THE IMPACT OF DATA COMPRESSION ON ENERGY CONSUMPTION AND PERFORMANCE IN THE CLOUD

Lappeenranta-Lahti University of Technology LUT

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

2023

Mochamad Rizal Hidayat

Examiners: Professor Jari Porras (LUT University)

Professor Henry Muccini (University of L'Aquila)

Professor Patricia Lago (Vrije Universiteit Amsterdam)

SE4GD
Software Engineers for Green Deal

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

Mochamad Rizal Hidayat

The Impact of Data Compression on Energy Consumption and Performance in the Cloud

Master's thesis

2023

65 pages, 24 figures, 24 tables and 0 appendices

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

Keywords: data compression, row compression, page compression, energy consumption, CPU usage, response time

This thesis investigates the data compression techniques available in Azure, specifically examining the impact of row and page compression on the energy consumption and performance of cloud applications. This thesis was conducted as part of a collaborative internship program with a financial institution in Amsterdam. The research encompasses a comprehensive literature examination, feasibility study, and controlled experiment.

The results of the study revealed that the implementation of data compression techniques, particularly row and page compression, did not yield a statistically significant impact on energy consumption or response time, specifically to the application subjected to the experiment. However, it was observed that these techniques had a noticeable effect on CPU usage.

TABLE OF CONTENTS

ABSTRACT **iii**

1 INTRODUCTION **5**

2 LITERATURE REVIEW **8**

2.1 Theoretical Background 8

2.1.1 Cloud Computing 8

2.1.2 Software Sustainability 9

2.1.3 Microsoft Azure 10

2.1.4 Data Compression 10

2.1.5 Cloud Carbon Footprint 13

2.2 Related Work 14

3 RESEARCH METHODOLOGY **17**

3.1 Study Design 18

3.2 Research Goal and Questions 19

3.3 Experiment Planning 21

3.3.1 Subject Selection 21

3.3.2 Experiment Variables 22

3.3.3 Experiment Hypothesis 22

3.3.4 Experiment Design 24

3.3.5 Data Analysis 25

4 EXPERIMENT EXECUTION **28**

4.1 Infrastructure and Configuration 28

4.2 Preliminary Testing 30

4.3 Orchestration 31

4.3.1 Preparation 31

4.3.2 Execution 32

4.4 Data Collection 33

5 RESULT **35**

5.1 Energy Consumption (RQ1) 35

5.1.1 Data Exploration 35

5.1.2 Normality Testing 37

5.1.3 Hypothesis Testing 39

5.2 CPU Usage (RQ2) 40

5.2.1 Data Exploration . . . 40
5.2.2 Normality Testing . . . 42
5.2.3 Hypothesis Testing . . . 43
5.2.4 Effect Size Estimation . . . 45
5.3 Response Time (RQ2) . . . 46
5.3.1 Data Exploration . . . 46
5.3.2 Normality Testing . . . 48
5.3.3 Hypothesis Testing . . . 49

6 DISCUSSION **51**
6.1 Energy Consumption (RQ1) . . . 51
6.2 Performance (RQ2) . . . 52

7 THREATS TO VALIDITY **55**
7.1 Internal Validity . . . 55
7.2 External Validity . . . 55
7.3 Construct Validity . . . 56
7.4 Conclusion Validity . . . 56

8 CONCLUSIONS **58**

REFERENCES **59**

LIST OF FIGURES

1 Public Cloud Worldwide . . . 5
2 Share of Corporate Data Stored in the Cloud . . . 6

3 Illustration of Prefix Compression . . . 12
4 Illustration of Dictionary Compression . . . 13

5 Overview of the study design . . . 17
6 GQM (Goal-Question-Metric) tree . . . 21
7 Overview of the data analysis . . . 27

8 Application Deployment Diagram . . . 29
9 Row Compression Query . . . 30
10 Page Compression Query . . . 30
11 Overview of Experiment Execution . . . 32
12 Overview of Data Collection . . . 34

13 Boxplot and Violin plot of Energy Consumption with 100 users 36
14 Boxplot and Violin plot of Energy Consumption with 200 users 36
15 Distribution Density Plots of Energy Consumption for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users 37
16 QQ Plots of Energy Consumption for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users 38
17 Boxplot and Violin plot of CPU Usage with 100 users 41
18 Boxplot and Violin plot of CPU Usage with 200 users 41
19 Distribution Density Plots of CPU Usage for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users. 42
20 QQ Plots of CPU Usage for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users 43
21 Boxplot and Violin plot of Response Time with 100 users 47
22 Boxplot and Violin plot of Response Time with 200 users 47
23 Distribution Density Plots of Response Time for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users 48
24 QQ Plots of Response Time for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users 49

LIST OF TABLES

1 Goal definition 19
2 Overview of experiment trials 24

3 App Service Specification 28
4 SQL Database Specification 28
5 Table Specification 29
6 Random Criteria 32

7 Summary Statistic of Energy Consumption with 0 users 35
8 Summary Statistic of Energy Consumption with 100 and 200 users 37
9 Shapiro-Wilk test for Energy Consumption 38
10 Levene's test for Energy Consumption 39
11 ANOVA test for Energy Consumption with 100 users 39
12 ANOVA test for Energy Consumption with 200 users 40
13 Summary Statistic of CPU Usage with 100 and 200 users 40
14 Shapiro-Wilk test for CPU Usage 42
15 Levene's test for CPU Usage 44
16 ANOVA Test for CPU Usage with 100 users 44

17 Tukey HSD Post Hoc Comparisons for CPU Usage with 100 users 44
18 ANOVA Test for CPU Usage with 200 users 45
19 Tukey HSD Post Hoc Comparisons for CPU Usage with 200 users 45
20 Effect Size Estimation: Cohen's d for CPU Usage with 100 users 45
21 Effect Size Estimation: Cohen's d for CPU Usage with 200 users 46
22 Summary Statistic of Response Time with 100 and 200 users 46
23 Shapiro-Wilk test for Response Time 48
24 Kruskal-Wallis test for Response Time 50

1 INTRODUCTION

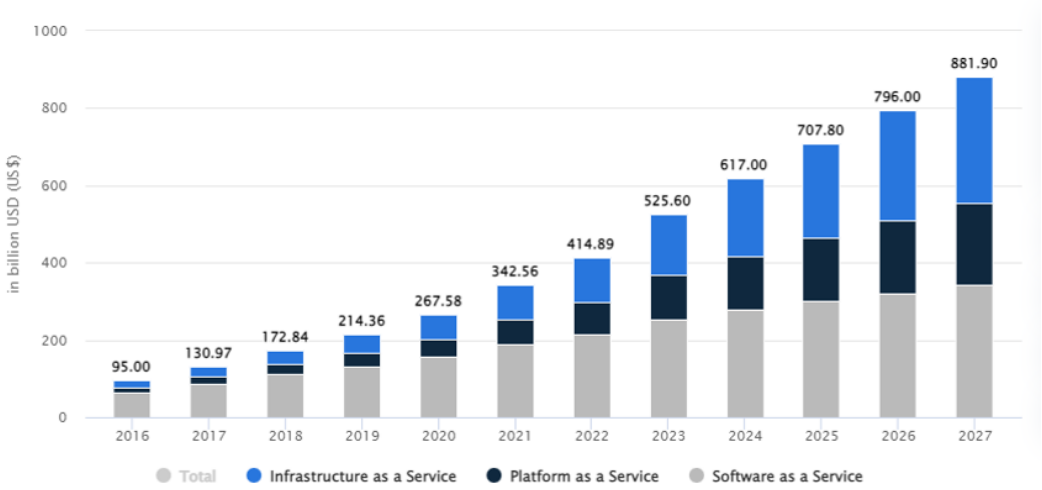

Figure 1: Public Cloud - Worldwide (Source: `https://www.statista.com/outlook/tmo/public-cloud/worldwide`)

In an era marked by remarkable digital transformation, enterprises persistently pursue innovative solutions to maintain competitiveness, enhance operational efficiency, and stimulate growth. In this context, cloud computing has emerged as a transformative technology revolutionizing business operations by providing services with the flexibility of being accessible anytime and anywhere [1]. Moreover, cloud computing infrastructures have the potential to enable enterprises to optimize the utilization of their IT hardware and software investments [2]. By consolidating resources into expansive cloud infrastructures, businesses can reduce costs and enhance utilization, providing resources only when they are required, resulting in increased efficiency. [2].

The utilization of cloud computing has demonstrated its potential to provide advantages in comparison to other computing systems in terms of decreased costs, and increased adaptability, and enables enterprises to enhance their operational efficiency. Furthermore, cloud computing is not constrained by limitations on space and storage [3]. As a result, the development of cloud computing has revolutionized the way of processing and organizing data, leading to a transformation in the way businesses manage their operations [3].

There has been a significant increase in the adoption and utilization of cloud computing technology in recent years. It is projected that by 2027, the revenue generated by cloud computing will exceed 850 billion US dollars on a global scale [4], as shown in Figure 1.

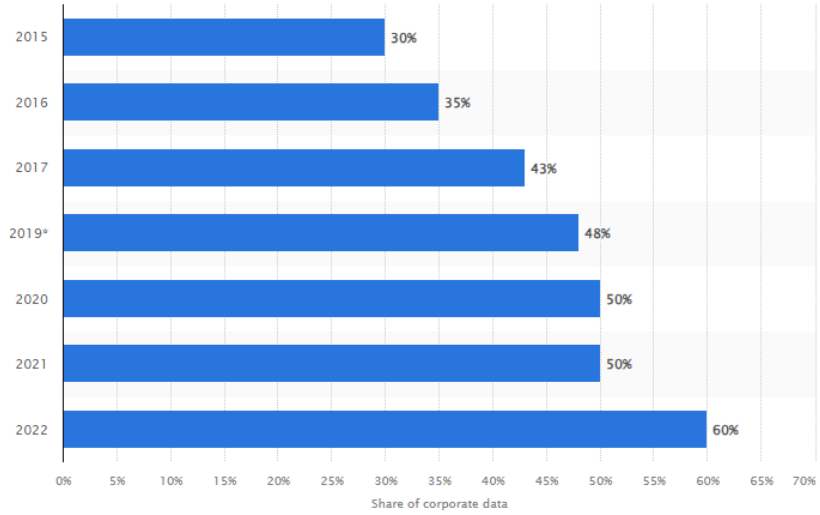

Figure 2: Share of Corporate Data Stored in the Cloud in Organizations Worldwide from 2015 to 2022 (Source: `https://www.statista.com/statistics/1062879/worldwide-cloud-storage-of-corporate-data/`)

By the year 2022, more than 60 percent of all corporate data will have been uploaded and kept in the cloud, as shown in Figure 2. This share reached 30 percent in 2015 and has continued to grow since then as businesses continue to move more of their resources into cloud environments in the expectation of enhancing their security and reliability while also expanding their business agility [5].

However, the transition to cloud computing presents a challenging task. The optimization of energy efficiency is an essential consideration in the design of cloud-based software, which requires appropriate adaptation in the software [6]. In the event of an abrupt migration of the application, there is a risk that the waste of energy would outweigh the expected benefits to a significant extent. As a result, it is essential for software engineers to take into account the effects their design decisions would have on energy consumption [6].

It is recommended for software engineers to incorporate strategies that facilitate efficient data management when redesigning and restructuring applications for deployment on the cloud. This is intended to reduce the number of resources needed, including energy for data storage, computing resources for data processing, and networking resources for data transfer between the cloud, edge, and customer sides [7]. Illustrations of energy-efficient data management strategies may encompass techniques such as optimizing data flow, implementing data deduplication, and utilizing intelligent data compression techniques based on the frequency of data usage [7].

This thesis was conducted in collaboration with a financial institution located in Amsterdam, a prominent financial institution that places a significant emphasis on incorporating sustain-

ability into the institution's strategy. However, due to a confidentiality agreement with the institution, the details about the institution could not be disclosed. The confidentiality agreement necessitates that only a limited amount of information can be disclosed, ensuring the institution's proprietary and sensitive information remains secure and confidential.

The present study dived into the contribution of the IT department of the financial institution towards environmental sustainability, with a particular focus on its efforts to enhance energy efficiency. According to findings from an interview conducted with a specific application team, it was discovered that a certain number of the applications had undergone migration to the cloud platform. Nonetheless, it became evident that the sustainable strategy towards managing data storage in cloud-based databases, such as the utilization of data compression techniques, had not been taken into account. This study aims to investigate the potential impacts of implementing data compression strategies on the environmental sustainability of cloud-based applications.

This research aims to study the impacts of data compression implementation, as a chosen software tactic, on the energy consumption and performance of a cloud-based application. We started by performing literature reviews and documentation studies of VU Sustainability Tactics [1] to investigate the tactics available to enhance energy efficiency in cloud applications. Consequently, a feasibility study was conducted to determine the viable tactics to be implemented in the selected application by looking into the application documentation and Azure documentation which is the cloud platform that the institution uses for the specific application. Based on the findings from the feasibility study, it is determined that a suitable software tactic to be explored and investigated is the implementation of data compression techniques. Afterward, we conducted exploratory interviews to collect specific information from the application that needed to be set up to perform a controlled experiment. Finally, the data collected from the controlled experiment was then used to evaluate the impacts of data compression techniques on the cloud application.

The structure of this thesis is as follows: In Chapter 2, we provide the reader with a literature review encompasses theoretical background and related work section. Chapter 3 presents our research methodology. Chapter 4 presents the details of the experiment execution, of which the results are discussed in Chapter 5. We discuss the obtained results in Chapter 6 and threats of validity in Chapter 7. Finally, we conclude our thesis in Chapter 8.

[1] https://s2group.cs.vu.nl/AwesomeAndDarkTactics/

2 LITERATURE REVIEW

2.1 Theoretical Background

This section presents a comprehensive overview of fundamental concepts that are central to the scope of this thesis. This chapter is structured as follows. Section 2.1.1 defines cloud computing concept. In Section 2.1.2, we discuss software sustainability and software energy consumption. Section 2.1.3 provides information about Microsoft Azure, the cloud platform we focus on in this study, and the service types we use in the experiment: Azure App Service and Azure SQL Database. Section 2.1.4 defines the data compression techniques we evaluate in the experiment, namely Row Compression and $Page$ Compression. In the last Section 2.1.5, we discuss the tool that is used to measure the energy consumption of the cloud application in the experiment, which is the Cloud Carbon Footprint.

2.1.1 Cloud Computing

The definition of cloud computing is established based on the commonly accepted interpretation found in literature, specifically referring to the definition provided by the American National Institute of Standards and Technologies (NIST) [8]:

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

Cloud computing is an innovative information technology model that facilitates the consumption and delivery of services via the Internet. Cloud computing becomes appealing to business owners due to its ability to minimize the need for users to spend time on advance planning for resource allocation. This technology enables enterprises to start business operations with minimal resources and subsequently scale up their provisions solely in response to an increase in service demand [9]. Cloud computing encompasses three different service models, i.e., Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS) [8]. Moreover, cloud computing encompasses four deployment models, i.e., Private Cloud, Community Cloud, Public Cloud, and Hybrid Cloud [9].

2.1.2 Software Sustainability

Dick *et al.* [10] define sustainable software as "*software, whose impacts on economy, society, human beings, and environment that result from development, deployment, and usage of the software are minimal and/or which have a positive effect on sustainable development.*".

The concept of software sustainability encompasses three dimensions :

- Environmental sustainability: This dimension refers to the impact of software product development, maintenance, and usage on energy consumption and other natural resources [11].
- Human sustainability: This dimension evaluates the impact of software creation and utilization on the sociological and psychological dimensions of the software development community and its members [11].
- Economic sustainability: This dimension explores the financial balance between the costs and benefits linked to the development, deployment, maintenance, and usage of software [11].

This study primarily centers its attention on the concept of environmental sustainability. Specifically, it explores the intersection of software systems and energy consumption. While software systems do not directly consume energy, they have an impact on the utilization of hardware, resulting in indirect energy consumption [12]. Addressing the energy consumption of software, it is essential to consider the hardware platform, the computational context, and the duration of the process [12].

The objective of this study is to explore strategies for decreasing the energy consumption associated with cloud applications. Energy consumption in the context of cloud computing is most directly associated with the operation of physical hardware in data centers. These hardware devices can range from processing units like CPUs and GPUs, storage devices like hard drives and solid-state drives (SSDs), network devices, to the cooling systems that prevent these devices from overheating.

Energy consumption, denoted as E, can be defined as the product of power dissipation, represented by P, and the duration of time, denoted as t. In mathematical terms, this relationship can be expressed as [12]:

$$E = P \times t$$

2.1.3 Microsoft Azure

Azure is a cloud computing service provided by Microsoft. The initial announcement of this development occurred during Microsoft's Professional Developers Conference in 2008. The cloud platform was introduced to the public in February 2010 under the name Windows Azure. Subsequently, in 2014, it underwent a rebranding and was renamed Microsoft Azure [13]. The platform offers a variety of functionalities, which include software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). Microsoft Azure facilitates a wide variety of programming languages, tools, and frameworks [13]. In this section, we discuss the Microsoft Azure services utilized in our experiment: Azure App Service and Azure SQL Database. Azure App Service and Azure SQL Database are both examples of PaaS services [14].

Azure App Service. The Azure App Service is a web hosting service that utilizes HTTP to facilitate hosting web applications, mobile backends, and REST APIs. It can be developed using various programming languages such as .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. It supports Windows and Linux-based environments [14], [15].

The App Service incorporates features including security, load balancing, autoscaling, and automated management. Furthermore, it also provides the DevOps functionalities of the platform, including continuous deployment from various sources such as Azure DevOps, GitHub, and Docker Hub, package management, staging environments, custom domain, and TLS/SSL certificates [16].

Azure SQL Database. Azure SQL Database is a platform as a service (PaaS) database engine that is fully managed and capable of handling a majority of the database management functions, including to upgrades, patching, backups, and monitoring, without requiring user intervention. The Azure SQL Database operates on the latest stable edition of the SQL Server database. It provides advanced query processing features, such as high-performance in-memory technologies and intelligent query processing [17], [18].

Azure SQL Database enables the establishment of a data storage layer with high availability and performance for Azure-based applications. The SQL Database is a suitable option for contemporary cloud-based applications due to its capability to handle relational data as well as non-relational structures, including graphs, JSON, spatial, and XML [18].

2.1.4 Data Compression

Compression refers to the technique of condensing information into a more concise format compared to its original form [19]. Compression can be categorized into two main types: lossless compression and lossy compression. Lossless compression techniques can recon-

struct the original data from a compressed file without data loss. Lossy compression techniques are capable of reconstructing an approximate version of the original message with some information loss [20]. The implementation of compression within a database management system (DBMS) consistently conforms to the principles of lossless, as the loss of data is typically deemed undesirable by users. In contrast, lossy compression is typically implemented at the application level [21].

The compression implementation at the database level has the potential to decrease the number of pages used for data storage. This leads to a decrease in the number of pages that queries have to read from the disk, consequently enhancing the overall performance of the system. However, the database server requires extra CPU resources to facilitate the compression and decompression of data while it is being transmitted with the application [22].

This section provides an overview of the data compression techniques available in Azure SQL Database: Row Compression and $Page$ Compression. These compression techniques can be utilized to compress data at the database level.

Row Compression

Row compression alters the physical storage format of the data linked to a particular data type without affecting its syntax or semantics. No modifications to the application are necessary in the event that compression is enabled [23]. There are a few things that occur on a row when row compression is enabled [24]. These are as follows: (1) Modification of the metadata for the row, (2) Fixed-length character data is stored in a variable-length format, (3) Numeric-based data types are stored in variable-length format.

The modification of the metadata result in a general reduction in the amount of information stored for each column compared to a record without row compression. In order to cut down on space, unnecessary bits in the row overhead will be deleted, and the content will be simplified [24]. For fixed-length characters, any white space at the end of the values in the column will be eliminated. For binary data, trailing zeros on the value are stripped off, much like how white space is eliminated. This information is not lost, information on the characters removed from a column is stored in the row overhead [24]. When row compression is used, the data types that are manipulated the most are numeric-based data types. These data types are stored in the smallest form possible to save space. This means that a column with the *bigint* data type, which ordinarily requires 8 bytes, would require only 1 byte if the value stored is between 0 and 255. After reaching the value 256, the column would then save it in 2 separate bytes. This applies to all numeric data types. [24].

Page Compression

The initial step in implementing page compression involves the implementation of row compression. Page compression consists of two following compression steps : Prefix compression and Dictionary compression [25].

In the event of creating a new table with page compression, compression does not occur. Nevertheless, the metadata associated with the table specifies the utilization of page compression. As new data is appended to the first data page, the data is row-compressed. Page compression does not provide any advantage when the page is not completely filled. Upon reaching the full capacity of the page, the subsequent row to be inserted triggers the start of the page compression process. The entire page undergoes a comprehensive review process, wherein each column is assessed for its prefix compression, followed by an evaluation of all columns for dictionary compression [26].

On the conversion of an already existing table that comprises data to page compression, each page undergoes a rebuilding process and is evaluated. The process of rebuilding all pages results in the rebuilding of the corresponding table, index, or partition [26].

Prefix Compression

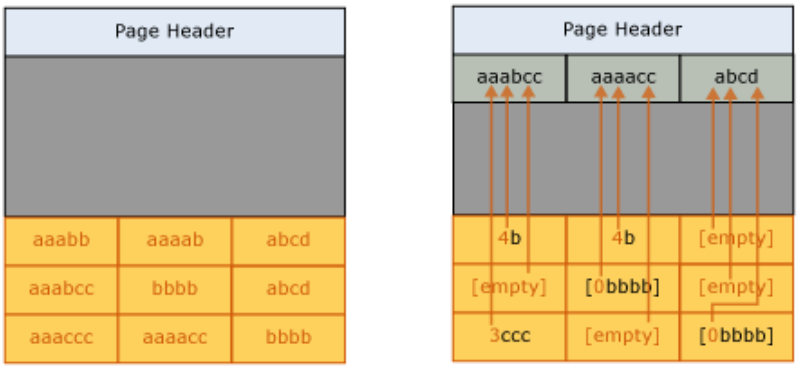

Figure 3: Illustration of Prefix Compression (Source: `https://learn.microsoft.com/en-us/sql/relational-databases/data-compression/page-compression-implementation`)

The process of prefix compression involves determining a common prefix for a specific column among multiple rows within a given page. After determining the optimal prefix value, SQL Server selects the longest value that encompasses the entire prefix as the anchor row. It then stores all other values within the column as a differential of the anchor row, rather than storing the values directly [25].

The prefix compression technique involves a series of steps [26] : (1) A distinct value is determined for every column, which can be used to reduce the storage capacity for the values in each column, (2) The compression information (CI) structure, which is located directly

after the page header, contains a row that refers to the prefix values for each column, (3) The column's recurring prefix values are substituted with a reference to the corresponding prefix. In cases where the value in a row does not precisely correspond to the chosen prefix value, it is possible to refer to a partial match.

Dictionary Compression

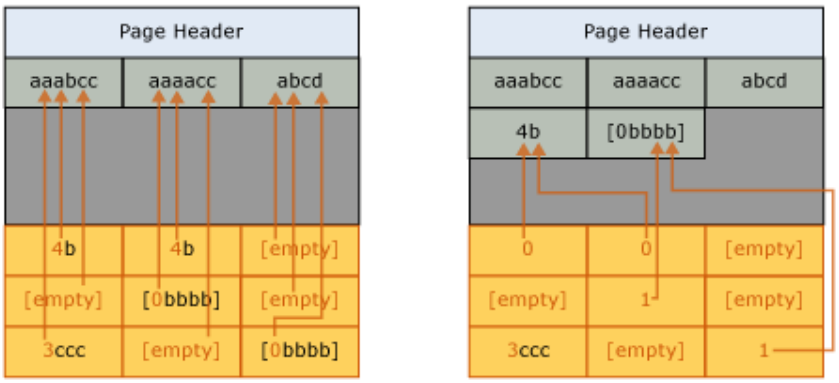

Figure 4: Illustration of Dictionary Compression (Source: `https://learn.microsoft.com/en-us/sql/relational-databases/data-compression/page-compression-implementation`)

Following the completion of prefix compression, the next step involves the implementation of dictionary compression. The process of dictionary compression involves the identification of repeating values present on a given page, which are afterwards stored in the CI area [26]. The matching process is conducted by utilizing the binary representation of a value, thereby ensuring that the process is independent of the data type. When encountering duplicate values, the program appends them to a dedicated dictionary located at the beginning of the page. In the corresponding row, it merely stores a reference to the location of the value within the dictionary [25]. In contrast to prefix compression, dictionary compression is not limited to a single column. The technique of dictionary compression has the ability to substitute recurring values that are present at any location on a page [26].

2.1.5 Cloud Carbon Footprint

The Cloud Carbon Footprint[1] is an open source solution that facilitates the measurement, monitoring, and reduction of cloud-based carbon emissions, thereby enhancing visibility and tooling. The tool employs specific methodologies to transform cloud utilization data into approximations of energy consumption and carbon emissions [27].

[1] https://www.cloudcarbonfootprint.org/

The tool is designed to function across various cloud service providers, such as AWS, Google Cloud, and Microsoft Azure, and presents a comprehensive view of cloud carbon metrics. Cloud Carbon Footprint works by taking cloud provider usage data, converting it into energy, and then taking into account the power usage effectiveness of the cloud provider's data centers and the carbon intensity of the region where the data center pulls power from [27].

The Cloud Carbon Footprint methodology involves the conversion of cloud provider usage data into energy units, followed by the consideration of the power usage effectiveness of the cloud provider's data centers and the carbon intensity of the region from which the data center sources its power [27]. The Cloud Carbon Footprint retrieves utilization metrics related to computing, storage, networking, and other relevant factors from prominent cloud service providers. Subsequently, it computes the estimated energy consumption in units of Watt-Hours and evaluates the corresponding greenhouse gas emissions in terms of metric tons of carbon dioxide equivalents (CO2e) [28]. The estimation of energy consumption employs the Cloud Jewels[2] methodology, originally developed by Etsy, to calculate energy coefficients (kWh) related to computing and storage use in the cloud. Furthermore, this estimation process also includes considerations for energy associated with networking and memory usage [28].

2.2 Related Work

This chapter provides a comprehensive discussion of prior studies aligned with this thesis. Our objective in this thesis is to investigate the impact of data compression implementation on energy consumption and performance in the cloud. In order to achieve this objective, we have captured previous studies related to this topic.

A comparative study by Shanmugasundaram *et al.* [29] compared the compression efficiency in various lossless data compression algorithms. It specifically emphasizes their application to text data. This study compares the compression efficiency between Statistical compression techniques and Dictionary-based compression techniques. The researchers discovered that Arithmetic coding performed better than other statistical compression techniques. In the dictionary-based category, the LZFG algorithm exhibited the highest level of performance.

Another comparative study by Al-Laham *et al.* [30] presents a comparative analysis of different algorithms used in data compression techniques, particularly LZW and Huffman compression. The primary objective of this study is to analyze and evaluate the efficiency and effectiveness of these techniques in terms of their compression ratios.

The compression of information within a database system is an attractive option due to two primary advantages: reduction in storage requirements and enhancement of system perfor-

[2]https://www.etsy.com/codeascraft/cloud-jewels-estimating-kwh-in-the-cloud/

mance. The advantage of storage-saving is a direct impact, while the performance enhancement results from the reduced space needed to relocate data during database operations physically [31].

A study conducted by Cormack *et al.* [31] investigates the impact of data compression techniques on IBM's Information Management System (IMS) database. The study uses a methodology that utilizes a data compression technique created on modified Huffman codes. The method considers the influence of the field's data type on the probability of a character's occurrence. This study's findings demonstrate that data compression techniques can lead to notable reductions in disk storage and activity. Although the compression implementation may increase the CPU time required for loading the database, the overall time required for this process is reduced.

A study performed by Ray *et al.* [32] provides a comprehensive review of the utilization of data compression techniques in databases. This study examines a range of compression algorithms: Run Length Encoding (RLE), Lempel-Ziv-Welch (LZW), Huffman, and Arithmetic coding. These algorithms are evaluated in terms of their application at different levels of granularity, specifically file, page, record, and attribute levels. The researchers conducted experiments on three distinct categories of databases, namely, a bibliographic database (referred to as PUB), a bank database (referred to as BANK), and a synthetic numeric database (referred to as NUM). The authors state that the utilization of database compression benefits in terms of saving space and improving the efficiency of query processing.

A study performed by Salgova *et al.* [33] examines the impact of table compression on the reduction of disk and buffer cache requirements for database tables. When a table is declared as compressed, it will utilize a reduced number of data blocks on the disk, decreasing the overall disk space needed. The information contained in a compressed table is retrieved and stored in its compressed state, and it is only decompressed when it is accessed for retrieving the data. Salgova *et al.* [33] conducted the experiment wherein nine tables were utilized, each exhibiting different levels of compression and index compression. The findings indicated minimal variation in data access times across the tables. The costs associated with the central processing unit (CPU) exhibited a logical increase as the number of rows increased. However, the compression level did not decrease to its fullest potential in the case of a table with low cardinality. The researchers concluded that the table's attributes possessed distinct values, resulting in a high cardinality. Consequently, the impact of compression was deemed insignificant [33].

A study by Govinda *et al.* [34] evaluates the implementation of the LZW algorithm. It shows that the use of LZW to compress data during storage in a cloud environment could effectively reduce data transmission time and storage requirements. This compression and

decompression process relies on the speed of the compiler. From the experimentation, it can be inferred that LZW does not bear the overhead of key transmission since the decompression is pre-set with ASCII values. Thus, the author claims that the LZW can be successfully integrated with the cloud.

A study by Nicolae [35] evaluates the benefits of implementing transparent data compression at the level of the database service, unlike the previous studies that primarily focused on optimizing end-to-end data transfers, the work aimed to achieve a high total aggregated throughput, which was the relevant performance I/O metric for large-scale deployments.

A study by Nilsson *et al.* [36] explores the techniques to optimize databases in the cloud based on performance and cost savings. The study focuses the optimization with compression and indexing. Compression and indexing were selected because they are expected to have the most significant impact when managing databases with variable bottle-necked utilization metrics. The compression techniques evaluated are row compression and page compression. The result for both compressions shows a positive result in faster server processing time and enhanced performance in general.

Based on the insights from prior studies, data compression offers numerous benefits. However, it exposes certain gaps that need to be addressed. Firstly, most studies have concentrated primarily on the correlation between data compression and application performance. It leads to the limited studies of data compression techniques on energy consumption remaining unexplored. Secondly, it is important to acknowledge that only a limited number of prior studies have specifically conducted experiments in a cloud environment. Conducting such experiments is essential as it enables software engineers to assess the real-world implications of data compression in the cloud. Lastly, previous research has predominantly focused on compression techniques at the application level, which may not be as efficient as applying compression techniques at the database level.

3 RESEARCH METHODOLOGY

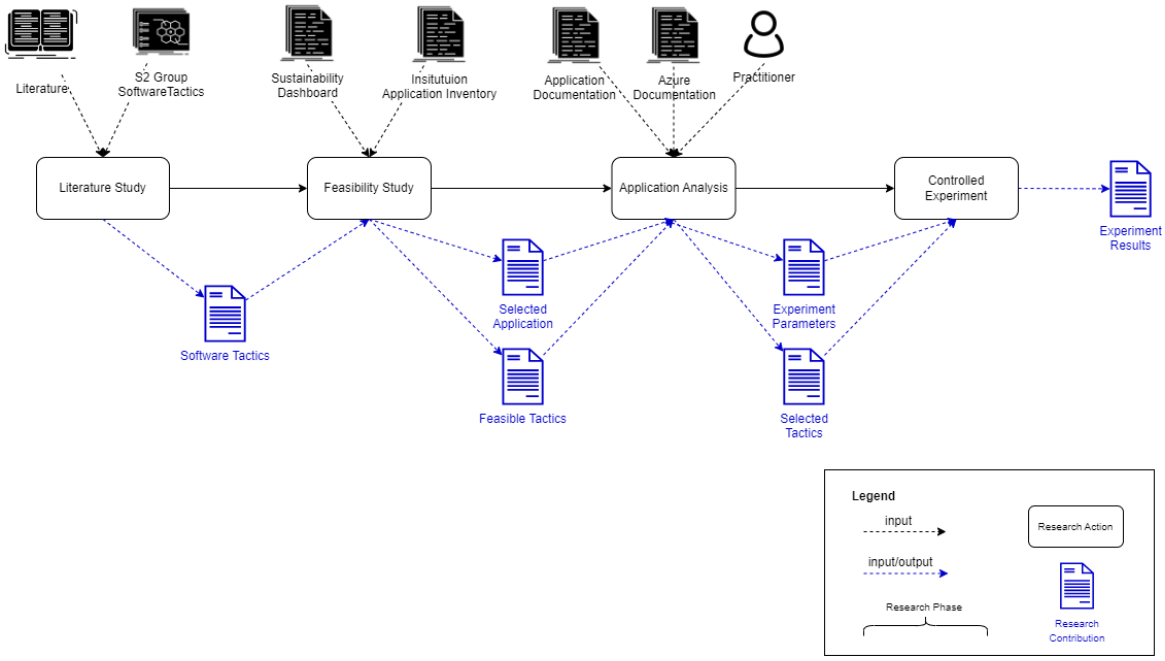

Figure 5: Overview of the study design

This thesis was conducted as part of a collaborative internship program with the ABC, a financial institution based in Amsterdam that prioritizes sustainability in its strategic planning. The ABC offers a diverse range of financial services encompassing retail banking, corporate banking, private banking, asset management, and investment banking. The ABC places considerable emphasis on digital banking services in response to the development of the digital era. In order to facilitate this process of digital transformation, the bank has made substantial investments in its information technology infrastructure and has actively adopted the services offered by cloud computing. The implementation of cloud technologies has facilitated the improvement of service delivery within the bank, enabling the provision of more efficient and reliable banking solutions to its customers.

The ABC utilizes data centers to support its extensive operations. The data centers, which are essential components of the bank's IT infrastructure, contain essential hardware such as servers, storage systems, and network devices. These facilities are either owned by the bank itself or leased from external data center providers. Furthermore, in accordance with the current trend towards cloud computing, ABC has taken advantage of virtual data centers offered by cloud service providers. This approach improves the scalability and flexibility of the bank's digital services.

The goal of this study was to investigate the impact of applying software tactics from the S2 Group[1] of the Vrije Universiteit (VU) Amsterdam within the operational context of the financial institution ABC.

3.1 Study Design

Figure 5 visualizes an overview of the study design for this research collaboration. The process consists of four steps: literature study, feasibility study, application analysis, and controlled experiment.

Literature Study. The primary objective was to get a deeper comprehension of the topics that are the subject of this study. To comprehensively understand the sustainable cloud research topic, we examined the literature on cloud computing, cloud application architecture, and energy optimization in cloud environments. To examine the software tactics strategy, software tactics documentation from S2 Group of the Vrije Universiteit (VU) Amsterdam was studied to analyze practices and approaches to enhance energy efficiency in cloud applications.

Feasibility Study. A feasibility study was conducted to select the application for the experiment and the feasible tactics to be implemented. It involved analyzing the sustainability dashboard of the financial institution ABC and details from the institution application inventory. The sustainability dashboard provides essential information on each application's energy consumption and carbon emission, allowing us to identify prospective applications with high energy consumption and carbon emission for which software tactics can be applied. Application inventory complements the feasibility study by providing additional information related to the application details, which is required to determine the viability of implementing particular software tactics. Considering also the developer team that available to support, it is identified that application X is the selected application to be used as a subject in our experiment.

Application Analysis. Following the completion of the feasibility study, application analysis was performed. During the application analysis, various sources were utilized to gather detailed information and the current state of the application to select the tactic compatible with the application's architecture and behavior. The inputs for the application analysis were documentation of application X, Azure documentation as the cloud platform utilized by the application, and detailed information from the developer team during the exploratory interviews. From the application analysis, we identified that the application has already implemented some software tactics but has not yet considered data compression. We refer to tactic from Vos *et. al* about tactic to compress infrequently accessed data [37] that implies less

[1] https://s2group.cs.vu.nl/

energy used when less data is stored for storage. However, during the exploratory interview, it is discovered that the nature of the application does not have infrequently access data. Moreover, the manual compression using customized algorithms in the code seemed not practically to be implemented as there is a need to change in application code, and not scalable to be implemented in many applications. For those reasons, we explored the data compression techniques available in the Azure environment.

Controlled Experiment. Due to some reasons from the ABC, the experiment could not be conducted in the ABC environment. Another concern is that the source code of the application could not be shared due to confidential and security reasons. As a result, the application for the experiment will follow the information provided by the documentation and information gathered from the developer. An experiment was performed in a personal Azure account and was created to replicate the intended cloud environment experiment from financial institution ABC. The application was deployed in Azure, mimicking the architecture and behavior of the particular application in the financial institution ABC. We performed energy consumption measurement calculations in the experiment using open-source technology, Cloud Carbon Footprint[2]. For the simulation test, Locust[3] was utilized to conduct a load test aimed at simulating and measuring the system's performance under different levels of user workload.

3.2 Research Goal and Questions

In this section, we define the goal, research questions, and the related metrics. Finally, the overall of the goal, question and metrics is depicted in a GQM-tree.

Goal. We define the methodology in this section using the GQM (Goal-Question-Metric) methodology introduced by Basili *et al.* [38]. In a formal manner, the overall objective of our study is introduced as :

Analyze	data compression techniques
For the purpose of	evaluation
With respect to	energy consumption and performance
From the point of view of	software developers
In the context of	cloud applications

Table 1: Goal definition

Questions. In order to reach our objective, we pinpoint two critical aspects that could simultaneously influence by data compression techniques implementation. First, it is essential to take into account the energy consumption of software products related to sustainable development [11] as previously discussed in Section 2.1.2. Second, in terms of performance, one

[2]https://www.cloudcarbonfootprint.org/
[3]https://locust.io/

of main benefits often highlighted with the implementation of data compression techniques is the potential to enhance the application performance [13]. We investigate these by using the following research questions:

- **RQ1**: What is the impact of data compression techniques on the energy consumption of the cloud application?

  In order to address this question, we perform an evaluation on the energy consumption of the implementation of data compression techniques ($None$ compression, Row compression, $Page$ compression). The main objective is to assess whether the implementation of data compression results in different in energy consumption.

- **RQ2**: What is the impact of data compression techniques on the performance of cloud application?

  For this purpose, we examine the impact of data compression techniques ($None$ compression, Row compression, $Page$ compression) on the performance of a cloud-based application in terms of its CPU usage and response time.

Metrics. To address the research questions discussed earlier, we defined the metrics used for the purpose of comparing the outcome data. The following metrics are presented as follows:

- **M1** : Energy Consumption

  The measurement of energy consumption in the cloud application is performed by the utilization of an open-source tool known as Cloud Carbon Footprint. This tool is integrated with the Azure subscription that is used as the hosting platform for the application. The unit of measurement used for quantifying energy consumption is the Joule (J).

- **M2** : CPU Usage

  The metric refers to the percentage of time the central processing unit (CPU) of the cloud server is actively performing computational tasks. The objective of this experiment is to evaluate the impact of different data compression techniques on the operational efficiency of cloud-based applications, with a specific focus on CPU usage percentage (%).

- **M3** : Response Time

  This metric in the context of cloud applications refers to the time required for the application to provide a response to request from user. The assessment of response time

for the cloud application is conducted using the open-source tool called Locust, and the measurements are expressed in milliseconds (ms).

GQM Tree. The visual representation of the GQM tree is depicted in Figure 6, showcasing the GQM approach. The present illustration depicts our experimental goal, the associated questions, and the metrics selected to address the questions.

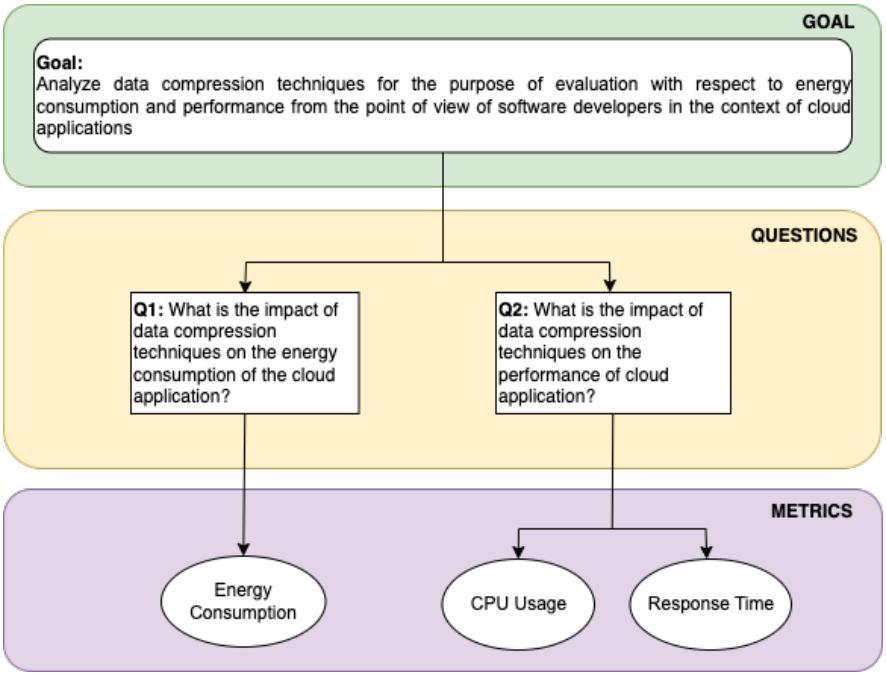

Figure 6: GQM (Goal-Question-Metric) tree

3.3 Experiment Planning

In this section, we elaborate on the details of experiment planning in our controlled experiment. Section 3.3.1 describes the details of subject in the experiment. Section 3.3.2 provides information of independent and dependent variables. Section 3.3.3 presents the hypothesis for the experiment. Section 3.3.4, we elaborate the experiment design. Finally, Section 3.3.5 mentions step by step of data analysis in our experiment.

3.3.1 Subject Selection

The feasibility study revealed that replication by source code of a particular application from financial institution ABC in our controlled environment was unfeasible because of confidentiality and regulatory constraints. Since it is impossible to get the application's source code, we developed the application based on the application architecture, behavior and information details from the developer team. We deployed it into the Azure cloud environment, which is the cloud provider used by financial institution ABC. We utilized the resource services from Azure. This approach aimed to mimic the particular application in financial institution ABC.

The application is a REST API application written in Java language and using Spring Boot[4] framework. The build jar file is deployed into Azure App Service and connected to the database component, which is running on Azure SQL Database.

The component in Azure App Service exposes the API endpoints as URLs that provide a means of communication. These URLs enable the sending of HTTP GET and POST requests. Upon sending HTTP requests to the API endpoint, a JSON-formatted text is returned to the user as a response containing information retrieved from the database.

3.3.2 Experiment Variables

In order to conduct a comprehensive analysis of the causal relationships within our research, we have identified the independent and dependent variables.

For both research questions **RQ1** and **RQ2**, the independent variable is :

- **Compression Type**: There are three treatments for the compression types: $None$ Compression, Row Compression and $Page$ Compression.

To assess the influence of independent variable treatments on energy consumption and performance, we conducted an experiment in which we measured energy consumption as the dependent variable for **RQ1** and CPU usage and response time as the dependent variable for **RQ2**.

- **Energy Consumption**: The energy consumed by the cloud application in Azure Server. The energy consumption is measured in Joule (J).
- **CPU Usage**: The CPU usage is a metric that measures the percentage of CPU resources that are currently being utilized by the database service. The CPU usage is measured in percentage (%).
- **Response Time**: The response time refers to the time required for the application to deliver a response to request from user. The response time is measured in milliseconds (ms).

3.3.3 Experiment Hypothesis

In this section, we discuss the hypotheses of our research.

For **RQ1**, we construct a pair of null and alternative hypothesis for energy consumption:

[4]https://spring.io/projects/spring-boot

- **Energy consumption**: being μ_i the mean energy consumption that is measured during 1 hour test for the treatment with a specific compression type *i*. The null hypothesis and alternative hypothesis are formulated as follows:

$$H_0^{energy} : \mu_{none} = \mu_{row} = \mu_{page}$$

Null Hypothesis : There is no significant difference in the energy consumption among the different compression types ($None$ compression, Row compression, $Page$ compression). In other words, the mean energy consumption for all compression types is equal.

$$H_a^{energy} : (\mu_{none} \neq \mu_{row}) \vee (\mu_{none} \neq \mu_{page}) \vee (\mu_{row} \neq \mu_{page})$$

Alternative Hypothesis : There is a significant difference in the energy consumption in at least one pair of two treatments. In other words, the mean energy consumption for at least one compression type is different from the others.

For **RQ2**, we construct a pair of null and alternative hypothesis for CPU usage and response time :

- **CPU usage** : being μ_i the mean CPU usage that is measured during 1 hour test for the treatment with a specific compression type *i*. The null hypothesis and alternative hypothesis are formulated as follows:

$$H_0^{cpu} : \mu_{none} = \mu_{row} = \mu_{page}$$

Null Hypothesis : There is no significant difference in the CPU usage among the different compression types ($None$ compression, Row compression, $Page$ compression). In other words, the mean CPU usage for all compression types is equal.

$$H_a^{cpu} : (\mu_{none} \neq \mu_{row}) \vee (\mu_{none} \neq \mu_{page}) \vee (\mu_{row} \neq \mu_{page})$$

Alternative Hypothesis : There is a significant difference in the CPU usage in at least one pair of two treatments. In other words, the mean CPU usage for at least one compression type is different from the others.

- **Response time** : being μ_i the median response time that is measured during 1 hour test for the treatment with a specific compression type *i*. The null hypothesis and alternative hypothesis are formulated as follows:

$$H_0^{resp} : \mu_{none} = \mu_{row} = \mu_{page}$$

Null Hypothesis : There is no significant difference in the response time among the different compression types ($None$ compression, Row compression, $Page$ compression). In other words, the median response time for all compression types is equal.

$$H_a^{resp} : (\mu_{none} \neq \mu_{row}) \vee (\mu_{none} \neq \mu_{page}) \vee (\mu_{row} \neq \mu_{page})$$

Alternative Hypothesis : There is a significant difference in the response time in at least one pair of two treatments. In other words, the median response time for at least one compression type is different from the others.

3.3.4 Experiment Design

For this experiment, we use the experimental design *"one factor, multiple treatments"* (1F-MT), where the factor is the compression type, and the treatments are $None$ Compression, Row Compression, and $Page$ Compression.

There is one blocking factor, namely the workload. The workload is a variable that is not of primary interest but is suspected to influence the outcome of the experiment, so, it is considered as a blocking factor. This means that we analyze the potential effect of workload separately. We decide to perform the experiment for several distinct workload levels, specifically 100 users and 200 users. These specific workloads are determined based on the preliminary testing that we conducted and will be detailed in the Section 4.2. We also performed 0 user workload to see the baseline for the energy consumption. Within each workload level, we would test all three types of data compression (none, row, page). This way, we would block on the workload variable, effectively controlling for its impact and isolating the effect of the main factor (compression type). Table 2 provides a complete overview of the distribution of the experiment trials.

Trial	RQ	Workload (user)	Compression Type
1	RQ1	0	None
2	RQ1	0	Row
3	RQ1	0	Page
4	RQ1, RQ2	100	None
5	RQ1, RQ2	100	Row
6	RQ1, RQ2	100	Page
7	RQ1, RQ2	200	None
8	RQ1, RQ2	200	Row
9	RQ1, RQ2	200	Page

Table 2: Overview of experiment trials

The collection of dependant variables, e.g., energy consumption, CPU usage, and response time might have inconsistencies when observed in real-world scenarios. In order to address

this issue, a total of 15 runs are conducted for each trial. Furthermore, to mitigate any potential bias in the sequence of execution, each run is served randomly for the selection of compression types and workloads. Data of the dependent variables for RQ1 and RQ2 is collected from the same trial.

3.3.5 Data Analysis

In order to evaluate the outcomes obtained from the experiment, a variety of statistical tests are used. Figure 7 shows the overview of the data analysis strategy in this experiment.

1. **Data Exploration**: The data collected corresponding to each dependent variable applying different treatments consists of numerical values. Then the data is subjected to quantitative analysis. The analysis begins by conducting preliminary analyses of the measurements using descriptive statistics and a number of plots to assess the data distribution.

2. **Normality Testing**: We visually represent the data through histograms and Q-Q plots for each measured metric to get an initial impression of the data distribution. Histograms give an overview of the distribution density of the data [39]. A roughly bell-shaped histogram is needed to assume the normally distributed data. We also presented other plots, Q-Q plots, which should show a diagonal line *(y=x)* for normality distributed data. To verify the normality precisely, we performed the Shapiro-Wilk test [40]. The test calculates a p-value, which represents the likelihood of obtaining test outcomes that are as extreme as the actual results observed. An α of 0.05 is used as the threshold. If the p-value exceeds the threshold, the data is normally distributed [40]. Based on the results obtained, the following step is to select the method for hypothesis testing, either parametric or non-parametric tests.

3. **Hypothesis Testing**: Before conducting the hypothesis testing, we assess the homogeneity of the variance by using Levene's test. A significance level of 0.05 was used for the test. We decided that a parametric test, one-way ANOVA, or a non-parametric test, Kruskal-Wallis, would be used based on the result of Levene's test.

   Assumptions are not violated. Once it has been defined that the data conforms to a normal distribution and the assumption of homogeneity of variance is not violated, the one-way ANOVA is applied to determine whether a statistically significant distinction exists between two or more categorical groups [39]. ANOVA, which stands for analysis of variance, is a statistical technique that enables us to identify whether there are any differences among the various levels of the independent variable. However, ANOVA does not provide information on which specific differences are significant [39]. In

order to determine the differences between the treatment levels, we utilize a TukeyHSD (Tukey's Honestly-Significant Difference) post-hoc test.

Assumptions are violated. After finding that the data does not follow a normal distribution or the assumption of homogeneity of variances is violated, the Kruskal-Wallis [41] test is used to assess whether there exists a statistically significant difference among at least two treatments. This test is recommended for cases of one factor with more than two treatments [39]. An α of 0.05 is used as the threshold. We reject the null hypothesis if the p-value is less than the threshold. If the p-value is greater, we do not have strong enough evidence to reject the null hypothesis. In order to mitigate the risk of false discoveries, the Bonferroni correction was used to adjust the significance levels of individual p-values obtained from the Kruskal-Wallis test.

4. **Effect Size Estimation Testing**: After finding that the data represents a normal distribution, applying Cohen's d test is suitable. Regarding not normally distributed data, we use the Cliff's Delta test [42] that does not assume a normal distribution. Both of these tests are conducted after the post-hoc doc test to better understand the difference between the treatments.

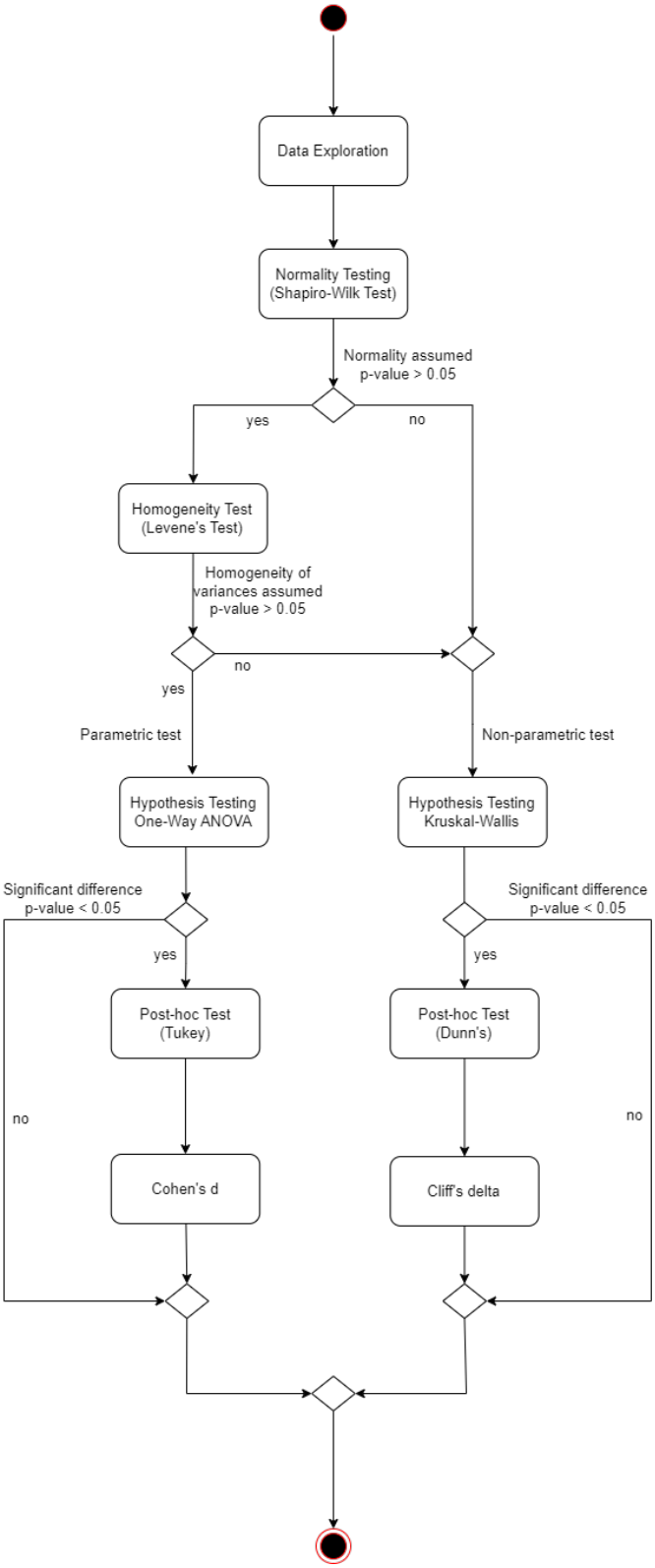

Figure 7: Overview of the data analysis

4 EXPERIMENT EXECUTION

This chapter provides a comprehensive explanation of our experiment execution. We discuss the infrastructure and configuration of our experiment in Section 4.1. We describe the preliminary testing that help us to determine the parameters value in Section 4.2. The orchestration of the experiment in Section 4.3 provides the details for preparing and performong the experiment in detail. Finally, we discuss how we collect the dependent variables from our experiment in Section 4.4.

4.1 Infrastructure and Configuration

Specification	Description
Pricing Plan	Standard S1
OS	Windows
Runtime	Java 17
Java Web Server	Java SE
Region	North Europe

Table 3: App Service Specification

Specification	Description
Pricing Tier	General Purpose
Compute Tier	Standard-series (Gen5)
vCore	2
Max Storage	1 GB
Region	North Europe

Table 4: SQL Database Specification

The primary components of the experiment consist of Web App and Database. As configured in a real-world scenario in the financial institution ABC, we use the services in Azure cloud as Platform as a Service (PaaS) to develop, run, and manage the application. We use Azure App Service for the Web App. For the database, we use Azure SQL Database which is built on Microsoft SQL Server technology. The specification of Azure App Service is provided in Table 3, and the specification of Azure SQL Database is presented in Table 4.

The application was written with Java programming language and built on Spring Boot[1] framework version 3.1.0. The application follows the fundamental of REST API, providing a stateless interface for client-server communication accessible through URLs. The application provides three API endpoints:

1. **Get List IDs**: The endpoint for retrieving a complete list of IDs (HTTP Method = *"GET"*).

2. **Get Details By ID**: The endpoint for retrieving detailed information based on an ID (HTTP Method = *"GET"*)

3. **Post Data**: The endpoint for submitting new data to be stored (HTTP Method = *"POST")*.

[1] https://spring.io/projects/spring-boot

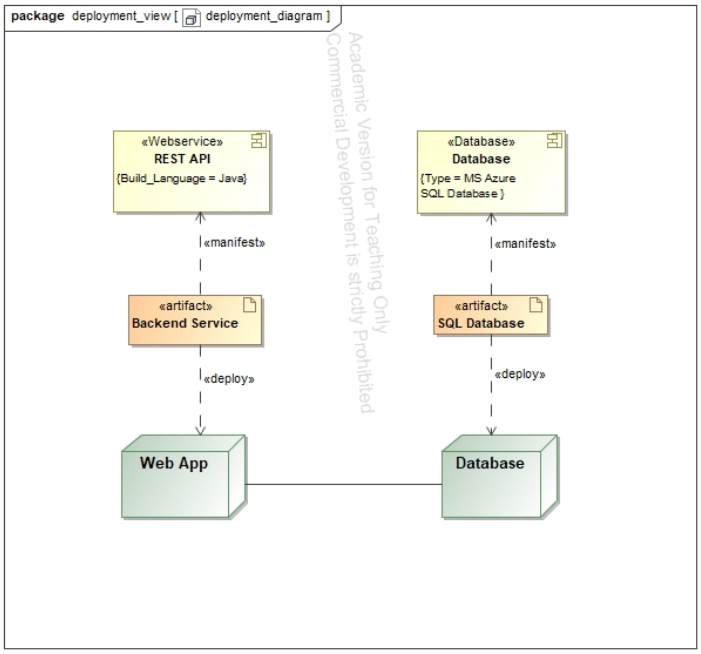

Figure 8: Application Deployment Diagram

The external system communicates with the application through the API URLs by sending HTTP requests and receiving responses. The data exchanged between the application and external systems is typically formatted in JSON[2] (JavaScript Object Notation). The architecture view for the application is depicted in Deployment Diagram in Figure 8.

Field Name	**Data Type**
Field 1	INT
Field 2	VARCHAR(20)
Field 3	VARCHAR(10)
Field 4	VARCHAR(50)
Field 5	VARCHAR(15)
Field 6	DATETIME
Field 7	VARCHAR(20)
Field 8	VARCHAR(255)
Field 9	VARCHAR(255)

Table 5: Table Specification

The application use a single database and a single table. The detailed information of the table including the fields and data types is presented in Table 5. For confidentiality reasons, table name and column names are not disclosed.

The application source code is stored in a GitHub[3] repository and deployed to Azure App Service utilizing GitHub Actions. The connection between Azure App Service and the Azure

[2]https://www.json.org/json-en.html
[3]https://github.com/mochamadrizal26/al-dummyapp-1

```
ALTER TABLE dbo.log
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = ROW)
```

Figure 9: Row Compression Query

```
ALTER TABLE dbo.log
REBUILD PARTITION = ALL
WITH (DATA_COMPRESSION = PAGE)
```

Figure 10: Page Compression Query

SQL Database is established by Java Database Connectivity (JDBC) connection string configured in the application source code. The application employs the Java Persistence API (JPA) Hibernate[4] for database handling. During the experiment, this framework assists in the automatic generation of a table when the application is deployed, so there is no need to create the table manually by scripting in the database server.

After the table is created, the compression type of the table is *"NONE"* compression as a default compression type. The modification of the compression type of the table for the need of the experiment is done by executing an ALTER TABLE script in the Azure query editor, as shown in the Figure 9 for *"ROW"* compression and Figure 10 for *"PAGE"* compression.

4.2 Preliminary Testing

During the initial experimentation phase, various customized workloads were examined in the load test to identify the right workload to represent the real-world scenario. It is discovered that 100 users workload is best suited for the experimentation since it produces 25,000 requests in 1 hour of testing. This workload represents requests received by the actual application in the real-world scenario. Then another workload level, 200 users, is determined to investigate the difference it may influence the dependent variables if we doubled the users' workload.

Regarding the duration of each run for both RQ1 and RQ2, the duration of 1-hour testing is decided based on several rationales. Firstly, in this experiment, we refer to the average requests received by the application in 1 hour in a real-world scenario. Additionally, 1-hour testing is best represented for the energy consumption calculation by Cloud Carbon Footprint, which uses the Terabyte-hour (Tbh) unit representing a Terabyte of storage space used in 1 hour. These preliminary tests played an essential part in identifying workload variations and determining the duration for each iteration.

[4]https://docs.spring.io/spring-framework/reference/data-access/orm/hibernate.html

4.3 Orchestration

We orchestrate and measure the metrics in this experiment using various frameworks and tools. We use the open-source framework Locust[5] to organize the request to the Web App. The Locust allows us to specify user behavior within Python script instead of using a user interface (UI) to make requests to the system. We measure the response time by the report obtained from Locust. For the energy consumption related, we utilize Cloud Carbon Footprint[6] to connect to the Azure environment to retrieve the Azure billing data and then calculate the energy consumption. For CPU usage, we collect the information from Azure monitoring.

4.3.1 Preparation

Before conducting the experiment, we have a few steps needed to be completed to prepare the Azure cloud environment. Here are the steps:

1. **Publish Code on GitHub**. We upload the application source code to the GitHub repository. It allows us to automatically utilize build and deployment processes within Azure App Service to simplify the overall development and deployment workflow.

2. **Set up an Azure SQL Database**. We create an instance of Azure SQL Database for the experiment referring to the specification mentioned in Table 4. Upon the completion of the database instance, we obtain the value of the JDBC connection string. This value is needed to establish a connection between the database and the application in Azure App Service.

3. **Set up Azure App Service**. We create an instance of Azure App Service for deploying the REST API application for the experiment. The specification of the App Service refers to Table 3. We configure the Azure App Service to connect to the GitHub repository of the application source code from the Step 1. Since the code utilizes JPA Hibernate framework, it will automatically create the necessary table from the Java class during deployment.

4. **Configure the Compression Type**. After the table has been created, the compression type can be configured as the experiment scenarios by executing the query 9 to alter the compression type to *"ROW"* compression or 10 to alter the compression type to *"PAGE"* compression in Azure query editor. No action is needed for *"NONE"* compression since the default value of the compression type is s *"NONE"* compression for a newly created table.

[5]https://locust.io/
[6]https://www.cloudcarbonfootprint.org/

4.3.2 Execution

The initial phase of the experiment involves the generation of the dataset. In order to mitigate potential bias, we randomize the data used for our experiment. The dataset is populated according to the random criteria set by the developer, as shown in Table 6. The same randomized dataset is then used to evaluate different compression types ($None$ compression, Row compression, $Page$ compression). This approach ensures that the differences observed across the trials are related to the compression type implemented rather than a result of fluctuating data. In summary, our approach attempts to find a balance between the advantages of randomization and the careful investigation of compression types.

Field Name	**Random Criteria**
Field 1	Randomized number
Field 2	Randomized string from the given list of strings
Field 3	Randomized string from the given list of strings
Field 4	Randomized string of 3 fixed options
Field 5	Fixed string
Field 6	Randomized datetime
Field 7	Randomized string with pattern of a user ID
Field 8	Patterned sentence contain random IBAN formats
Field 9	JSON format string with randomize values

Table 6: Random Criteria

As the cloud environment has been prepared, the experiment can be conducted. Figure 11 graphically represents the overview of the execution with the order of steps carried out.

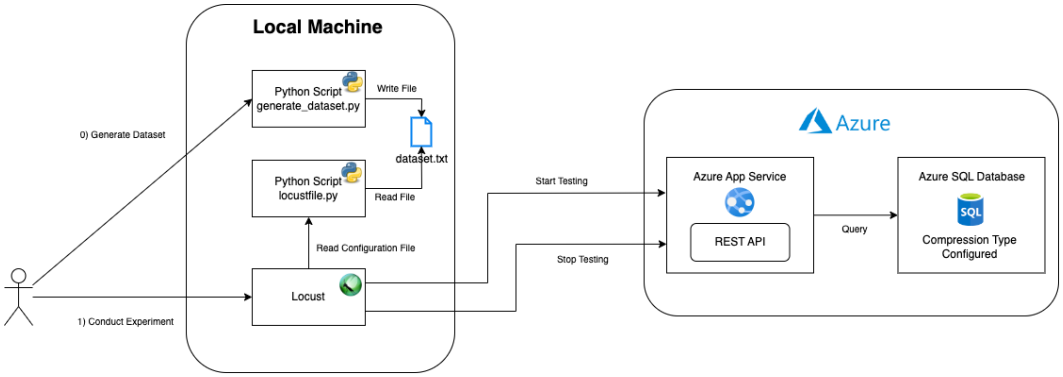

Figure 11: Overview of Experiment Execution

0. **Generate Dataset**. Generating the dataset is done by executing the Python script "$generate_dataset.py$" to populate the data set referring to the random criteria in Table 6. It writes data to a file named ‘‘$dataset.txt$”. This file will be used as a data source for table insertion by *POST* request.

1. **Conduct Experiment**. We initiate the experiment by running the Locust. To run the Locust, we need to fill in a set of parameters: number of users (workload), spawn rate, host (URL), and test duration. The Locust parameters is as follows:

    - **Number of users**. Total number of concurrent users that the system will simulate for the load test. In our experiment, it is varied by 0 user, 100 users, and 200 users as justified in Section 4.2.

    - **Spawn rate**. The frequency at which Locust will generate new users, reaching the maximum number of users as specified. In this experiment, we use 1 as the default value for all trials.

    - **Host** : The URL of the Azure App Service where the application has been successfully deployed.

    - **Test duration**. The duration of the test. In our experiment, it is determined in 1 hour per test run.

    The Locust starts the run by clicking the start button and stops the run automatically by the parameter configured (test duration). The detailed behavior performed by Locust during testing is managed by reading the configuration file *"locustfile.py"*. Locust accesses the REST API deployment in the Azure App Service by sending a HTTP request in JSON format. Azure App Service receives the request and then forward the request by executing query to the Azure SQL Database.

4.4 Data Collection

The data collected for trials 1 through 9 includes energy consumption, CPU usage, and response time measurement. The overview of data collection is depicted in Figure 12.

- **Response time measurement**. Response time is evaluated by extracting data from the Locust report. Locust calculates the response time for every request and the average response time from all the requests once the test is finished. The average response time, collected after each run, is the metric used in our analysis. We collect it in milliseconds (ms) as we get from the Locust report.

- **CPU usage measurement**. To get the information of the CPU usage, we access it through Azure Portal and collect it from the Azure SQL Database service in Azure monitoring.

- **Energy consumption measurement**. Cloud Carbon Footprint (CCF) calculates the energy consumption based on the Azure billing data extracted from Azure Subscrip-

tion. The CCF is set up to connect to the Azure Subscription by integrating it with the Azure Client ID, Client Secret, and Tenant ID. [43], [44]. We integrate the CCF with MongoDB to simplify the extraction of energy consumption data from Azure. It enables the CCF to store the data in our local MongoDB [45], and we can filter the data by the parameters, such as *timestamp*, *serviceName*, *region*, etc.

In this experiment, we calculate the energy consumption for the entire application, which means the total energy consumption from all services involved: Azure App Services, Azure SQL Database, and Bandwidth recorded by CCF. The CCF records the energy consumption in kilowatt-hours (kWh). Nevertheless, due to the difficulty in recognizing a difference in energy consumption in kilowatt-hours (kWh), we convert it to another standard unit of energy, Joule (J). The conversion follows the standard from NIST in the International System of Units (SI). The mathematical expression that represents the relationship between joules (J) and kilowatt-hours (kWh) is as follow [46]:

$$1 \text{ kWh} = 3.6 \times 10^6 \text{ Joule}$$

The Cloud Carbon Footprint (CCF) methodology measures energy consumption for storage by considering allocated bytes rather than utilized bytes [28]. However, in our experiment, we customize the CCF computation to evaluate the difference in energy consumption resulting from the impact of data compression techniques. This modification is implemented by updating the CCF formula from using Terabyte-hours (TBh) based on allocated disk space to using Terabyte-hours (TBh) based on the actual space utilized during the testing.

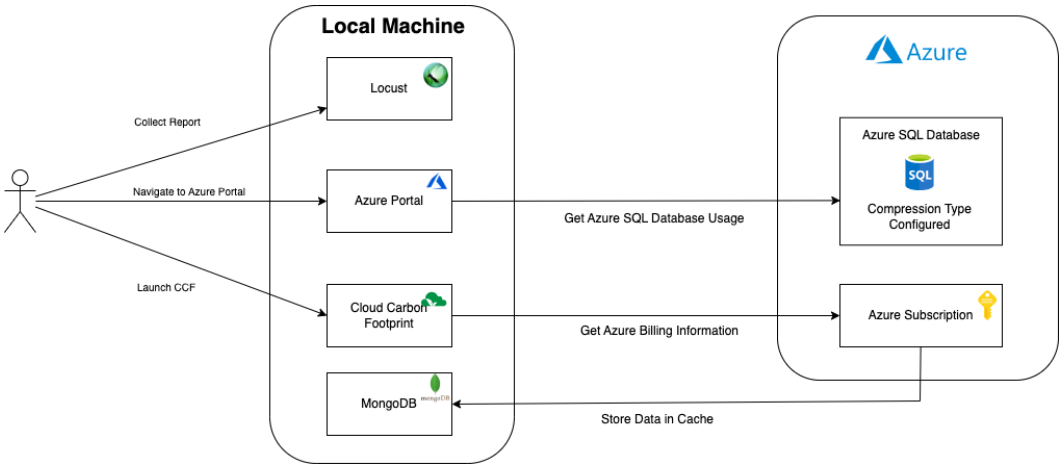

Figure 12: Overview of Data Collection

5 RESULT

This chapter presents the results collected for our dependent variables in our experiment following the data analysis plan in Section 3.3.5. The result of the impact of the energy consumption is presented in Section 5.1. Section 5.2 provides the result for the CPU usage, and Section 5.3 shows the results for the Response Time.

5.1 Energy Consumption (RQ1)

5.1.1 Data Exploration

	0 users		
	Energy Consumption (J)		
	None	Page	Row
Median	229422.146	229502.964	229314.432
Mean	229430.438	229217.339	229611.386
Std. Deviation	691.842	2330.662	683.752
Coefficient of variation	0.003	0.010	0.003
Minimum	227522.003	221776.097	228690.206
Maximum	230387.446	231621.118	231286.449

Table 7: Summary Statistic of Energy Consumption with 0 users

Summary statistic of energy consumption with 0 users is presented in The table 7. This information is essential for establishing the baseline from which we can measure the impact of implementing data compression techniques in the initial state. The mean energy consumption displays roughly the same size across all three compression techniques. This finding implies that, in the workload of 0 users, the impact of the compression technique does not substantially influence the average energy consumption.

This preliminary test is conducted under the assumption of a zero-load condition. The analysis of the influence of compression techniques on energy consumption in the presence of different workloads (100 and 200 users) is presented below.

The box plot and violin plot in Figure 13 and 14 show the distribution of Energy Consumption in Joule of different compression types with 100 and 200 users. A summary statistic of the Response Time for different compression types with different workload levels is presented in Table 8.

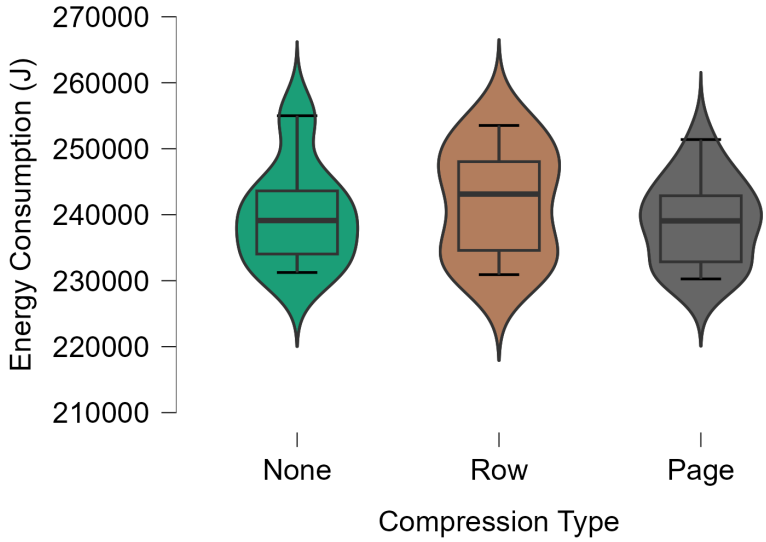

Figure 13: Boxplot and Violin plot of Energy Consumption with 100 users

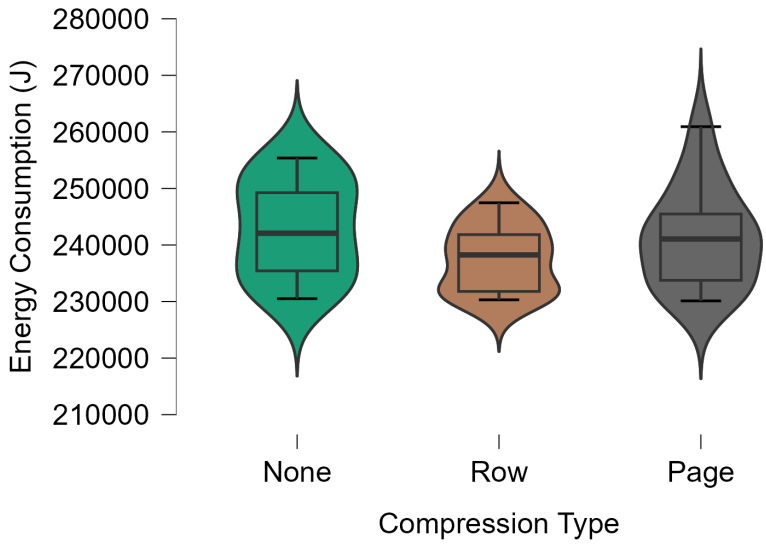

Figure 14: Boxplot and Violin plot of Energy Consumption with 200 users

Based on the data from Table 8, combined with the previous baseline data, it provides insight into how different compression techniques impact energy consumption under varying load conditions. All three compression techniques show a rise in energy consumption as the workload increases. Among the three compression techniques, namely None, Row, and Page, the None compression demonstrates the highest average energy consumption in the workload of 200 users. It implies that energy consumption is lower when the compression technique is implemented.

	100 users			200 users		
	Energy Consumption (J)			Energy Consumption (J)		
	None	Row	Page	None	Row	Page
Median	239121.717	243128.052	239068.598	242082.240	238247.785	241062.898
Mean	239923.036	241778.066	238774.761	242943.611	237427.375	241304.122
Std. Deviation	7624.786	8275.146	6480.103	8711.315	5811.905	8889.042
Coefficient of variation	0.032	0.034	0.027	0.036	0.024	0.037
Minimum	231242.781	230916.667	230265.508	230511.498	230306.094	230114.726
Maximum	255001.551	253530.339	251394.445	255381.944	247463.427	260920.544

Table 8: Summary Statistic of Energy Consumption with 100 and 200 users

5.1.2 Normality Testing

It is essential to know if the data is distributed normally or not before testing the hypothesis. We perform the normality testing in three steps: Density plots, Q-Q plots, and the Shapiro-Wilk test.

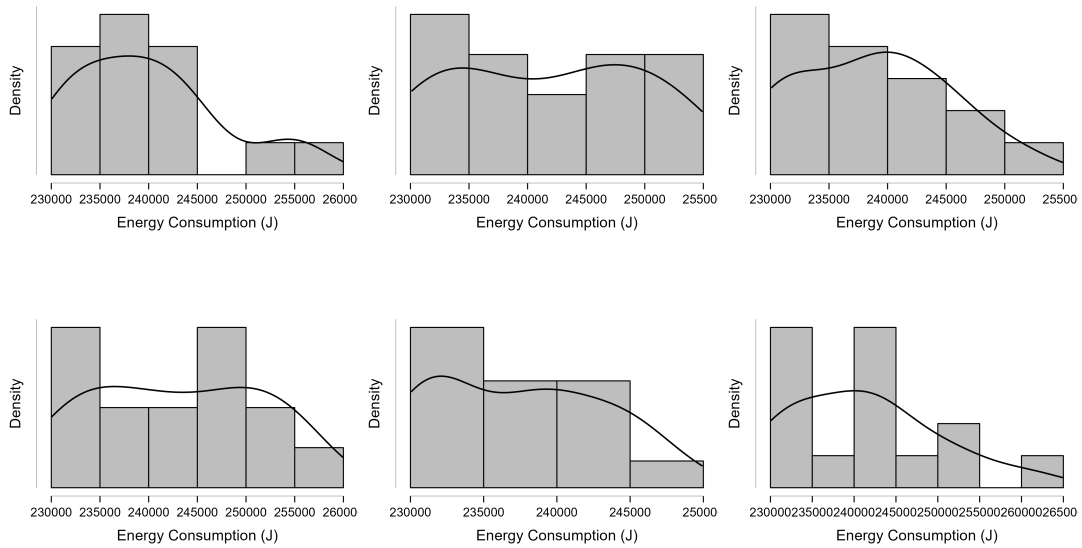

Figure 15: Distribution Density Plots of Energy Consumption for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users

- **Density plots**. In order to evaluate the conformance of the data to a normal distribution, we present density plots as a visual representation. Figure 15 shows the distribution density plots of energy consumption. The three plots on the top represent the energy consumption for a workload of 100 users, and the three plots on the bottom represent for a workload of 200 users. From left to right, $None$, Row, and $Page$. The density plots illustrate a difference from the perfectly symmetrical bell-shaped curve characteristic of a normal distribution, although there is no noticeable presence of significant skewness.

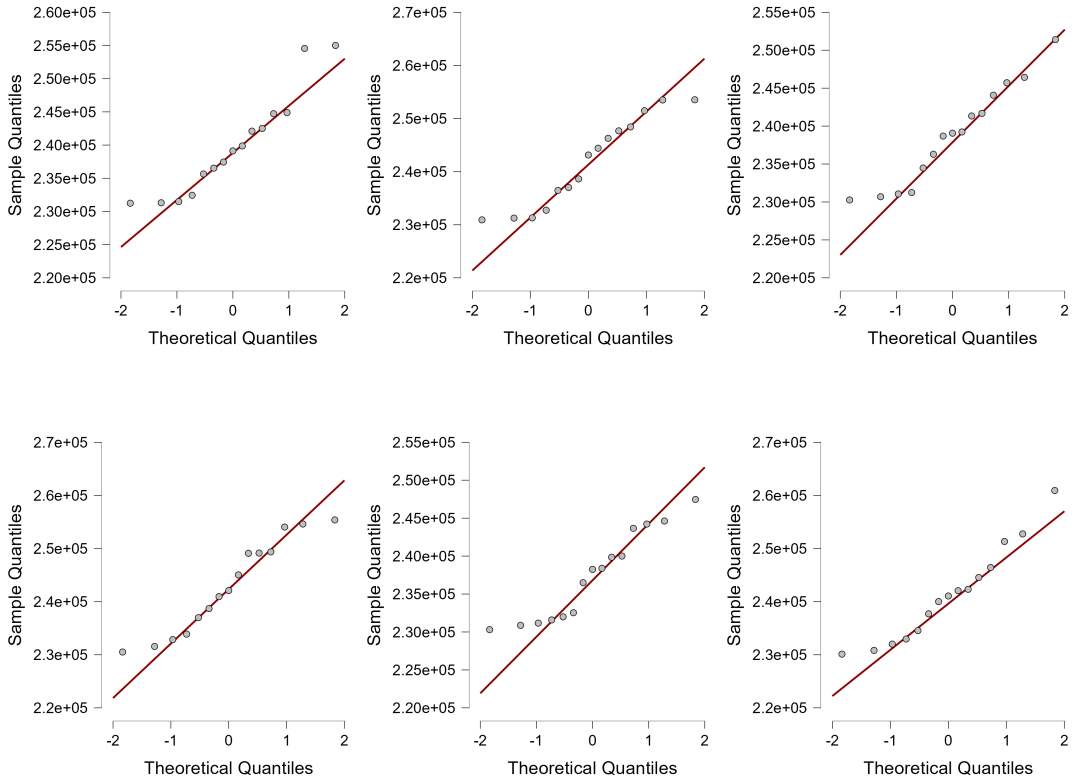

Figure 16: QQ Plots of Energy Consumption for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users

- **Q-Q plots**. To better investigate the normality, we draw the Q-Q plots presented in Figure 16. The plots are placed and ordered the same as the density plots. The Q-Q plots are close to the shape of the 45-degree, indicating that the data follows a normal distribution.

p-value	100 users	200 users
None	0.103	0.236
Row	0.144	0.124
Page	0.471	0.433

Table 9: Shapiro-Wilk test for Energy Consumption

- **Shapiro-Wilk test**. To enhance the accuracy of our evaluation of the normality of the data, we use a quantitative test to check the normality quantitatively by the Shapiro-Wilk test. The data presented in Table 9 show that the p-values calculated by the Shapiro-Wilk test are greater than the significance level of 0.05, implying that there is not enough evidence to reject the null hypothesis. This finding suggests that the data being analyzed follows a normal distribution.

5.1.3 Hypothesis Testing

As described in the experiment design, workloads are considered as a blocking factor in this experiment, we separately evaluate the hypothesis by each workload. Both workloads, 100 users and 200 users, show normally distributed data from the Shapiro-Wilk Test in Table 9.

We follow the data analysis plan that is detailed in Section 3.3.5. To accommodate the hypothesis testing for normally distributed data with $1F - MT$ *"one factor, multiple treatment"*, we use a parametric test one-way ANOVA. Before testing with ANOVA, we check the assumptions needed to be confirmed for performing ANOVA: (i) The dependent variable should be measured on an interval scale: The dependent variable is the energy consumption in Joule. (ii) Normality: The data is normally distributed based on the previous Shapiro-Wilk Test. (iii) Homogeneity of Variance: The variances of the different groups being compared should be equal. To test this homogeneity, we use Levene's Test. The p-values from Levene's Test are presented in Table 10.

Workload	F	df1	df2	p
100 users	1.046	2.000	42.000	0.360
200 users	1.482	2.000	42.000	0.239

Table 10: Levene's test for Energy Consumption

- **Workload 100 users**

  Based on Levene's Test shown in Table 10, the p-value of 100 users is 0.360. The value is greater than the significance level of 0.05. It implies that the assumption of homogeneity of variances is not violated. So, we can proceed with the one-way ANOVA test to test the hypothesis.

Cases	Sum of Squares	df	Mean Square	F	p
Compression Type	$6.890 \times 10^{+7}$	2	$3.445 \times 10^{+7}$	0.613	0.547
Residuals	$2.360 \times 10^{+9}$	42	$5.620 \times 10^{+7}$		

Table 11: ANOVA test for Energy Consumption with 100 users

  The p-value from the ANOVA test is 0.547, which is greater than the significant level of 0.05. It implies that we **fail to reject the null hypothesis**. It means that the evidence from the data is not strong enough to contradict the null hypothesis. It indicates that there are no significant differences between the compression types (None, Row, and Page).

- **Workload 200 users**

Based on Levene's Test shown in Table 10, the p-value of 200 users is 0.239. The value is greater than the significance level of 0.05. It implies that the assumption of homogeneity of variances is not violated. So, we can proceed with the one-way ANOVA test to test the hypothesis.

Cases	Sum of Squares	df	Mean Square	F	p
Compression Type	$2.407 \times 10^{+8}$	2	$1.204 \times 10^{+8}$	1.914	0.160
Residuals	$2.642 \times 10^{+9}$	42	$6.289 \times 10^{+7}$		

Table 12: ANOVA test for Energy Consumption with 200 users

The p-value from the ANOVA test is 0.160, which is less than the significant level of 0.05. It means that we **fail to reject the null hypothesis**. It means that the evidence from the data is not strong enough to contradict the null hypothesis. It indicates that there are no significant differences between the compression types (None, Row, and Page).

5.2 CPU Usage (RQ2)

5.2.1 Data Exploration

The box plot and violin plot in Figure 17 and 17 show the distribution of CPU usage (%) of different compression types with 100 and 200 users. A summary statistic of the CPU usage for different compression types with different workload levels is presented in Table 13.

	100 users			200 users		
	CPU Usage (%)			CPU Usage (%)		
	None	Row	Page	None	Row	Page
Median	8.517	13.613	14.644	9.761	14.004	13.075
Mean	9.050	13.086	14.474	9.498	13.874	13.232
Std. Deviation	1.073	1.785	2.013	1.583	1.347	1.915
Coefficient of variation	0.119	0.136	0.139	0.167	0.097	0.145
Minimum	7.517	8.303	11.565	6.811	11.661	10.059
Maximum	10.882	15.774	17.782	12.908	15.643	17.063

Table 13: Summary Statistic of CPU Usage with 100 and 200 users

Based on the summary statistics, a trend can be seen that the CPU usage with Row compression and Page compression is higher than with $None$ compression. It is shown by the mean and median that are higher for Row and $Page$ compression than $None$ compression. From the minimum and maximum values, we can see that the workload of 200 users shows

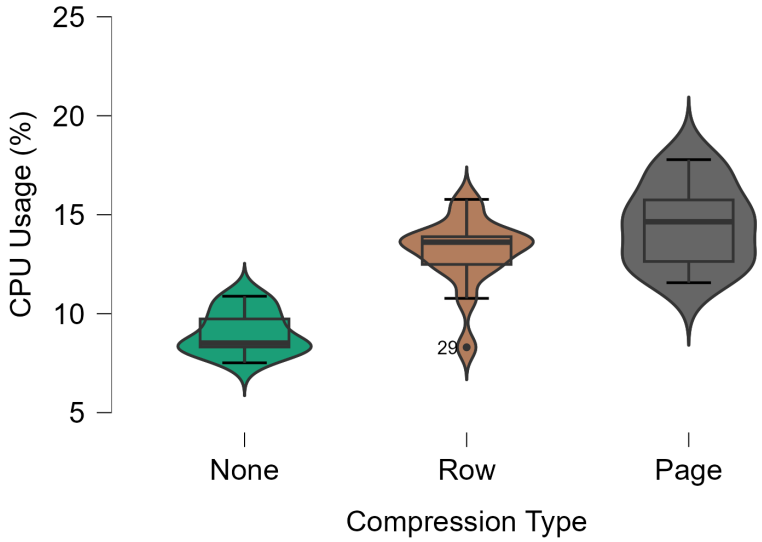

Figure 17: Boxplot and Violin plot of CPU Usage with 100 users

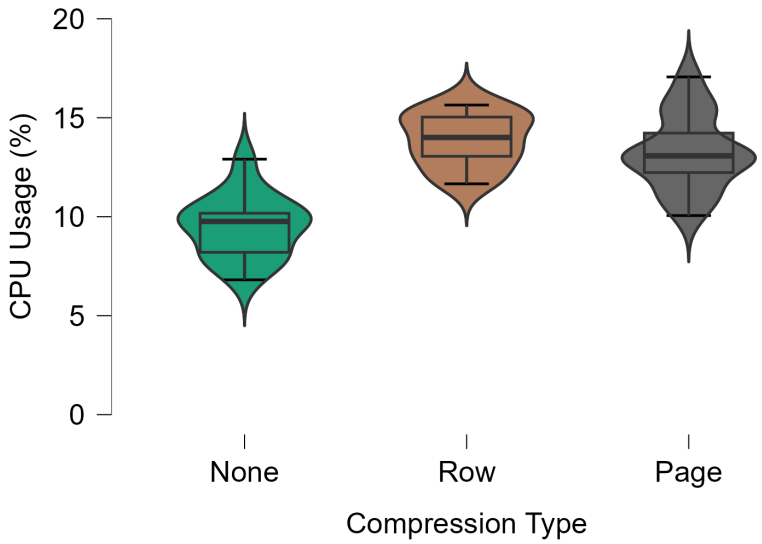

Figure 18: Boxplot and Violin plot of CPU Usage with 200 users

a broader range of values than 100 users indicating a higher degree of variability associated with a more increased workload.

The higher degree of variability also can be seen related to different compression types from the observation of higher standard deviations and coefficients of variation in $Page$ compression compared to $None$ or Row. $Page$ compression has a higher value than $None$ and Row compression, particularly in the 100 users scenario.

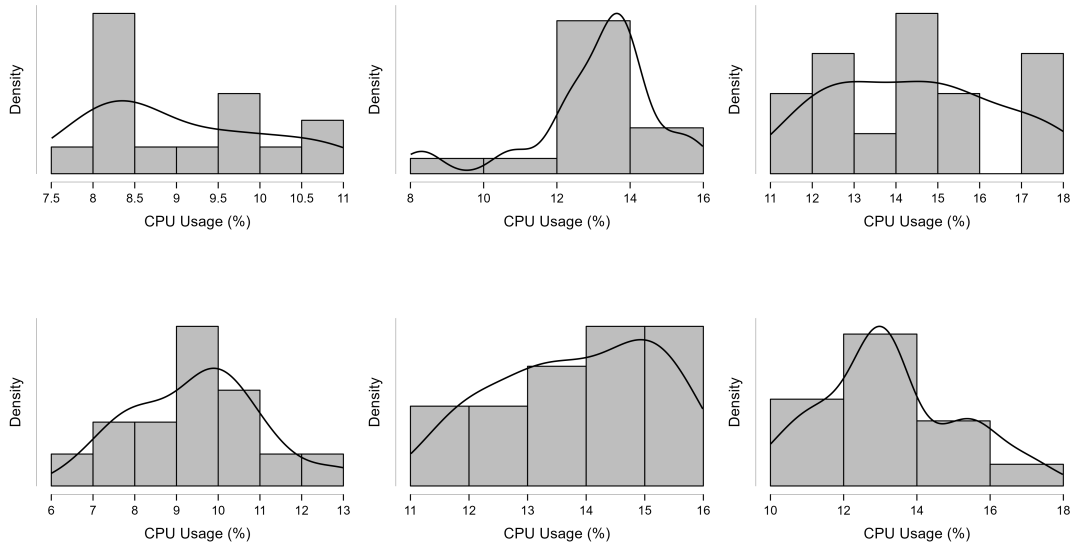

Figure 19: Distribution Density Plots of CPU Usage for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users.

5.2.2 Normality Testing

- **Density plots**. Figure 19 contains six density plots. The three plots on the top represent the CPU usage of 100 workload users with $None$, Row, and $Page$ compression from left to right. The three plots on the bottom represent 200 workload users with the same order of compression from left to right. The plots on the bottom seemed to follow a bell curve shape that leads to normally distributed data, while the top three show left-skewed and right-skewed shapes and need to be evaluated in more detail.

- **Q-Q plots**. Figure 20 gives more evidence of conformance for the top three plots on the bottom to be most likely to be normally distributed based on the shape of the diagonal line and data distribution around it. From these plots, we can see a clearer picture of the three plots on the top that can be assumed to be normally distributed with some outliers.

p-value	100 users	200 users
None	0.173	0.880
Row	0.071	0.259
Page	0.416	0.480

Table 14: Shapiro-Wilk test for CPU Usage

- **Shapiro-Wilk Test**. Table 14 shows the p-values calculated from the Shapiro-Wilk test. The p-values obtained from the Shapiro-Wilk test are greater than the significance level of 0.05, which indicates that there is insufficient evidence to reject the null

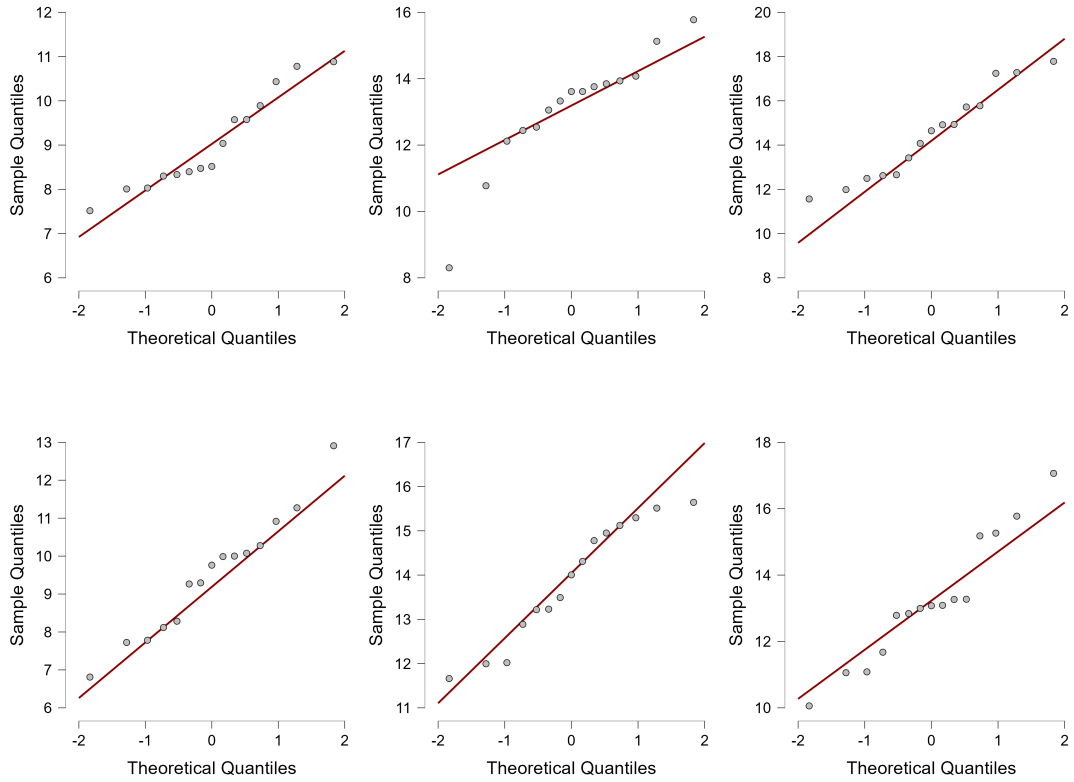

Figure 20: QQ Plots of CPU Usage for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users

hypothesis, which suggests that the data under consideration follows a normal distribution. It is assumed that the data follows a normal distribution.

5.2.3 Hypothesis Testing

As described in the experiment design, workloads are considered as a blocking factor in this experiment, we separately evaluate the hypothesis by each workload. Both workloads, 100 users and 200 users, show normally distributed data from the Shapiro-Wilk Test. We follow the data analysis plan that is detailed in Section 3.3.5.

To accommodate the hypothesis testing for normally distributed data with $1F - MT$ *"one factor, multiple treatment"*, we use a parametric test one-way ANOVA. Before performing the ANOVA test, we check the ANOVA assumptions: (i) The dependent variable should be measured on an interval scale: The dependent variable is the percentage of CPU usage measured in percentage (%) and measured in ratio scale. (ii) Normality: The data is normally distributed based on the previous Shapiro-Wilk Test. (iii) Homogeneity of Variance: The variances of the different groups being compared should be equal. To test this homogeneity, we use Levene's Test. The p-values from Levene's Test are presented in Table 15.

- **Workload 100 users**

Workload	F	df1	df2	p
100 users	4.005	2.000	42.000	0.119
200 users	0.236	2.000	42.000	0.791

Table 15: Levene's test for CPU Usage

Based on Levene's Test shown in Table 15, the p-value, 0.119, is greater than the significance level of 0.05. It implies that the assumption of homogeneity of variances is not violated. So, we can proceed with the one-way ANOVA test to test the hypothesis. The table of ANOVA test result is presented in Table 16.

Cases	Sum of Squares	df	Mean Square	F	p
Compression Type	238.174	2	119.087	42.591	7.857×10^{-11}
Residuals	117.434	42	2.796		

Table 16: ANOVA Test for CPU Usage with 100 users

The p-value from the ANOVA test is 7.857×10^{-11}, which is less than the significant level of 0.05. It allows us to **reject the null hypothesis**. This implies to the alternative statement that there is a significant difference in CPU usage between compression types (None, Row, and Page).

In order to examine differences between groups, a post hoc test known as the Tukey HSD was conducted, the findings are presented in Table 17.

		Mean Difference	SE	t	p_{tukey}
None	Row	-4.036	0.611	-6.611	1.563×10^{-7}
	Page	-5.424	0.611	-8.883	1.028×10^{-10}
Row	Page	-1.387	0.611	-2.272	0.071

Table 17: Tukey HSD Post Hoc Comparisons for CPU Usage with 100 users

Based on the Tukey HSD test result, we can deduce a significant difference in CPU Usage from the pair of None-Row and None-Page.

- **Workload 200 users**

Based on Levene's Test shown in Table 15, the p-value, 0.791, is greater than the significance level of 0.05. It implies that the assumption of homogeneity of variances is not violated. So, we can proceed with the one-way ANOVA test to test the hypothesis. The table of ANOVA test result is presented in Table 18.

Cases	Sum of Squares	df	Mean Square	F	p
Compression Type	167.528	2	83.764	31.466	4.459×10^{-9}
Residuals	111.807	42	2.662		

Table 18: ANOVA Test for CPU Usage with 200 users

The p-value from the ANOVA test is 4.459×10^{-9}, which is less than the significant level of 0.05. It allows us to **reject the null hypothesis**. This implies to the alternative statement that there is a significant difference in CPU usage between compression types (None, Row, and Page).

In order to examine differences between groups, a post hoc test known as the Tukey HSD was conducted, the findings are presented in Table 19.

		Mean Difference	SE	t	p_{tukey}
None	Row	-4.376	0.596	-7.345	1.395×10^{-8}
	Page	-3.734	0.596	-6.267	4.865×10^{-7}
Row	Page	0.642	0.596	1.078	0.533

Table 19: Tukey HSD Post Hoc Comparisons for CPU Usage with 200 users

Based on the Tukey HSD test result, we can deduce a significant difference in CPU Usage from the pair of None-Row and None-Page.

5.2.4 Effect Size Estimation

- **Workload 100 users**

To understand how the effect size of the difference, we perform Cohen's effect size estimation since the data is normally distributed, and variances are equal across the group.

		Cohen's d
None	Row	-2.414
	Page	-3.244
Row	Page	-0.830

Table 20: Effect Size Estimation: Cohen's d for CPU Usage with 100 users

Based on the result in Table 20, we see that there are significant differences between the CPU usage of different data compression methods. Cohen's d value for None-Row is -2.414, and None-Page is -3.244, indicating a very large effect size according to Cohen's d guidelines.

- **Workload 200 users**

  To understand how the effect size of the difference, we perform Cohen's effect size estimation since the data is normally distributed, and variances are equal across the group.

		Cohen's d
None	Row	-2.682
	Page	-2.288
Row	Page	0.394

Table 21: Effect Size Estimation: Cohen's d for CPU Usage with 200 users

Based on the result in Table 21, we see that there are significant differences between the CPU usage of different data compression methods. Cohen's d value for None-Row is -2.682, and None-Page is -2.288, indicating a very large effect size according to Cohen's d guidelines.

5.3 Response Time (RQ2)

5.3.1 Data Exploration

The box plot and violin plot in Figure 21 and 21 show the distribution of Response Time in milliseconds of different compression types with 100 and 200 users. A summary statistic of the Response Time for different compression types with different workload levels is presented in Table 22.

	100 users			200 users		
	Response Time (ms)			Response Time (ms)		
	None	Row	Page	None	Row	Page
Median	1102.462	1089.457	1112.064	3217.299	3214.971	2897.210
Mean	1077.207	1111.458	1180.930	2956.815	2877.051	2722.264
Std. Deviation	216.594	122.503	384.576	917.116	767.553	785.975
Coefficient of variation	0.201	0.110	0.326	0.310	0.267	0.289
Minimum	545.351	903.729	804.097	987.906	854.136	1135.244
Maximum	1331.434	1359.188	2451.848	3994.131	3623.067	3764.110

Table 22: Summary Statistic of Response Time with 100 and 200 users

For the workload of 100 users, the response times for the $None$, Row, and $Page$ show a notable similarity, with median values of 1102.462 ms, 1089.457 ms, and 1112.064 ms, respectively. The mean response times of the $Page$, $None$, and Row are also relatively quite

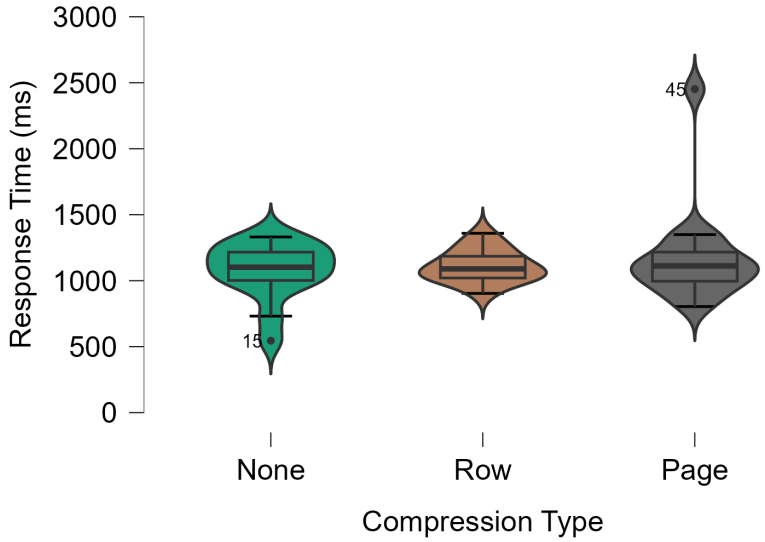

Figure 21: Boxplot and Violin plot of Response Time with 100 users

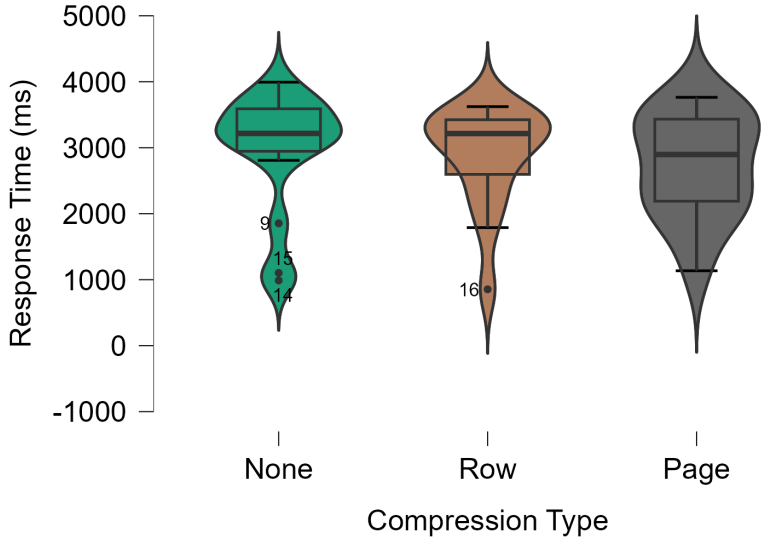

Figure 22: Boxplot and Violin plot of Response Time with 200 users

similar, with the $Page$ value having a slightly higher mean response time compared to the $None$ and Row.

For the workload of 200 users, the highest median response times are $None$ (3217.299 ms) and Row (3214.971 ms), while the $Page$ has the lowest median response time (2897.210 ms). For the mean, $None$ has the highest mean response time, 2956.815 ms, while the $Page$ layout has the lowest mean response time, 2722.264 ms.

In both instances, a noticeable trend of response times increases as the workload doubles from 100 to 200.

5.3.2 Normality Testing

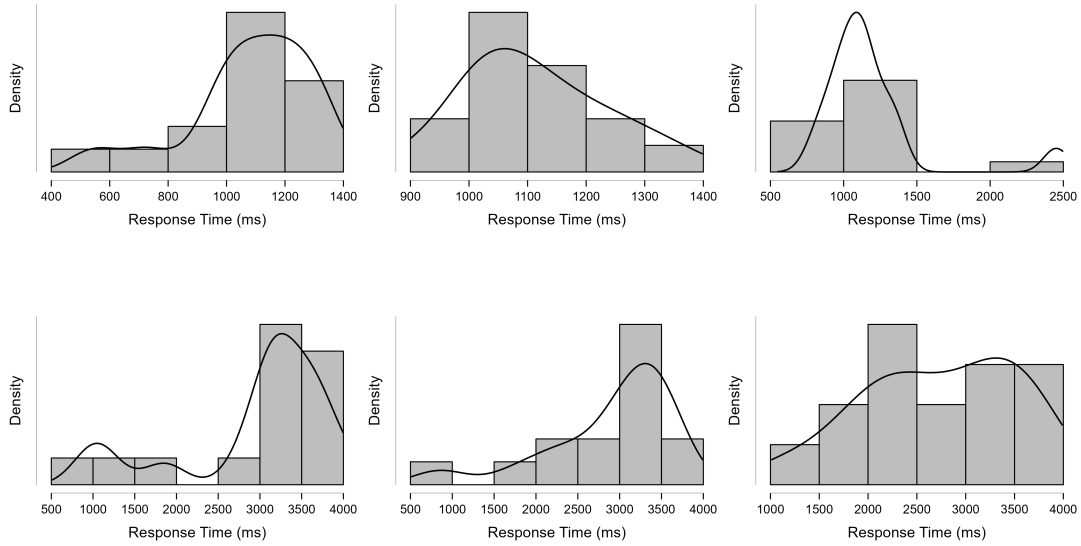

Figure 23: Distribution Density Plots of Response Time for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users

- **Density plots**. Figure 23 consists of six density plots. The three plots at the top of the figure represent the Response Time of a sample of 100 workload users under three different compression settings, namely $None$, Row, and $Page$, arranged in left-to-right order. The three plots at the bottom of the figure depict 200 workload users, each showing the same compression order, arranged in a left-to-right manner. From the figure, we assume some data are not normally distributed because the shape seems left-skewed for the $Page$ Compression plot with a workload of 100 users. For the 200 users, two plot seems to be right-skewed: $None$ compression and Row Compression.

- **Q-Q plots**. Figure 24 provides additional visual evidence to investigate whether the data is normally distributed. From these plots, we can see that the anomaly of the data appears on some plots. It shows some outliers in the $None$ compression plots in both workloads and in the Row compression plot in the workload of 200 users.

p-value	100 users	200 users
None	0.105	0.005
Row	0.875	0.009
Page	1.584×10^{-4}	0.504

Table 23: Shapiro-Wilk test for Response Time

- **Shapiro-Wilk Test**. Table 23 displays the p-values derived from the Shapiro-Wilk test. The p-values obtained from the Shapiro-Wilk confirm the anomaly from some plots. The p-values are less than the significance level of 0.05 for Page in 100 users

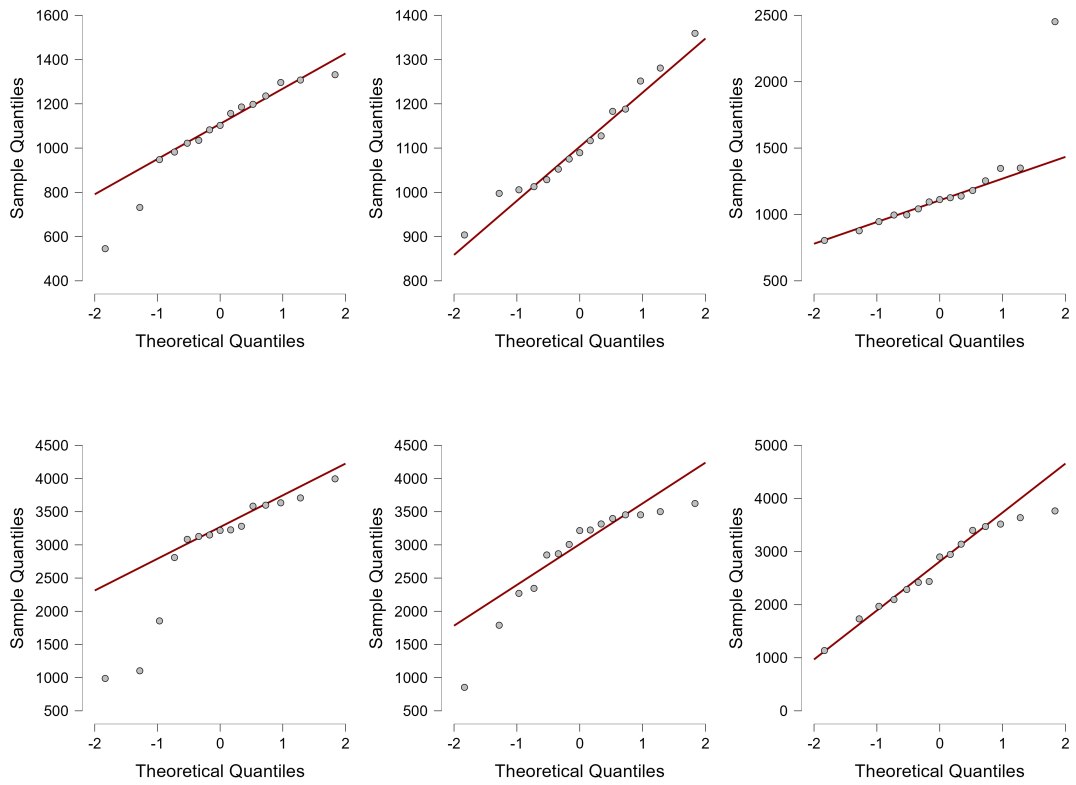

Figure 24: QQ Plots of Response Time for Compression Type (None, Row, Page) with 100 (top) and 200 (bottom) users

and for $None$ and Row in 200 users, which indicates that there is insufficient evidence to reject the null hypothesis, which suggests that the data under consideration follows a normal distribution. It is assumed that the data does not follow a normal distribution.

5.3.3 Hypothesis Testing

Since both workloads do not have normally distributed data, we utilize the Kruskal-Wallis test to test the hypothesis. We confirm the Kruskal-Wallis assumptions from our experiment: (i) The dependent variable should be measured on an interval scale: The dependent variable is the response time measured in milliseconds (ms) and measured in ratio scale. (ii) The independent variable should consist of two or more categorical, independent groups. The independent variable is the compression type and has three different values ($None$, Row, and $Page$). (iii) The observations should be independent of each other. The observations are done independently, as each trial runs a different combination implementation of various compression types and workloads and is monitored independently.

The p-values from the Kruskal-Wallis are presented in Table 24. The p-value for a workload of 100 users is 0.995, and for 200 users is 0.622, which is far greater than the significant level of 0.05. It implies to us that we **can not reject the null hypothesis**. This implies that we

Workload	Factor	Statistic	df	p
100 users	Compression Type	0.010	2	0.995
200 users	Compression Type	0.948	2	0.622

Table 24: Kruskal-Wallis test for Response Time

do not have strong enough evidence to suggest that the different compression types lead to a statistically significant difference in response time.

6 DISCUSSION

In this chapter, we continue with a comprehensive review of the results shown previously in Chapter 5. We analyze these findings in the context of our research questions, considering their value and implications. Section 6.1 focuses on reviewing the finding in the context of the RQ1, which is energy consumption. Section 6.2 discusses the results related to RQ2, which is related to performance that focuses on two metrics: CPU usage and response time.

6.1 Energy Consumption (RQ1)

RQ1*: What is the impact of data compression techniques on the energy consumption of the cloud application?*

In the context of the the worldwide challenge of climate change, the implementation of sustainability strategies have come to be as an essential component within various sectors, including software development and cloud computing. The optimization of energy consumption is a fundamental objective in the pursuit of sustainability. Considering the significant energy consumption of data centers globally, the implementation of strategies such as data compression plays a crucial role in promoting sustainable operations by effectively reducing energy consumption.

The implementation of data compression can potentially reduce the energy consumption of a software system. When data is compressed, less storage space is required. Less storage usage leads to less energy consumed by the hardware devices, whether hard drives, solid-state drives, or cloud storage servers.

Given the important role of data compression in promoting sustainable operations and reducing energy consumption, it is essential to empirically validate and understand its impact. In this experiment, the impact of data compression techniques on energy consumption in the cloud application was evaluated through a comparative analysis of energy consumption. The collected data undergo an analysis of variance (ANOVA) hypothesis test to determine whether any statistically significant differences could be detected.

Upon evaluating the application's performance under 100 users, the obtained p-value is 0.547. Moreover, a comparable experiment with an increased user load of 200 resulted in a p-value of 0.160. The observed values exceed the generally accepted significance threshold of 0.05 in both instances. Therefore, the lack of evidence to reject the null hypothesis in these situations

implies that the observed data compression techniques have a statistically insignificant effect on the energy consumption of the cloud application under experiment.

Evaluating the descriptive statistics from the experiment, the median and mean energy consumption analysis reveals an inconsistent pattern of decrease or increase in energy consumption in different compression techniques of $None$, Row, and $Page$ compression. This statement remains true for both scenarios that involve 100 users and 200 users. These results indicate that the implementation of the data compression technique does not have a consistent impact on energy consumption.

Row and $Page$ compression, particularly database-level compression, does not significantly affect the energy consumption of the cloud application. It may cause by the nature of database-level compression that primarily focuses on reducing storage needs rather than reducing data transmission. So, it does not take full advantage of the potential benefits in terms of reduced data transmission.

The inconsistent pattern in the obtained results potentially be attributed to the nature of the application subjected to the experiment and the data variability. The high degree of variability could lead to inconsistent results in energy consumption. The variability of data influences the variation of the compression ratio to reduce storage usage, which leads to energy consumption reduction. Moreover, in this experiment, we calculate the energy consumption for an overall application not only to the effect of a specific storage aspect. It also leads to inconsistent energy consumption because of the variability of the decreasing storage used and computation usage increased, like CPU Usage, which could increase the energy consumption that will be discussed further in the following discussion of RQ2.

6.2 Performance (RQ2)

***RQ2**: What is the impact of data compression techniques on the performance of cloud application?*

From a sustainability perspective, improving performance through the implementation of data compression offers a significant value. Although sustainability is frequently motivated by environmental concerns, it also presents a solid business rationale. Enhanced resource utilization has the potential to result in significant cost reductions over an extended period. This can be exemplified by decreased energy costs and the potential for lower demands on hardware infrastructure.

The implementation of data compression has the potential to optimize the effectiveness of data transmission and storage within cloud applications, thereby resulting in noticeable improvements in performance. However, the effectiveness of these enhancements is determined

by several factors, such as the characteristics of the data, the particular compression method used, and the system's architecture. By conducting an experiment, we can gain insights into these dependencies, allowing us to better understand when and how to use compression techniques for optimal performance. In this experiment, we focus primarily on two performance-related metrics, namely CPU usage and response time. These metrics will be explored in greater depth.

CPU Usage

Despite the fact that compression has the potential to decrease the quantity of data that stored, it requires computational resources for the compression and decompression processes. In certain situations, the increased computational impact may outweigh the advantages derived from decreased data size. An empirical study can help to investigate these trade-offs and get valuable insights into the optimal balance between the advantages of compression and the corresponding computational costs.

The result related to the impact of data compression techniques on CPU usage shows a significant difference from our experiment. When subjected to a workload of 100 users, the ANOVA test produced a p-value of 7.857×10^{-11} for CPU usage. This result indicates that we can reject the null hypothesis. This observation indicates that there is present a significant difference in CPU utilization between different compression techniques. Post hoc comparisons were conducted using Tukey's Honestly Significant Difference (HSD) test, which revealed statistically significant differences between the $None$-Row condition ($p = 1.563 \times 10^{-7}$) the $None$-$Page$ condition ($p = 1.028 \times 10^{-10}$), and the Row-$Page$ condition (p = 0.571). The effect sizes for Cohen's d were calculated as -2.414 for the $None$-Row comparison, -3.244 for the $None$-$Page$ comparison, and -0.830 for the Row-$Page$ comparison. These results suggest that there are strong effects observed in the $None$-Row and $None$-$Page$ comparisons.

For another variation workload of 200 users, the ANOVA test produced a p-value of 4.459×10^{-9}, which was found to be statistically significant. Therefore, the null hypothesis was rejected. Post hoc comparisons were conducted using Tukey's Honestly Significant Difference (HSD) test, which revealed statistically significant differences between the $None$-Row condition ($p = 1.395 \times 10^{-8}$), the $None$-$Page$ condition ($p = 4.865x10^{-7}$), and the Row-$Page$ condition (p = 0.533). The effect sizes for Cohen's d were calculated as -2.682 for the $None$-Row comparison, -2.288 for the $None$-$Page$ comparison, and 0.394 for the Row-$Page$ comparison. These results suggest that there are strong effects observed in the $None$-Row and $None$-$Page$ comparisons.

Both workloads give the same trend for the impact of data compression types. The result shows the strong effects observed in the $None$-Row and $None$-$Page$ comparisons. It leads to the conclusion that $None$ compression, where the data compression implementation is absent, significantly differs from Row and $Page$ compression, where data compression is present.

The findings demonstrate the significance of considering the effects on CPU utilization when planning infrastructure development. In the event that data compression will be incorporated into the overall strategy, it is important to allocate sufficient CPU resources or utilize efficient compression algorithms to mitigate potential performance bottlenecks.

Response Time

The utilization of compressed data in storage systems can result in reduced data reading and writing operations, thereby potentially enhancing access speed. Consequently, an improvement of application response time can be achieved by optimizing data storage and retrieval processes, which frequently constitute a substantial portion of operational activities in cloud-based applications.

The potential advantages of implementing data compression in relation to application response time present a convincing case for its adoption. Nevertheless, it is crucial to recognize that these potential benefits are based on theoretical comprehension and current research findings. Although theory serves as a solid basis, its real-world effects may differ based on the particular circumstances and execution. Therefore, it is essential to verify these advantages through rigorous experimentation in order to validate them. We have therefore conducted an empirical study to investigate the impact of data compression on the response time of cloud-based applications.

In our experiment, the impact of the implementation of data compression techniques on the response time of the cloud application uses the parametric test Kruskal-Wallis since the data does not normally distribute. The descriptive data summary shows that $Page$ compression has the lowest mean response time. However, according to the Kruskal-Wallis test results, these differences are not statistically significant.

From the analysis of results collected from a workload of 100 users, the Kruskal-Wallis test showed a p-value of 0.995. The obtained p-value is greater than the significance level of 0.05, indicating insufficient evidence to reject the null hypothesis. For the workload of 200 users, the p-value obtained is 0.622. The value exceeds the significance level of 0.05. These findings on both workloads imply no statistically significant variation in response times among the different compression types.

7 THREATS TO VALIDITY

This chapter discusses various types of possible threats that may impact the validity of this study. It has been analyzed using the four classifications outlined by Wohlin *et al.* [39]. The four classifications (internal validity, external validity, construct validity, and conclusion validity) are presented below.

7.1 Internal Validity

Internal validity conforms to the reliability of the findings and their applicability within the given environment. Internal validity is threatened by factors that may indicate a causal relationship [39].

- **Selection Bias**. Selection bias may arise when the testing data sets are improperly selected or fail to represent real-world scenarios accurately. A biased selection of data sets could skew the results and limit the applicability of the findings [39]. In order to address this concern, a strategy of randomization is implemented, considering criteria based on the real-world application. This approach enhances the generalizability of the results, making it more relevant and applicable.

- **Instrumentation**. The potential threat to internal validity from instrumentation comes from the possibility of inconsistency or changes in the software and hardware tools used [39]. To mitigate this threat, we ensure set up the Azure resources used to follow the same configuration: computer tier, storage size, type, and performance level are controlled to be consistent for all runs.

7.2 External Validity

External validity is a threat that questions how general the findings are [39].

- **Interaction of selection and treatment**. This threat could arise from selecting a population sample that does not represent the overall population [39]. To address this threat, the study was performed to determine the parameters for an experimental subject that reflects the usage scenario of financial institution ABC.

- **Interaction of setting and treatment**. In our study, we performed a controlled experiment. The potential threat could result from the experimental environment, wherein the components used may not follow the standards typically observed in industrial settings.

The replication of the exact experimental setting was not feasible due to limitations in cost, time, and concern of sharing information from financial institution ABC.

7.3 Construct Validity

Construct validity is a concept that involves evaluating whether the treatment that has been put into practice reflects the root of the issue and if the observed outcome truly represents the expected impact [39].

- **Inadequate Preoperational Explication of Constructs**. This threat refers to a lack of clear definitions for the constructs the experiment aims to measure or investigate [39]. In our experiment, we construct the ideas and frameworks of the experiment in advance before the execution. The GQM approach was adopted to establish this study's goal, then guide the definition of the research questions. The hypotheses, dependent and independent variables, and treatments were defined during the experiment's planning phase.

- **Mono-operation bias**. This threat happens when a construct is operationalized only by implementing a single method, situation, or condition [39]. Since our experiment design shape is 1F-MT *"one-factor multiple treatments"*, this threat may happen in our case. To mitigate this, we add the variation of workloads as a blocking factor. Using different workloads adds value to the study to see how the compression techniques perform under varying workloads. The introduction of increased variation can potentially reduce mono-operation bias and improve the generalizability of the experiment results.

7.4 Conclusion Validity

Threats to conclusion validity refer to factors that impact the ability to reach an accurate conclusion on the relationship between the treatment and the outcome of an experiment [39].

- **Low statistical power**. Low statistical power occurs when the sample size (or the number of observations) is too small to detect an effect if it does exist [39]. In order to address this threat, a specified number of treatments was determined for all the dependent variables under observation. Our study has a combination of one factor, three treatments, and a blocking factor with three levels. We then have nine trials from a combination of treatments and workload levels. In each trial, we do the repetition of 15 runs. As a result, we have a relatively large total sample size of 135.

- **Violated assumptions of statistical tests**. In order to address the potential issue of violating assumptions in the use of statistical tests, we conducted an assessment to

ensure that the corresponding sample data followed the assumptions associated with each of the statistical tests performed. We used the plots and quantitative tests to assess the normality and homogeneity of variance from the data.

8 CONCLUSIONS

This study was conducted to investigate the impact of data compression techniques, particularly row and page compression, on the energy consumption and performance of a cloud application. The obtained result showed an insignificant effect on energy consumption and response time of the application. In contrast, it showed a significant difference in CPU usage.

The absence of significant impacts on energy consumption and response time might indicate that data compression is not generally essential or beneficial under all circumstances. The observed increase in CPU utilization can be attributed to the computational demands associated with data compression and decompression operations. So, It is important to perform a careful assessment of the trade-off between CPU usage and the advantages associated with data compression.

The findings of the experiment highlight the significance of implementing a comprehensive strategy for data management. This involves a careful balance of data compression, storage, processing capabilities, and response times. In scenarios where the hardware infrastructure does not have sufficient robustness to accommodate the increased CPU load, the potential approach is to consider allocating resources toward expanding storage capacity or focusing on alternative data reduction methods instead of data compression.

REFERENCES

[1] S. Berman, L. Kesterson-Townes, A. Marshall, and R. Srivathsa, "How cloud computing enables process and business model innovation," *Strategy & Leadership*, vol. 40, pp. 27–35, Jun. 2012. DOI: 10.1108/10878571211242920.

[2] G. Boss, P. Malladi, D. Quan, L. Legregni, and H. Hall, "Cloud computing," *IBM white paper*, vol. 321, pp. 224–231, 2007.

[3] H. Susanto, M. N. Almunawar, and C. C. Kang, "A review of cloud computing evolution individual and business perspective," *SSRN Electronic Journal*, Oct. 2012. DOI: 10.2139/ssrn.2161693.

[4] Statista, *Public Cloud - Worldwide*, https://www.statista.com/outlook/tmo/public-cloud/worldwide, Accessed on: 2023-05-05, 2022.

[5] Lionel Sujay Vailshery, *Cloud storage of corporate data in organizations worldwide 2015-2022*, https://www.statista.com/statistics/1062879/worldwide-cloud-storage-of-corporate-data/, Accessed on: 2023-05-05, 2022.

[6] 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.

[7] R. Verdecchia, P. Lago, C. Ebert, and C. D. Vries, "Green it and green software," *IEEE Software*, vol. 38, pp. 7–15, 6 2021, ISSN: 19374194. DOI: 10.1109/MS.2021.3102254.

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

[9] L. Savu, "Cloud computing: Deployment models, delivery models, risks and research challenges," in *2011 International Conference on Computer and Management (CAMAN)*, 2011, pp. 1–4. DOI: 10.1109/CAMAN.2011.5778816.

[10] M. Dick, J. Drangmeister, E. Kern, and S. Naumann, "Green software engineering with agile methods," May 2013, pp. 78–85. DOI: 10.1109/GREENS.2013.6606425.

[11] C. Calero, M. Á. Moraga, and M. Piattini, "Introduction to software sustainability," in *Software Sustainability*, C. Calero, M. Á. Moraga, and M. Piattini, Eds. Cham: Springer International Publishing, 2021, pp. 1–15, ISBN: 978-3-030-69970-3. DOI: 10.1007/978-3-030-69970-3_1. [Online]. Available: https://doi.org/10.1007/978-3-030-69970-3_1.

[12] W. Oliveira, H. Matalonga, G. Pinto, F. Castor, and J. P. Fernandes, "Small changes, big impacts: Leveraging diversity to improve energy efficiency," in *Software Sustainability*, C. Calero, M. Á. Moraga, and M. Piattini, Eds. Cham: Springer International Publishing, 2021, pp. 123–152, ISBN: 978-3-030-69970-3. DOI: 10.1007/978-3-

030-69970-3_6. [Online]. Available: https://doi.org/10.1007/978-3-030-69970-3_6.

[13] A. Mishra, "Getting started with java development for azure," in *Microsoft Azure for Java Developers: Deploying Java Applications through Azure WebApp, Azure Kubernetes Service, Azure Functions, and Azure Spring Cloud*. Berkeley, CA: Apress, 2022, pp. 3–14, ISBN: 978-1-4842-8251-9. DOI: 10.1007/978-1-4842-8251-9_1. [Online]. Available: https://doi.org/10.1007/978-1-4842-8251-9_1.

[14] J. Soh, M. Copeland, A. Puca, and M. Harris, "Overview of azure platform as a service," in *Microsoft Azure: Planning, Deploying, and Managing the Cloud*. Berkeley, CA: Apress, 2020, pp. 43–55, ISBN: 978-1-4842-5958-0. DOI: 10.1007/978-1-4842-5958-0_3. [Online]. Available: https://doi.org/10.1007/978-1-4842-5958-0_3.

[15] A. Mishra, "Java for azure webapp," in *Microsoft Azure for Java Developers: Deploying Java Applications through Azure WebApp, Azure Kubernetes Service, Azure Functions, and Azure Spring Cloud*. Berkeley, CA: Apress, 2022, pp. 15–38, ISBN: 978-1-4842-8251-9. DOI: 10.1007/978-1-4842-8251-9_2. [Online]. Available: https://doi.org/10.1007/978-1-4842-8251-9_2.

[16] Microsoft, *Azure App Service Overview*, https://learn.microsoft.com/en-us/azure/app-service/overview, Accessed on 10-06-2023, 2023.

[17] A. Mishra, "Azure sql from java applications," in *Microsoft Azure for Java Developers: Deploying Java Applications through Azure WebApp, Azure Kubernetes Service, Azure Functions, and Azure Spring Cloud*. Berkeley, CA: Apress, 2022, pp. 137–159, ISBN: 978-1-4842-8251-9. DOI: 10.1007/978-1-4842-8251-9_7. [Online]. Available: https://doi.org/10.1007/978-1-4842-8251-9_7.

[18] Microsoft, *Azure SQL Database Overview*, https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql, Accessed on 10-06-2023, 2023.

[19] I. M. Pu, "Chapter 1 - introduction," in *Fundamental Data Compression*, I. M. Pu, Ed., Oxford: Butterworth-Heinemann, 2006, pp. 1–17, ISBN: 978-0-7506-6310-6. DOI: https://doi.org/10.1016/B978-075066310-6/50004-0. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780750663106500040.

[20] G. E. Blelloch, *Introduction to data compression*, 2001.

[21] A. Pavlo, *Cmu 15-721 :: Advanced database systems (spring 2023)*, Feb. 2023. [Online]. Available: https://15721.courses.cs.cmu.edu/spring2023/slides/05-compression.pdf.

[22] Microsoft, *Data Compression*, https://learn.microsoft.com/en-us/sql/relational-databases/data-compression/data-compression?view=azuresqldb-current, Accessed 05-06-2023, 2023.

[23] Microsoft, *Row Compression Implementation*, https://learn.microsoft.com/en-us/sql/relational-databases/data-compression/row-compression-implementation?view=azuresqldb-current, Accessed 05-06-2023, 2023.

[24] E. Pollack and J. Strate, "Indexing strategies," in *Expert Performance Indexing in Azure SQL and SQL Server 2022: Toward Faster Results and Lower Maintenance Both on Premises and in the Cloud*. Berkeley, CA: Apress, 2023, pp. 359–448, ISBN: 978-1-4842-9215-0. DOI: 10.1007/978-1-4842-9215-0_13. [Online]. Available: https://doi.org/10.1007/978-1-4842-9215-0_13.

[25] P. A. Carter, "Table optimizations," in *Pro SQL Server 2022 Administration: A Guide for the Modern DBA*. Berkeley, CA: Apress, 2023, pp. 219–261, ISBN: 978-1-4842-8864-1. DOI: 10.1007/978-1-4842-8864-1_7. [Online]. Available: https://doi.org/10.1007/978-1-4842-8864-1_7.

[26] Microsoft, *Page compression implementation*, https://learn.microsoft.com/en-us/sql/relational-databases/data-compression/page-compression-implementation?view=azuresqldb-current, Accessed 05-06-2023, 2023.

[27] Cloud Carbon Footprint, *Cloud Carbon Footprint*, https://www.cloudcarbonfootprint.org/, Accessed on: 2023-06-01, 2022.

[28] Cloud Carbon Footprint, *Cloud Carbon Footprint Methodology*, https://www.cloudcarbonfootprint.org/docs/methodology/, Accessed on: 2023-06-01, 2022.

[29] S. Shanmugasundaram and R. Lourdusamy, "A comparative study of text compression algorithms," *International Journal of Wisdom Based Computing*, vol. 1, no. 3, pp. 68–76, 2011.

[30] M. Al-Laham and I. M. El Emary, "Comparative study between various algorithms of data compression techniques," *IJCSNS*, vol. 7, no. 4, p. 281, 2007.

[31] G. V. Cormack, "Data compression on a database system," *Communications of the ACM*, vol. 28, no. 12, pp. 1336–1342, 1985.

[32] G. Ray, J. R. Haritsa, and S. Seshadri, "Database compression: A performance enhancement tool.," in *COMAD*, Citeseer, 1995, p. 0.

[33] V. Šalgová and M. Kvet, "The impact of table and index compression," in *2021 19th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, IEEE, 2021, pp. 327–332.

[34] K. Govinda and Y. Kumar, "Storage optimization in cloud environment using compression algorithm," *nternational Journal of Emerging Trends & Technology in Computer Science*, vol. 1, no. 1, 2012.

[35] B. Nicolae, "High throughput data-compression for cloud storage," in *International Conference on Data Management in Grid and P2P Systems*, Springer, 2010, pp. 1–12.

[36] F. NILSSON and D. MIHAJLOVIC, “Optimizing databases in the cloud based on performance and cost savings,” 2022.

[37] 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)*, IEEE, 2022, pp. 77–87.

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

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

[40] B. W. Yap and C. H. Sim, “Comparisons of various types of normality tests,” *Journal of Statistical Computation and Simulation*, vol. 81, no. 12, pp. 2141–2155, 2011.

[41] E. Ostertagova, O. Ostertag, and J. Kováč, “Methodology and application of the kruskal-wallis test,” *Applied Mechanics and Materials*, vol. 611, pp. 115–120, Aug. 2014.

[42] N. Cliff, “Dominance statistics: Ordinal analyses to answer ordinal questions.,” *Psychological bulletin*, vol. 114, no. 3, p. 494, 1993.

[43] Cloud Carbon Footprint, *Cloud Carbon Footprint - Azure*, `https://cloud-carbon-footprint.github.io/docs/azure`, Accessed on: 2023-07-30, 2022.

[44] Microsoft, *Register an app in the Microsoft identity platform*, `https://learn.microsoft.com/en-us/azure/active-directory/develop/quickstart-register-app`, Accessed on: 2023-07-30, Jun. 2023.

[45] Cloud Carbon Footprint, *Cloud Carbon Footprint - Data Persistence and Caching*, `https://cloud-carbon-footprint.github.io/docs/data-persistence-and-caching`, Accessed on: 2023-07-30, 2022.

[46] A. Thompson and B. N. Taylor, *Guide for the use of the international system of units (si) - nist*, Mar. 2008. [Online]. Available: `https://physics.nist.gov/cuu/pdf/sp811.pdf`.