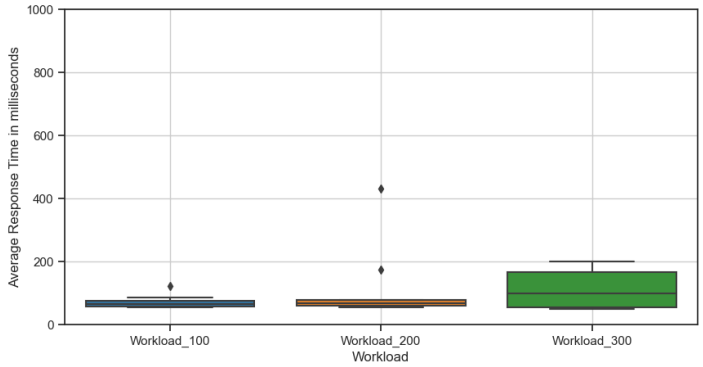

Figure 15: Boxplot: Response Time per workload for *implementation A* (without software tactic) (RQ2)

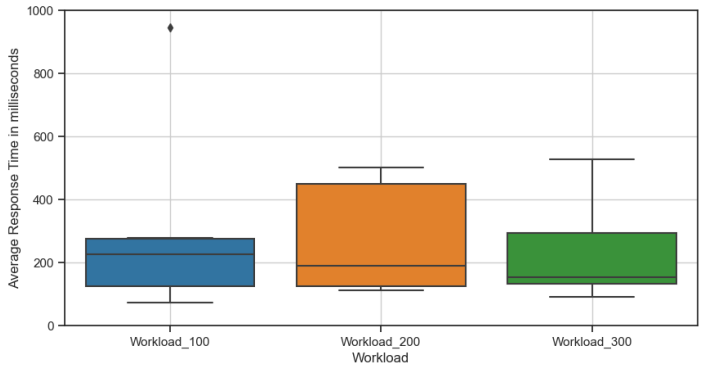

Figure 16: Boxplot: Response Time per workload for *implementation B* (with software tactic) (RQ2)

Similar Results are observed from the box plot. Figure 15 and 16 represent the box plots for response time analysis for *implementation A and B respectively*. For *implementation A*, workload 200 has a better spread than workloads 100 and 300. Workloads 100 and 200 appear to contain some outliers, and it appears that the average response time is not significantly affected by the workload. For *implementation B* workload 300 has a better spread than workloads 100 and 200. Also, it is clear that the Response time of *implementation A* is more than that of *implementation B*.

5.2.2 Normality Testing

In order to evaluate the normality of the sample data, which refers to whether or not it is derived from a normally distributed population, we employ various methods. We visually examine histograms and Q-Q plots of the data to assess their distributional characteristics. Additionally, we utilize the Shapiro-Wilk normality test to support further and validate our observations.

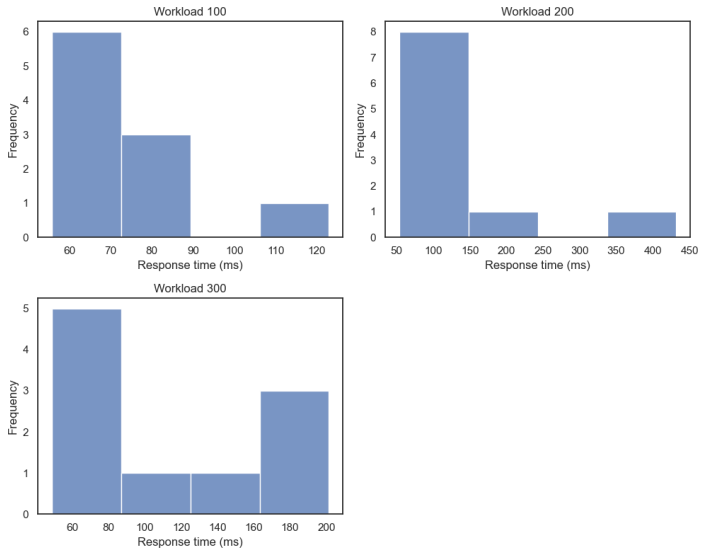

Figure 17: Histogram: Response Time per workload for *implementation A* (without tactic) (RQ2)

	p-value
Workload 100	0.0066637140698730946
Workload 200	1.4894671949150506e-05
Workload 300	0.09393592923879623

Table 14: Shapiro-Wilk test: p-values for data sets of workload 100, 200 and 300 for *implementation A* (RQ2)

Regarding *implementation A*, Figure 17 depicts histograms for the energy consumption at workloads 100, 200, and 300. The histograms for *implementation B* reveal that none of the

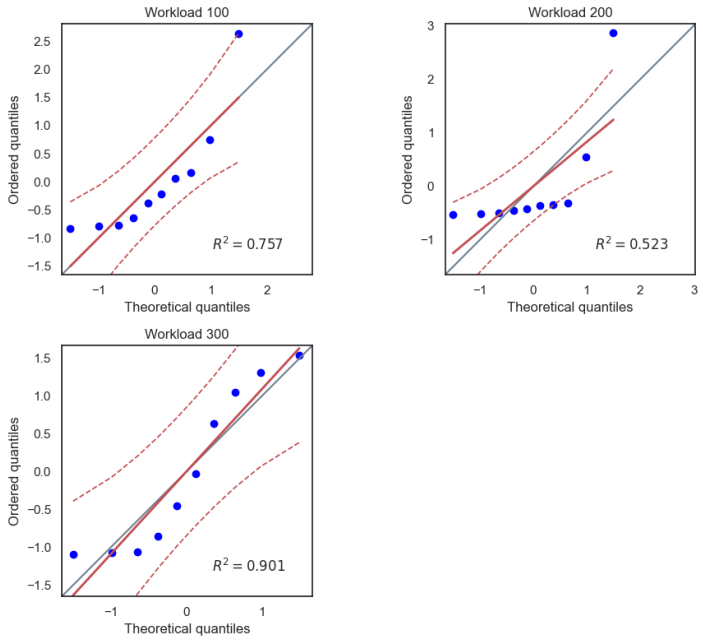

Figure 18: QQ plot: Response Time per workload for *implementation A* (without software tactic) (RQ2)

workloads strictly follow a normal distribution. The QQ plot in Figure 18 further supports this observation.

Additionally, the Shapiro-Wilk test results in Table 14 indicate that the data from workloads 100 and 200 are abnormal. We fail to reject the null hypothesis that the data is normally distributed, given that the p-values for workloads 100 and 200 are less than the chosen significance level of $\alpha = 0.05$. However, it is essential to note that we cannot make the same conclusion for workload 300, as its p-values exceed the chosen significance level. This implies that the data for workload 300 may not significantly deviate from a normal distribution. Nonetheless, the non-normal distribution of energy consumption for workloads 100 and 200 emphasizes the need for cautious interpretation of results and consideration of non-parametric statistical tests when analyzing this particular data.

	p-value
Workload 100	0.00022712466306984425
Workload 200	0.0001700721331872046
Workload 300	0.05499153584241867

Table 15: Shapiro-Wilk test: p-values for data sets of workload 100, 200 and 300 for *implementation B* (with tactic) (RQ2)

Regarding *implementation B*, Figure 19 presents histograms for the response time at all four workloads (0, 100, 200, and 300). Similar to *implementation A*, none of the workloads in *im-*

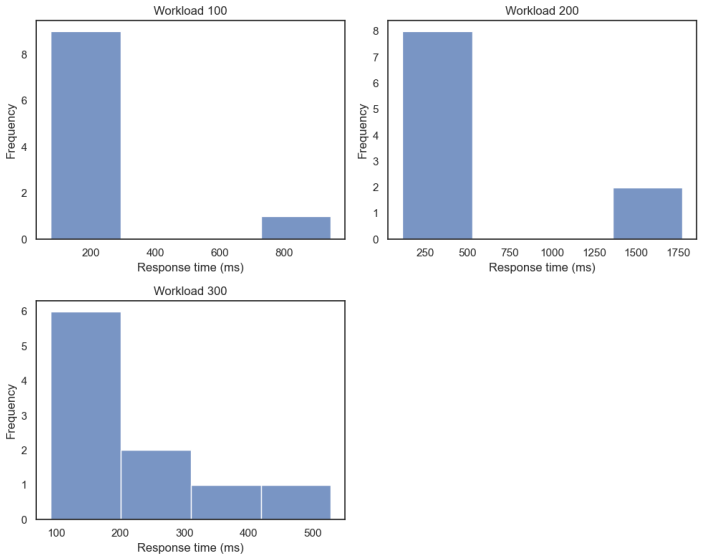

Figure 19: Histogram: Response Time per workload for *implementation B* (with tactic) (RQ2)

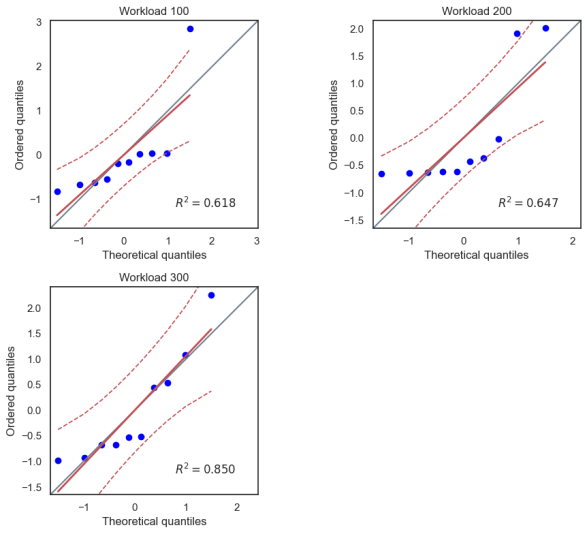

Figure 20: QQ plot: Response Time per workload for *implementation B* (with software tactic) (RQ2)

plementation B follow a normal distribution. Workloads 100 and 200 appear to have outliers, while workload 300 exhibits a left-skewed curve in the histograms.

The QQ plot in Figure 14 reaffirms these observations, showing a departure from normality for all workloads in *implementation B*. The Shapiro-Wilk test results in Table 11 further support the non-normal distribution for workloads 100 and 200, as the p-values for these workloads are less than the chosen significance level of $\alpha = 0.05$.

The non-normal response time distribution for workloads 100 and 200 reinforces the need for careful consideration when interpreting results. It highlights the potential need for non-parametric statistical methods to analyze this data.

5.2.3 Hypothesis Testing

In order to account for the non-normally distributed data for trials 100, 200, and 300 in both *implementation A and B*, we have opted to use the non-parametric Mann-Whitney U test to test our hypotheses statistically. This choice was made because the data sets of these trials violate the assumption of normality required for the one-way ANOVA test.

To ensure that we adhere to the assumptions of the Mann-Whitney U test, we verify the following criteria:
The dependent variable should be measured at the ordinal or continuous level. The dependent variable is the response time measured in milliseconds and is measured at the continuous level.
The independent variable should be two categorical, independent groups. The independent variable is the API proxy implementation and has two unique treatments: Azure API management and API proxy using VMs and Lbs.
The observations should be independent of each other. The observations between groups are independent, API proxy strategies of the two groups are different.

By satisfying these assumptions, we can use the Mann-Whitney U test to compare the response times between the two API proxy implementations, providing a valid and robust statistical analysis despite the non-normal distribution of the data.

User Workload	U-val	p-value
100	95.0	0.00076
200	88.0	0.00458
300	76.0	0.05390

Table 16: Mann-Whitney U test: U-val and p-values for data sets of workload 100, 200, and 300 (RQ2)

Based on the results from the Mann-Whitney U test shown in Table 16, we observe that the p-values for workloads 100 and 200 are less than the chosen significance level of $\alpha = 0.05$, while the p-value for workload 300 is slightly higher than $\alpha = 0.05$.

For workloads 100 and 200, we reject the null hypothesis, indicating that the mean response time of using Proxy VMs and LBs and cloud-native API gateway is not the same, i.e., the

mean response time of *implementation A* (without tactic) and *implementation B* (with tactic) is not the same. This suggests significant response time differences between the API proxy implementations for workloads 100 and 200.

However, for workload 300, we fail to reject the null hypothesis, indicating that we do not have sufficient evidence to conclude that there are significant differences in the response times between the two API proxy implementations for this workload.

The Mann-Whitney U test results demonstrate significant differences in the mean response times for workloads 100 and 200 between *implementation A* and *implementation B*. In contrast, no significant differences are observed for workload 300.

6 Discussion

This chapter revisits the research questions and delves into the implications of the findings presented in the preceding chapter. The following sections provide a thorough discussion of the results and offer answers to the sub-research questions RQ1 and RQ2.

6.1 Energy Consumption (RQ1)

RQ1: What is the effect of a cloud tactic: cloud-native API gateway on the energy consumption of a cloud-based software application?

Software tactics play a vital role in guiding software practitioners toward designing energy-efficient software applications. RQ1 focuses on exploring these software tactics within the context of cloud computing. The primary objective behind RQ1 is to conduct an empirical study that investigates software tactics' impact on applications hosted in the cloud. The significance of this empirical study lies in providing statistically significant results that support the efficacy of software tactics, thereby facilitating their wider and more rapid adoption in the industry. By establishing an empirical foundation, this research contributes to the advancement of energy-efficient software development practices in cloud-based environments.

The chapter 5 presents the results of the empirical study related to the energy consumption of the two different software *implementation A and B* that represent implementation without and with software tactic.

During the experiment, we explored the integration policies of Microsoft Azure with Cloud's carbon footprint. The experiment is set up to examine the energy consumption of *implementation A and B* at different workloads (0, 100, 200, and 300 user workload) for 10 trials per workload. This is done to minimize bias and examine the behavior of the given software tactic at different workloads.

During the data collection process, it was observed that Cloud Carbon Footprint (CCF) presents energy consumption results in kilowatt-hours for both *implementation A and B*. However, the internal parameters used by CCF to calculate this energy consumption are different for the two implementations. For *implementation A*, which utilizes Proxy VMs and LBs, CCF uses CPU percentage utilization metrics in its calculations. On the other hand, for *implementation B*, which utilizes Azure API Management, CCF uses virtual CPU hours and the number of API calls as input parameters for energy consumption calculations. As a result, the output energy consumption values for the two implementations come from different sources.

Due to the differences in input parameters and the nature of energy consumption calculations for *implementation A* and *implementation B*, it is not appropriate to perform a hypothesis test on RQ1. Consequently, we are unable to reject the null hypothesis. Therefore, RQ1 concludes that there is no significant difference between the energy consumption of the software application implementation with and without software tactic implementation as we cannot reject our null hypothesis. Additionally, considering the difference in the energy consumption observed during the data analysis process we conclude that the software tactic: cloud-native API gateway may have an impact on the energy consumption, however, to test it scientifically we need more transparent measurement metrics from the cloud providers offering the cloud-native API gateway service. This finding suggests potential areas for further exploration, especially considering Microsoft's commitment to enhancing the transparency of its cloud environment in terms of service metrics in the future. Additionally, as the Cloud Carbon Footprint (CCF) tool continues to develop, it could provide valuable insights and more comprehensive data for future investigations in the field of energy efficiency in cloud-based software applications.

6.2 Response time (RQ2)

RQ2: What is the effect of a cloud tactic: cloud-native API gateway on the response time of a cloud-based software application?

Software tactics are crucial in guiding software practitioners to design energy-efficient software applications. RQ2 specifically aims to investigate the impact of these software tactics on software performance metrics, in this case, response time. By analyzing the response time performance of software applications in the cloud, this research seeks to gain valuable insights into how software tactics can influence response times in cloud-based environments. The findings from RQ2 can contribute to providing software practitioners with more information about the given software tactic that can aid them in making more informed design decisions.

The second half of chapter 5 presents the results of the empirical study related to the response time for the two different software *implementation A and B* that represent application implementation without and with software tactic. For RQ2, we observed that most workloads don't follow a normal distribution. We also observed that the response time for *implementation A* is less than *implementation and B*. Furthermore, we did the Mann-Whitney U test to answer our RQ2. The test results show that for workloads 100 and 200 there is a significant difference in the response time between the software application without and with the software tactic implementation. Thus for workloads 100 and 200, we reject our null hypothesis that states that the mean response time of using Proxy VMs and LB and cloud-native API gateway does

not have any significant difference. While for workload 300 the tests show no significant difference and we cannot reject our null hypothesis. Therefore, we answer our RQ2 as yes, there is a significant difference in the response time due to implementing a software tactic in the context of cloud computing. However, this statement is only valid for a workload of 100 and 200 users per hour and not for 300 user workload.

Therefore, when adopting the tactic cloud-native API gateway it should be noted that the tactic adoption may impact the response time of the cloud-based software application significantly. Before adopting the tactic the application will require more load testing for a higher number of trials to examine whether the new response time due to the tactic adoption is within the performance requirements or not.

7 Threats To Validity

This chapter is dedicated to identifying potential threats to the validity of our study. To achieve this, we employ a classification framework proposed by Cook and Campbell in Cook *et al.* [54], further elaborated by Cook *et al.* [55]. The following four sections delve into our study's internal, external, construct, and conclusion validity, addressing the respective threats associated with each type of validity. By thoroughly examining these aspects, we aim to ensure the robustness and reliability of our research findings.

7.1 Internal Validity

Regarding internal validity, it is crucial to address potential factors that could influence the causal relationships between variables in our study. Cloud computing, a shared service with dynamic resource allocation and hidden underlying hardware, introduces the possibility of external factors such as power supply variations, cloud server conditions, or maintenance activities affecting our experiment results. To mitigate these threats, we implemented several precautions: We conducted multiple test runs to account for potential fluctuations in the cloud environment. All experiment trials were performed in the same cloud region with identical configurations to minimize variations in the setup. We carefully ensured that the trials ran only when we could confirm the responsiveness of the cloud applications, considering that cloud resources like virtual machines may enter auto-shutdown mode if idle for extended periods. These measures helped reduce the likelihood of external factors impacting our results and strengthen the internal validity of our study. Another possible threat could be the maturation, as the experiment was repeated several times over the same applications. To mitigate this effect, the cache and cookies were cleaned for locust and cloud carbon footprint after each run. Thus, we ensured that no stored data could affect the accuracy of the energy consumption measurements of the following run.

7.2 External Validity

The study's external validity may be impacted by the limited scope of subjects' selection. Due to the numerous combinations of tactics, applications, and time constraints, it was only feasible to experiment with some possible scenarios. Consequently, the chosen application and software tactic was based on specific criteria to ensure the practicality and comparability of the results.

(1) Accessibility was a crucial factor in the selection process. Insufficientneeded to be more information about the software application and tactic could result in a setup that deviates significantly from the practical industry setup used by financial institutions. This could limit the generalization of the study's findings to real-world scenarios.

(2) Functionality was another significant criterion. Both software application versions needed the same functionality but different architectural implementations to ensure that similar tasks could be performed, allowing for a fair comparison of the results.

(3) Utilization was considered to ensure the relevance of the chosen software tactic. Selecting a tactic actively used in production would make the study more meaningful, as it directly relates to the applications used in practical settings.

While these criteria guided the selection process and were essential for the feasibility of the study, it is essential to acknowledge that the results may only be universally applicable to some possible combinations of software tactics and applications used in different cloud environments. The study's findings should be interpreted within the context of the specific software tactic and application chosen for investigation.

7.3 Construct Validity

One of the main threats to external validity in this study is the inadequate preoperational explication of constructs. This means that the constructs and theory used in the experiment might need to be more well-defined and clear. To mitigate this threat, we carefully planned the experiment before its actual execution: We employed a well-defined study design to serve as a knowledge base for the experiment. We defined the goal, research question, and metrics using the Goal-Question-Metric (GQM) framework. We precisely selected and described the independent and dependent variables, treatments for the independent variable, hypothesis, and the statistical tests used for data analysis.

To mitigate the mono-operational bias, where the experiment relies solely on one treatment for a single factor, we applied two treatments of the same factor on four different workloads. Additionally, each trial was repeated ten times to ensure more robust results.

A potential threat to the study's internal validity is the mono-method bias, as only one energy consumption measurement tool (Cloud Carbon Footprint) was utilized. This could limit the diversity of measurements and potential perspectives on energy consumption. To address this issue partially, each trial was repeated multiple times, and the reliability of the measures was evaluated to enhance confidence in the results. Unfortunately, using another energy consumption measurement tool was not possible due to time constraints and other limitations, such as the lack of integration with the Azure cloud environment.

Despite the efforts to mitigate these threats, researchers should be cautious in generalizing the study's results beyond the specific context of the chosen software tactic, application, and cloud environment. The findings should be interpreted and applied within the limitations of the study design and the specific conditions under which the experiment was conducted.

7.4 Conclusion Validity

Indeed, low statistical power is a common threat that can impact the ability to detect actual effects in an experiment. In this study, we addressed this threat by ensuring a sufficient sample size. By applying two treatments on four different workloads for ten repetitions each, resulting in a total of 80 hours of experiment, we aimed to have enough data points to conduct reliable statistical analyses. This larger sample size increases the likelihood of identifying significant effects if they exist.

Furthermore, another threat to validity is the violation of statistical test assumptions. To mitigate this concern, we took a proactive approach. Before conducting the statistical analyses, we predefined the specific tests that would be used based on the research questions and the nature of the data. Additionally, we thoroughly checked whether the assumptions required for each test were met. If the assumptions were not satisfied for a particular test, we would have needed to choose a more appropriate test or consider alternative analytical approaches. This attention to validating assumptions helped ensure that the chosen statistical tests were appropriate for the data and that the results were interpreted accurately.

8 Conclusion

In this thesis, we focused on investigating software tactics for sustainable cloud computing through an empirical study conducted in collaboration with a financial institution based in the Netherlands. The goal of this study is to equip the financial institution with data that can assist them in implementing software tactics within their cloud environment, thereby curbing energy consumption. To achieve this goal, we commenced with an exploratory literature review exploring the concepts of cloud sustainability, cloud software tactics, and relevant information. Additionally, we evaluated the feasibility of experimenting within the cloud environment of the financial institution. Subsequently, we executed an empirical experiment in a controlled setting to examine the impact of utilizing a cloud-native API gateway as an alternative to Proxy VMs and LBs on the energy consumption of a cloud-based software application. The study aimed to shed light on the potential benefits of adopting specific software tactics for energy efficiency in cloud environments.

Based on our findings, it is essential to acknowledge the limitations arising from the measuring units of energy consumption for both software application implementations, with and without the tactic. As a result, we cannot conclusively state, from a statistical standpoint, that one method consumes more energy than the other. However, our investigation into response time revealed significant differences when employing the software tactic for some workloads. Notably, this was confirmed for two out of the three workloads studied. These outcomes highlight the importance of considering the impact of software tactics on response time and energy consumption in cloud-based applications and suggest further exploration into the interplay between different software tactics and their effects on application performance.

As a future endeavor, this thesis opens up several potential avenues for further exploration. The experiment can be expanded to test the software tactics on different software applications, allowing for a more comprehensive understanding of its impact on energy consumption and response time. Additionally, while our current experiment involved ten trials for each workload, further investigations with more trials for a single workload could provide a larger dataset for analysis and validation. Furthermore, advancements in cloud technologies, particularly Microsoft Azure's commitment to improving transparency in its measurement units, could offer enhanced opportunities for future studies in this domain. Moreover, the Cloud Carbon Footprint tool could also benefit from ongoing developments to improve accuracy and performance, thus providing more reliable data for energy consumption assessment. By addressing these aspects, future research can continue to deepen our understanding of sustainable cloud software tactics and their implications in real-world applications.

REFERENCES

[1] Y. Lee and G. Circella, "Chapter five - ict, millennials' lifestyles and travel choices," in *The Evolving Impacts of ICT on Activities and Travel Behavior*, ser. Advances in Transport Policy and Planning, E. Ben-Elia, Ed., vol. 3, Academic Press, 2019, pp. 107–141. DOI: https://doi.org/10.1016/bs.atpp.2019.08.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2543000919300113.

[2] M. Silverio-Fernández, S. Renukappa, and S. Suresh, "What is a smart device? - a conceptualisation within the paradigm of the internet of things," en, *Visualization in Engineering*, vol. 6, no. 1, p. 3, Dec. 2018, ISSN: 2213-7459. DOI: 10.1186/s40327-018-0063-8. [Online]. Available: https://link.springer.com/10.1186/s40327-018-0063-8 (visited on 04/30/2023).

[3] S. Chetan, G. Kumar, K. Dinesh, K. Mathew, and M. Abhimanyu, "Cloud computing for mobile world," *available at chetan. ueuo. com*, 2010.

[4] J. Stajic, R. Stone, G. Chin, and B. Wible, "Rise of the machines," *Science*, vol. 349, no. 6245, pp. 248–249, 2015. DOI: 10.1126/science.349.6245.248. eprint: https://www.science.org/doi/pdf/10.1126/science.349.6245.248. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.349.6245.248.

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

[6] A. Sunyaev, "Cloud Computing," en, in *Internet Computing*, Cham: Springer International Publishing, 2020, pp. 195–236, ISBN: 9783030349561 9783030349578. DOI: 10.1007/978-3-030-34957-8_7. [Online]. Available: http://link.springer.com/10.1007/978-3-030-34957-8_7 (visited on 04/30/2023).

[7] A. Aljabre, "Cloud computing for increased business value," *International Journal of Business and social science*, vol. 3, no. 1, 2012.

[8] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the cloudsim toolkit: Challenges and opportunities," in *2009 international conference on high performance computing and simulation*, IEEE, 2009, pp. 1–11.

[9] H. Saini, A. Upadhyaya, and M. K. Khandelwal, "Benefits of cloud computing for business enterprises: A review," in *Proceedings of International Conference on Advancements in Computing and Management (ICACM)*, 2019.

[10] P. Neto, "Demystifying cloud computing," in *Proceeding of doctoral symposium on informatics engineering*, vol. 24, 2011, pp. 16–21.

[11] Gartner. "Gartner says worldwide iaas public cloud services market grew 41% in 2021." Accessed on: 2023-06-21, Gartner. (2022), [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-06-02-gartner-says-worldwide-iaas-public-cloud-services-market-grew-41-percent-in-2021.

[12] Gartner. "Gartner says cloud will be the centerpiece of new digital experiences." Accessed on: 2023-06-21, Gartner. (2021), [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2021-11-10-gartner-says-cloud-will-be-the-centerpiece-of-new-digital-experiences.

[13] Greenpeace. "Oil in the cloud: How tech companies are helping big oil profit from climate destruction." Accessed on: Date of access, Greenpeace. (2021), [Online]. Available: https://www.greenpeace.org/usa/reports/oil-in-the-cloud/#executive-summary.

[14] A. Hasna, "Dimensions of sustainability," 2007.

[15] M. Xu and R. Buyya, "Managing renewable energy and carbon footprint in multi-cloud computing environments," *Journal of Parallel and Distributed Computing*, vol. 135, pp. 191–202, 2020.

[16] D. Mytton, "Data centre water consumption," *npj Clean Water*, vol. 4, no. 1, p. 11, 2021.

[17] S. I. Zandalinas, F. B. Fritschi, and R. Mittler, "Global warming, climate change, and environmental pollution: Recipe for a multifactorial stress combination disaster," *Trends in Plant Science*, vol. 26, no. 6, pp. 588–599, 2021.

[18] E. Masanet, A. Shehabi, N. Lei, S. Smith, and J. Koomey, "Recalibrating global data center energy-use estimates," *Science*, vol. 367, no. 6481, pp. 984–986, 2020.

[19] J. Gao, H. Wang, and H. Shen, "Smartly handling renewable energy instability in supporting a cloud datacenter," in *2020 IEEE international parallel and distributed processing symposium (IPDPS)*, IEEE, 2020, pp. 769–778.

[20] G. Procaccianti, P. Lago, and G. A. Lewis, "Green architectural tactics for the cloud," in *2014 IEEE/IFIP Conference on Software Architecture*, IEEE, 2014, pp. 41–44.

[21] Gartner. "Gartner says more than half of enterprise it spending will shift to cloud by 2024." Accessed on: 2023-06-21, Gartner. (2022), [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-02-09-gartner-says-more-than-half-of-enterprise-it-spending.

[22] S. Vikas, K. Gurudatt, M. Vishnu, and K. Prashant, "Private vs public cloud," *International Journal of Computer Science and Communication Networks*, vol. 3, no. 2, p. 79, 2013.

[23] C. M. Mohammed, S. R. Zeebaree, *et al.*, "Sufficient comparison among cloud computing services: Iaas, paas, and saas: A review," *International Journal of Science and Business*, vol. 5, no. 2, pp. 17–30, 2021.

[24] S. Bhardwaj, L. Jain, and S. Jain, "Cloud computing: A study of infrastructure as a service (iaas)," *International Journal of engineering and information Technology*, vol. 2, no. 1, pp. 60–63, 2010.

[25] A. Giessmann and K. Stanoevska-Slabeva, "Business models of platform as a service (paas) providers: Current state and future directions," *JITTA: Journal of Information Technology Theory and Application*, vol. 13, no. 4, p. 31, 2012.

[26] R. Sharma and M. Sood, "Cloud saas: Models and transformation," in *Advances in Digital Image Processing and Information Technology: First International Conference on Digital Image Processing and Pattern Recognition, DPPR 2011, Tirunelveli, Tamil Nadu, India, September 23-25, 2011. Proceedings*, Springer, 2011, pp. 305–314.

[27] D. B. Newell, E. Tiesinga, *et al.*, "The international system of units (si)," *NIST Special Publication*, vol. 330, pp. 1–138, 2019.

[28] L. Pérez-Lombard, J. Ortiz, and D. Velázquez, "Revisiting energy efficiency fundamentals," *Energy Efficiency*, vol. 6, pp. 239–254, 2013.

[29] "World bank: Electric power consumption (kwh per capita) - netherlands." Accessed on: Date of access, World Bank. (Year of data retrieval), [Online]. Available: `https://data.worldbank.org/indicator/EG.USE.ELEC.KH.PC?locations=NL`.

[30] O. W. in Data. "Energy production and consumption." Accessed on: Date of access, Our World in Data. (Year of data retrieval), [Online]. Available: `https://ourworldindata.org/energy-production-consumption`.

[31] U. Nations. "United nations sustainable development goals." Accessed on: Date of access, United Nations. (Year of data retrieval), [Online]. Available: `https://www.un.org/sustainabledevelopment/development-agenda/`.

[32] S. Oyedeji, A. Seffah, and B. Penzenstadler, "A catalogue supporting software sustainability design," *Sustainability*, vol. 10, no. 7, p. 2296, 2018.

[33] H. Herzog, B. Eliasson, and O. Kaarstad, "Capturing greenhouse gases," *Scientific American*, vol. 282, no. 2, pp. 72–79, 2000.

[34] J. Houghton, "Global warming," *Reports on progress in physics*, vol. 68, no. 6, p. 1343, 2005.

[35] K. O. Yoro and M. O. Daramola, "Co2 emission sources, greenhouse gases, and the global warming effect," in *Advances in carbon capture*, Elsevier, 2020, pp. 3–28.

[36] T. S. Project. "Lean ict: Towards digital sobriety." Accessed on: 2023-05-05, The Shift Project. (2021), [Online]. Available: `https://theshiftproject.org/en/article/lean-ict-our-new-report/`.

[37] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice: Software Architect Practice_c3*. Addison-Wesley, 2012.

[38] S. Vos, P. Lago, R. Verdecchia, and I. Heitlager, "Architectural tactics to optimize software for energy efficiency in the public cloud," in *2022 International Conference on ICT for Sustainability (ICT4S)*, 2022, pp. 77–87. DOI: 10.1109/ICT4S55073.2022.00019.

[39] S. A. Ali, M. Affan, and M. Alam, "A study of efficient energy management techniques for cloud computing environment," in *2019 9th international conference on cloud computing, data science and engineering (confluence)*, IEEE, 2019, pp. 13–18.

[40] N. B. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? a model and annotation," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1735–1758, 2010.

[41] A. Uchechukwu, K. Li, Y. Shen, *et al.*, "Energy consumption in cloud computing data centers," *International Journal of Cloud Computing and Services Science (IJ-CLOSER)*, vol. 3, no. 3, pp. 31–48, 2014.

[42] J. Subirats and J. Guitart, "Assessing and forecasting energy efficiency on cloud computing platforms," *Future Generation Computer Systems*, vol. 45, pp. 70–94, 2015.

[43] M. H. Kumar and S. K. Peddoju, "Energy efficient task scheduling for parallel workflows in cloud environment," in *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, 2014, pp. 1298–1303. DOI: 10.1109/ICCICCT.2014.6993161.

[44] Y. C. Lee and A. Y. Zomaya, "Energy efficient utilization of resources in cloud computing systems," *The Journal of Supercomputing*, vol. 60, pp. 268–280, 2012.

[45] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer, 2014, vol. 545.

[46] F. Khomh and S. A. Abtahizadeh, "Understanding the impact of cloud patterns on performance and energy consumption," *Journal of Systems and Software*, vol. 141, pp. 151–170, 2018, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2018.03.063. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218300621.

[47] B. Bani, F. Khomh, and Y.-G. Guéhéneuc, "A study of the energy consumption of databases and cloud patterns," in *Service-Oriented Computing: 14th International Conference, ICSOC 2016, Banff, AB, Canada, October 10-13, 2016, Proceedings 14*, Springer, 2016, pp. 606–614.

[48] S. A. Abtahizadeh, F. Khomh, and Y.-G. Guéhéneuc, "How green are cloud patterns?" In *2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, IEEE, 2015, pp. 1–8.

[49] V. R. Basili, G. Caldiera, and H. D. Rombach, “The goal question metric approach,” 1994.

[50] Microsoft. “Azure App Service Overview.” (N/A), [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/overview (visited on 07/17/2023).

[51] Microsoft. “Azure load balancer,” Microsoft. (2021), [Online]. Available: https://learn.microsoft.com/en-us/azure/load-balancer/ (visited on 07/17/2023).

[52] Microsoft. “Microsoft azure virtual machines.” Accessed on July 17, 2023, Microsoft. (N/A), [Online]. Available: https://learn.microsoft.com/en-us/azure/virtual-machines/overview.

[53] Microsoft. “Azure API Management Key Concepts.” (N/A), [Online]. Available: https://learn.microsoft.com/en-us/azure/api-management/api-management-key-concepts (visited on 07/17/2023).

[54] T. D. Cook, D. T. Campbell, and A. Day, *Quasi-experimentation: Design and analysis issues for field settings*. Houghton Mifflin Boston, 1979, vol. 351.

[55] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2012.