Master Thesis

A First Investigation Into the Detection of Energy-related Issues in Microservice-based Systems via Anomaly Detection and Root-Cause Analysis

by

Christos Petalotis

(2739394)

First supervisor:	Ivano Malavolta
Daily supervisor:	Ivano Malavolta
Second reader:	Vincenzo Stoico

November 17, 2023

Submitted in partial fulfillment of the requirements for the joint UvA-VU degree of Master of Science in Computer Science

A First Investigation Into the Detection of Energy-related Issues in Microservice-based Systems via Anomaly Detection and Root-Cause Analysis

Christos Petalotis
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
c.petalotis@student.vu.nl

ABSTRACT

Context. As the trend toward microservice-based architecture gains traction in software application development and deployment, system complexity experiences a significant upsurge. This, in turn, has a detrimental influence on observability and maintainability. To tackle these challenges, numerous solutions that conduct anomaly detection and root cause analysis have been developed. These solutions make use of logged metrics pertaining to hardware resource utilization. However, these solutions commonly overlook the incorporation of system energy consumption. While prior research has predominantly focused on the fundamental techniques underpinning anomaly detection and root cause analysis, as well as their performance assessment in conjunction with hardware utilisation metrics, the aspect of system energy consumption has remained noticeably unaddressed.
Goal. In this paper, we examine the effectiveness of anomaly detection and root cause analysis tools for microservice-based applications, particularly emphasizing on energy consumption metrics. We investigate whether and how energy consumption metrics, can enhance anomaly detection's ability to identify energy-related anomalies and aid root cause analysis in revealing their true causes.
Method. This study is centered on two distinct systems, namely SockShop and Zahori, varying in complexity and purpose. From a large pool of tools, PyCaret was selected for anomaly detection and RCD for root cause analysis in our experimentation efforts. Metrics were gathered from the target systems using a cross-over paired comparison design, involving multiple randomised runs to establish robust measures.
Results. The anomaly detection solution employed shows weak performance in terms of adjusted recall, and F1-score, indicating poor ability detecting anomalies related to energy consumption. These values are around 19.5%, and 32.1% accordingly for both target systems. Precision stands around 90%. The results for root cause analysis in terms of precision at the top 3 predictions by the employed solution tend to be weak, with the average value (AP@3) standing at 43.6% for SockShop and 37.2% for Zahori.
Conclusions. The research findings demonstrate that anomaly detection yields precise forecasts regarding anomalies within metric segments. Nonetheless, both anomaly detection and root cause analysis exhibit a tendency to miss several instances that require identification as anomalies or their causal factors. Consequently, further research efforts are imperative to develop tools and methodologies that can harness energy consumption metrics from microservice-based applications effectively, ensuring satisfactory performance. Practitioners are encouraged to employ AD solutions together with other tools to cover a wider spectrum of energy consumption anomalies. The performance of RCA on energy consumption data and related anomalies was found lacking, thus it is advised system operators rely on other tools to find the root causes of energy consumption anomalies.

1 INTRODUCTION

The increasing adoption of microservice-based architectures has revolutionised the development and deployment of modern software applications. By decomposing complex monolithic systems into smaller, loosely coupled services, microservices offer numerous benefits such as scalability, flexibility, and faster time-to-market [1, 2]. However, microservices' distributed nature brings new challenges, particularly in monitoring and managing individual services' performance and energy consumption within the overall application [3].

The efficient utilisation of hardware resources and energy conservation are crucial considerations for the sustainable operation of high-performing microservice-based applications. Excessive energy consumption not only incurs unnecessary costs but also contributes to environmental concerns, such as carbon emissions and energy waste. To address this issue, it is imperative to employ effective techniques to detect energy anomalies within microservice-based architectures and identify their root causes to timely and effectively provide solutions to them.

However, today the task of detecting anomalies in a system is primarily focused on performance-related metrics [4, 5]. To effectively discover performance-related issues in systems utilising the microservice architecture, specialised tools that conduct anomaly detection (AD) and root-cause analysis (RCA) are usually employed [6–9].

Previous research in the field has predominantly focused on the various ways for conducting AD or RCA, whether this involves supervised [10–12] or unsupervised machine learning models [13–15], the analysis of different artefacts, such as Key performance indicators (KPIs), logs, and traces, as well as the deployment configurations employed, such as Kubernetes [16–18]. Furthermore, other studies focus specifically on the performance evaluation of such techniques based on KPIs [19, 20]. Additionally, extensive academic research has been conducted to define specific approaches to perform AD or RCA, individually or in combination, and compare their effectiveness against similar tools [9, 21].

However, a critical aspect that has been noticeably absent from these studies is the consideration of energy-related metrics. The research conducted so far has not sufficiently explored the accuracy of AD and RCA techniques in detecting energy anomalies, thereby leaving an essential aspect of microservice-based systems unattended. Consequently, such systems remain vulnerable to energy-related faults that may elude detection by system administrators and operators.

In this paper, we delve into the efficacy of anomaly detection and root cause analysis tools in the context of microservice-based applications, with a specific focus on energy consumption metrics. Our research aims to investigate the extent to which energy consumption metrics, along with performance metrics, can be efficiently utilised by anomaly detection techniques to detect energy anomalies and by root cause analysis techniques to uncover the underlying causes of such anomalies.

To achieve this, the performance of AD and RCA is assessed in terms of correctness, completeness, and accuracy. This is done by defining a ground truth using specific rules about what constitutes an anomaly in the metrics and evaluating the results AD and RCA produce according to the ground truth data.

For this exploration, two microservice-based systems have been utilised - SockShop[1] and Zahori[2]. SockShop has been developed by a group of individual developers, while Zahori is the product of a dedicated team of developers working for Panel Sistemas[3]. Moreover, specialised software is utilised to monitor and gather metrics from the selected systems. More specifically, Prometheus[4] is used to monitor and collect hardware utilisation-oriented metrics, such as CPU and RAM usage, while SmartWatts[5] is employed to monitor the power consumption of the underlying system. Additionally, we conduct anomaly detection on the collected KPIs and energy consumption metrics using the PyCaret[6] solution by utilising multiple of the provided Machine Learning (ML) models that can be used with this solution. If anomalies are detected in the metrics of the underlying system, then RCA is conducted using the RCD[7] tool. These solutions were selected from a large pool of considered tools because of their ease of use, the low effort for setting them up, their flexibility in performing their functionality using different configurations, and their ability to accept multiple types of data rather than have requirements on the types of metrics that can be used for their operation, constraining their applicability to only those types.

Furthermore, this study introduces a pipeline to measure systems of interest and includes the steps to perform anomaly detection and root-cause analysis on these systems. We name this pipeline *Data-Driven Anomaly Diagnosis* (DDAD), as data constitutes the foundation on which anomalies in a system are found and analysed, with the ultimate goal of fixing problems and errors related to diagnosed energy anomalies.

The target audience of this paper is mainly system operators, who are in charge of managing, maintaining and adjusting microservice-based systems, but also researchers in the field of AD or RCA and of microservices more generally. As stated earlier, system operators could benefit a lot from the results of this study, as they could utilise anomaly detection and root-cause analysis techniques to swiftly detect issues that affect the systems' energy consumption, making these systems more sustainable and cost-effective. Additionally, operators can potentially proactively detect and tackle system errors and other issues using energy consumption as an early indicator of abnormal utilisation of system resources. Moreover, the findings of this study can be used as hard evidence about the current state of AD and RCA techniques in terms of their appropriateness of operating using energy consumption metrics and as a first touch-point by researchers to conduct further studies on this topic and expand the knowledge of the community on the potential benefits of using AD and RCA techniques on energy consumption data.

In summary, this paper makes three main contributions.

⋆ Empirically assesses the effectiveness of anomaly detection techniques in discovering energy consumption anomalies in microservice-based systems.

⋆ Empirically assesses the effectiveness of root cause analysis techniques in identifying the root causes of energy consumption anomalies in microservice-based systems.

⋆ Introduces a method, called Data-Driven Anomaly Diagnosis (or DDAD), that can be followed to perform anomaly detection and root-cause analysis on microservice-based applications.

This study is part of a research project conducted in collaboration with Luka Krumpak. Sections 1 - 6 and 9 were completed partially in collaboration, while the rest of the sections are unique to this paper. The replication package for this study is available on GitHub[8].

2 BACKGROUND

2.1 Microservice Architecture

Microservices architecture (MSA) is a software development approach that involves breaking down a monolithic application into smaller, independent services [22]. These self-contained services can be developed, deployed, and scaled autonomously, resulting in enhanced flexibility and agility throughout the development process.

In the context of MSA, each service is designed to perform a specific function and only this function. The services communicate with other services through APIs, ensuring seamless interactions between components. Unlike monolithic approaches, a key characteristic of microservices is having the services loosely coupled. This approach improves flexibility and scalability, enabling independent development, deployment, and maintenance of each service.

By employing this approach, the complexities of managing and maintaining large applications are significantly reduced. One of the key benefits of MSA lies in its fault tolerance and resilience. Since each service operates independently, issues in one service are less likely to cascade and affect the entire system. This isolation contributes to a more stable and robust application.

However, it is worth noting that service faults are not absent. Based on a survey conducted by Markos et al [23], a significant proportion of interviewed professionals (57% and 49%, respectively) consider the challenges of testing the entire system and detecting

[1] https://github.com/microservices-demo/microservices-demo
[2] https://zahori.io
[3] https://www.panel.es/
[4] https://prometheus.io/
[5] https://github.com/powerapi-ng/smartwatts-formula
[6] https://pycaret.org/
[7] https://github.com/azamikram/rcd
[8] https://github.com/vuDevOps

service faults to be crucial in the context of microservices. Consequently, our study aims to explore the possibility of leveraging energy performance metrics to facilitate the detection of service faults and enable informed decisions regarding optimising system architecture and deployment strategies, thereby minimising environmental impact and operational expenses.

2.2 Containerisation

When it comes to containerisation, there are multiple solutions that are available, most notably Docker. Each service has its benefits and applications. Nevertheless, Docker remains one of the most commonly used and popular containerisation platforms [24].

Docker is a platform that enables developers to create and manage containers, which are lightweight, portable, and self-contained environments for running applications. Containers are used extensively in microservices architecture to help break down large applications into smaller, more manageable components. By using Docker containers, developers can package their applications with all the necessary dependencies and configurations, making deploying and scaling their services more manageable. In addition, Docker's containerization technology guarantees uniformity across various environments, minimizing any possible problems that may arise from differences between development, testing, and production configurations.

2.3 Anomaly Detection & Root Cause Analysis

Anomaly detection and root cause analysis are essential data analysis and problem-solving techniques. Anomaly detection involves identifying data points or patterns that deviate significantly from the norm within a given dataset. These anomalies can represent critical insights, such as irregular energy consumption patterns, unexpected microservice performance fluctuations, or atypical system behaviours.

Root cause analysis, on the other hand, aims to uncover the underlying factors or triggers that lead to observed anomalies or problems. By delving into the root causes, one can understand why certain events occur and devise effective strategies to address and prevent them in the future. In unison, the fusion of anomaly detection and root cause analysis empowers organizations to manage risks and optimize operations proactively.

2.4 Measuring energy consumption

2.4.1 Smartwatts. When it comes to energy profiling techniques and power management frameworks, the options available are quite diverse[25, 26]. These methods encompass a wide range of granularity when collecting metrics, varying levels of accuracy in measurements, different degrees of performance overhead, and distinct levels of reproducibility. To ensure an optimal choice for dynamic power monitoring, careful consideration of these factors is essential.

In this study, we have chosen to use PowerAPI[9], an open-source project that provides tools and libraries for monitoring energy consumption at a process level. Within the PowerAPI project, there is a component called SmartWatts[10], which is specifically designed to support power monitoring and management in real time. This component adopts online calibration to automatically adjust the CPU and DRAM power models, resulting in more accurate runtime power estimations for docker containers [27].

[9]https://powerapi.org/

[10]https://github.com/powerapi-ng/smartwatts-formula

3 RELATED WORK

Numerous scientific papers discuss techniques for anomaly detection, root-cause analysis, or both on microservice-based applications and ways of measuring systems' energy consumption. This section provides a overview of some important work related to our study and how they could help aid in our research.

Jay et al. [28] emphasise the importance of considering the energy aspect of digital activities and explores various software-based power meters that can help realise the energy impact of such activities. They assess the accuracy of these tools in measuring power consumption and compare their performances. The focus is on measuring the energy consumption of computer nodes, applications and individual services that are CPU or GPU-intensive. The environmental impact of applications is also evaluated by measuring the energy consumption required by the systems used for the conducted experiments. Our study, however, focuses on microservice-based applications and whether anomaly detection and root-cause analysis techniques can utilise the gathered energy metrics. The research carried out by Jay et al. is the initial step in the process followed in this study, which involves capturing the energy consumption of software systems. Their findings will be useful in conducting AD and RCA to detect energy anomalies in a system by providing an indication of accurate software-based power meters based on the target system's characteristics.

In a separate study, Noureddine et al. [26] has thoroughly evaluated a range of methodologies for quantifying both hardware and software energy consumption. The current energy measurement landscape is divided into three categories: hardware measurement, power models, and software measurement. While hardware measurement offers unparalleled precision, it operates at a more granular level and, as such, requires additional hardware and greater scalability. On the other hand, power models can suffer from generality or platform dependency. However, when deployed, they provide a clearer insight into how and where energy is spent in software. The present study differs from that of Noureddine et al. in that it does not evaluate various methods for determining energy consumption. Instead, the focus is solely on measuring the energy consumption of microservices. Nonetheless, powermodels have been identified as the most suitable for our research.

In another study, Soldani et al. [4] direct their attention to the research conducted on AD and RCA techniques that enable and expedite the detection of system failure symptoms and the root causes for observed anomalies. In particular, they focus on tools that can be applied to systems developed using a microservice architecture. Their primary goal is to present these techniques' main characteristics, shortcomings, and requirements. However, none of the explored techniques for AD and RCA centres on the energy consumption of the underlying systems and the detection of energy-related irregularities. Unlike Soldani et al., we aim to evaluate how effective AD and RCA techniques are in detecting energy anomalies and their causes by integrating energy consumption metrics into their process. In both studies, the interest is in microservice-based systems. Nevertheless, we are concerned about AD and RCA as processes and do not investigate which tool performs better. Similarly to the research conducted by Jay et al., the paper by Soldani et al. is complementary to our research as it can facilitate the selection of the appropriate technique to conduct AD or RCA based on the system characteristics and the need for detecting energy consumption anomalies.

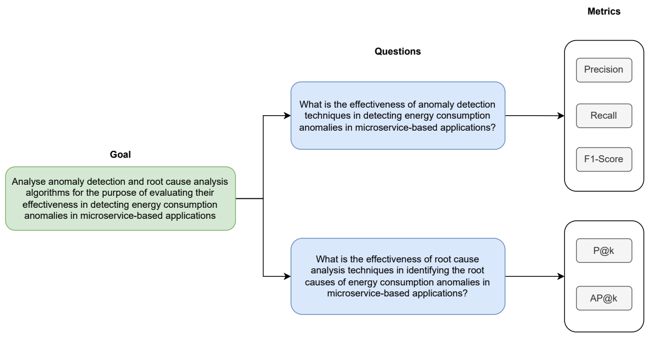
Jiang et al. [29] recognise the difficulty in monitoring microservice architectures and discovering anomalies. They challenge the degree to which diagnostic metrics are appropriate for performing such processes and suggest an alternative solution that depends on only minimal API logs. A detailed description of their approach and its performance and a comparison with previously developed solutions are provided. Although both Jiang et al. and our study focus on microservice-based systems and the discovery of anomalies in these systems' metrics, we are interested in the sustainability aspect of the solution rather than the way the solution is designed and developed. Additionally, the underlying metrics utilised to conduct the considered anomaly detection techniques differ in that Jiang et al. highlight the adequacy of sparse API logs for AD, while our research attempts to realise how effective AD and RCA processes can be when they utilise performance-oriented metrics and target energy anomalies.

The subsequent study by Capra et al. [30] underscores the significance of investigating energy anomalies, as application software significantly influences energy consumption. While executing the application, the server's power consumption surged by as much as 72 percent compared to the power drawn during idle periods. Moreover, the choice of software introduced substantial variability in energy usage. Furthermore, energy consumption is lower when software is structured with fewer layers and adeptly leverages application development environments. With the popularity of microservices, investigating methods to identify and address these energy anomalies emerges as a crucial endeavour, poised to provide a fine-grained explanation of energy anomalies which otherwise might have been missed.

In summary, this paper presents the first study into detecting energy-related problems in microservice-based systems using anomaly detection and root cause analysis. By utilising advanced techniques, our study aims to contribute to the field by providing a comprehensive understanding of energy consumption patterns within microservices architectures, which is critical for system performance and sustainability.

4 EXPERIMENT DEFINITION

The scope of the experiment comprises applications developed using a microservice-oriented architectural style, focusing on exploring energy consumption as an appropriate parameter used to conduct anomaly detection and root-cause analysis on such systems. The goal is to empirically evaluate the extent to which the outcomes generated by such techniques include predetermined, deliberately introduced anomalies related to performance and energy consumption within microservice-based applications. We investigate the efficacy of integrating energy metrics into said AD and RCA techniques to fulfil their intended purpose.



Metrics
Questions
Goal
Analyse anomaly detection and root cause analysis algorithms for the purpose of evaluating their effectiveness in detecting energy consumption anomalies in microservice-based applications
What is the effectiveness of anomaly detection techniques in detecting energy consumption anomalies in microservice-based applications?
What is the effectiveness of root cause analysis techniques in identifying the root causes of energy consumption anomalies in microservice-based applications?
Precision
Recall
F1-Score
P@k
AP@k

Figure 1: Visual Representation of the GQM

In this section, we establish the objectives of the presented experiment, delineate the research questions addressed in this paper, and specify the metrics employed to achieve these objectives. More specifically, the goal of the experiment can be stated formally using the GQM framework [31] in the following manner:

"*Analyse anomaly detection and root cause analysis algorithms for the purpose of evaluation with respect to their effectiveness in detecting energy consumption anomalies as seen from the point of view of system operators and researchers in the context of microservice-based applications*".

A visual representation of this definition is shown in Figure 1.

The motivation behind such exploration stems from the fact that despite the abundance of AD and RCA tools for analysing anomalies in microservice-based systems and finding the root of these problems, energy consumption is notably absent from the documentation and corresponding studies of such tools. As systems get bigger and more powerful, and organisations are looking to minimise their infrastructure-related costs, it is important to understand the energy consumption anomalies that arise on microservice-based applications so that system operators, or other responsible parties, can quickly detect and address the root causes of these situations. At the same time, abnormally low or high energy measures might indicate an issue in the underlying system before the problem has propagated to other areas or even has any notable effects on the system's operation. Such information could be used proactively to address dysfunctional components within the system, thereby mitigating the escalation of issues and their subsequent propagation to other facets of the application. For example, a web application system that uses a cache to serve some popular user requests might have a faulty cache strategy or error in the code for the cache logic leading to extensive use of the system's database system and, subsequently, to an abnormally increased energy consumption by the corresponding service. Such a conclusion might be drawn even before the abnormally increased operational usage of the database is detected, by analysing energy-related metrics, allowing the system operators to realise the presence of a system irregularity at an earlier time than usual.

To investigate the aforementioned objective of this study, we explore the answers to the following research questions:

RQ1: What is the effectiveness of anomaly detection techniques in detecting energy consumption anomalies in microservice-based applications?

RQ2: What is the effectiveness of root cause analysis techniques in identifying the root causes of energy consumption anomalies in microservice-based applications?

The effort to assess AD and RCA tools in their ability to correctly identify anomalies in a system's energy consumption logged metrics focuses on the experimentation of two microservice-based systems and proposes an indicative pipeline to conduct AD and RCA in a partly-automated way. Firstly, a research-oriented reference application, called SockShop, is used to perform a small-scale exploration and refinement of the AD and RCA pipeline used in this study and to verify its execution reliability. A reliable, fully-operational pipeline is then utilised to evaluate how effective AD and RCA tools are in integrating energy consumption metrics on a production-level application that is deployed and applied "in the wild". The industrial system employed is Zahori, an open-source RPA tool for process automation developed by Panel Sistemas[11]. We employ PyCaret and RCD to perform anomaly detection and root-cause analysis on these systems. Both tools utilise machine learning techniques to achieve their indented functionality.

This experimentation is based on the power consumption of each service of each system, which is measured in Watts using specialised software called SmartWatts. The power consumption values are translated to energy consumption values for the purpose of our experiment. Other performance-oriented metrics are gathered from the system as well, such as CPU and Memory (RAM) usage, to be used by the tools. For this purpose, the underlying systems have been instrumented using Prometheus[12].

The assessment of the performance of AD and RCA tools in identifying anomalies within a microservice-based system and discovering the underlying root cause of these anomalies, respectively, is conducted through three distinct sections: correctness, completeness, and accuracy. Correctness is used to evaluate the performance of both AD and RCA, while completeness and accuracy are used solely for the evaluation of AD's performance.

The correctness of the results for AD and RCA is evaluated using the precision metric. This metric enables the measurement of the relevance of the generated results by indicating the percentage of anomalies or root causes that are accurately identified as such. This assessment considers the anomalies related to energy consumption detected by the AD process, whereas it considers all the root causes that RCA identifies. In particular, we use an adjusted version of precision for AD, while for RCA, we compute the precision at top k, signified by P@k, and the average precision at top k, shown as AP@k. These metrics are important, as low levels of correctness could indicate that AD or RCA cannot operate efficiently when they integrate energy consumption metrics into their process.

Recall (also known as sensitivity or true positive rate [32]) is used to measure the extent of completeness in the outcomes generated by the AD process. This metric reveals the process' ability to identify pertinent results, specifically services exhibiting energy anomalies in the case of AD. When the completeness level is low, employing AD may lead to unnecessary overhead, which fails to generate useful conclusions. This overhead manifests as the requirement to appropriately instrument the target system to expose energy consumption metrics, as well as tasks for data collection and processing, before passing them to AD tools.

The accuracy of the results AD generates is gauged using the F1-score metric. The value of this metric is a measurement of both the completeness and the correctness of the tools and is defined as the harmonic mean of precision and recall. This metric is useful when an imbalance between positive and negative samples is expected. This is true in our case, as the number of dataset entries with anomalies is expected to be significantly higher compared to the ones with non-anomalous values. For both recall and F1-score, we compute the adjusted versions. This is further discussed in Section 5.

Using a partially automated pipeline to explore irregularities and their root causes is crucial for system operators, as it allows them to focus on resolving system problems and errors and not dawdling on inspecting and detecting potential issues. This way, their workflow is significantly accelerated while the system is monitored and analysed for potential issues. Moreover, this pipeline contributes to the automation of a large part of the experiment conducted as part of this research, enabling long and multiple runs of the AD and RCA cycle and repeating the same process with different configurations. Some steps of this pipeline need to be conducted manually by system operators. This is required to set up the application being inspected and the tools used for AD and RCA. Human intervention is also essential to make a final assessment of the generated results. This pipeline is further discussed in Sections 5 and 6.

The findings of this study yield insights and empirical evidence concerning anomalies in energy consumption exhibited by microservice-based applications, which can be utilized as input in anomaly detection and root-cause analysis techniques. Incorporating such metrics, previously overlooked in the context of anomaly detection and root-cause analysis of microservice-based applications, has the potential to foster a software engineering approach that prioritizes sustainability.

5 EXPERIMENT PLANNING

In an effort to establish a transparent and unambiguous experimental environment, the present section defines the contextual background and outlines the research plan adopted in this study. This approach facilitates the derivation of robust and reproducible results. To ensure methodological rigour, we adhere to the guidelines proposed by Wohin et al. [33]

5.1 Subjects Selection

The experiment carried out as an integral part of this study focuses on the offline analysis of the results generated by anomaly detection and root-cause analysis techniques when applied to systems built upon a microservice-oriented architecture. Specifically, the emphasis lies on evaluating the effectiveness of AD and RCA processes when operating on such systems by using measures related to the power consumption of the individual services that constitute a system, along with metrics about hardware resources

[11]https://www.panel.es/
[12]https://prometheus.io/

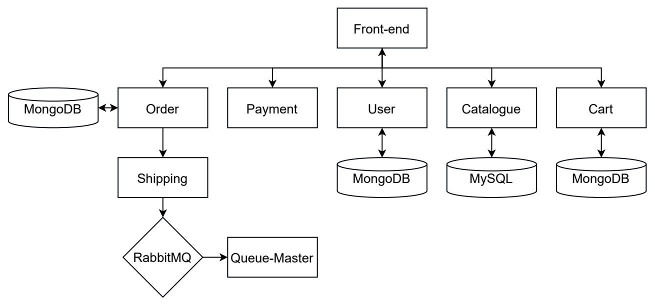


Figure 2: SockShop Architecture

utilisation by these services as inputs. In particular, anomaly detection is conducted solely on energy consumption metrics, with RCA utilising the whole spectrum of metrics collected from the underlying systems, namely hardware resource utilisation and energy consumption metrics. This process takes place after the system has operated for a period of five minutes and the conclusions of AD and RCA have been generated and stored in CSV files for further exploration by human operators.

The subject group for this experiment is comprised of two systems of different complexity levels; a research-oriented reference application of medium complexity called SockShop and a industrial system named Zahori, that is developed by Panel Sistemas and is utilised in the real world for secure and efficient process automation. Panel Sistemas is a partner organisation in the conducted research that contributes to the advancement of techniques that enhance sustainability awareness while optimising its development and deployment process to account for energy anomalies in Zahori.

Below you can find a brief description of each system that is used as subject in our experiment.

5.1.1 SockShop. This application is a collective creation of Weaveworks[13] and Container Solutions[14] designed to simulate the user-facing part of an e-commerce website that sells socks. It is a widely used microservice designed to serve as an ideal setting for testing new methodologies, technologies, and microservices-related tools [3, 9, 16, 34–37]. As illustrated in Figure 2, this microservice is made up of 13 different components that communicate with each other using REST over HTTP. The front-end is where users can make requests, while the catalogue provides product information and the carts hold shopping carts. The user service handles user authentication and stores user accounts, including payment cards and addresses. The orders service is responsible for placing orders from carts after a user logs in through the user service, and then processes payment and shipping through the payment and shipping services separately. It was intentionally designed to provide as many microservices as feasible, utilizing frameworks such as Spring Boot, Go Kit, and Node.js in its development.

5.1.2 Zahori. This is an open-source software solution that is developed by Panel. It is used in QA testing to automate repetitive manual tasks on computer or software systems. It consists of two main components, as shown in Figure 3 - the server and the automated processes.

[13]https://www.weave.works/
[14]https://www.container-solutions.com/

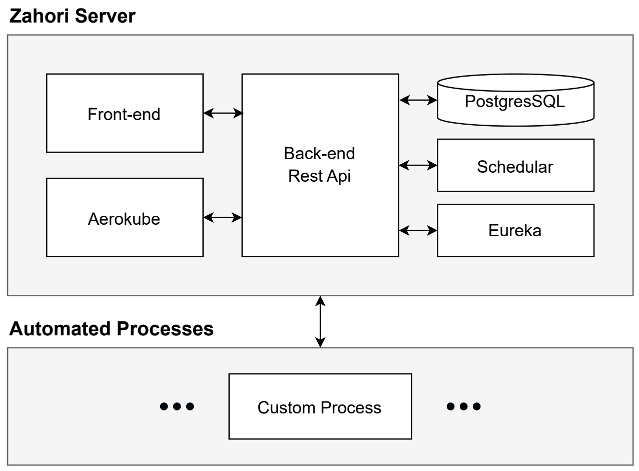


Figure 3: Zahori Architecture

The server is responsible for coordinating the execution of test cases, defining test cases, configuring and scheduling processes to be executed across multiple browsers and resolutions, displaying the progress of running executions, and presenting completed execution results. It offers both a GUI and a REST API for communication with automated processes and other components. Service discovery is facilitated by Eureka[15] to ensure communication between the server and processes. Aerokube/Selenoid[16] serves as the testing platform responsible for running browsers in isolated Docker containers, while Zahori utilizes PostgreSQL[17] as a relational database for persistence.

The automated processes are responsible for implementing the automated tests utilizing Selenium[18]. Each process communicates with the server via a REST API, and is capable of generating evidence of executed tests, including logs, screenshots, documents, videos, and HAR files. With Zahorí, users can streamline their QA testing process and improve their overall efficiency.

In order to draw meaningful and reliable conclusions about multiple types of systems and make the experiment applicable to a wider spectrum of software solutions, our experiment incorporates both relatively simple and more complex applications. Furthermore, we evaluate the performance of more than one AD algorithms. This approach enhances the applicability of our experiment, ensuring that the findings can be applied to systems of various complexity levels, without focusing only on a specific solution.

Both SockShop and Zahori are microservice-based applications. The deployment of these applications involves leveraging Docker[19] containers and Docker Compose[20], a tool that facilitates the definition and deployment of multi-container applications by specifying

[15]https://cloud.spring.io/spring-cloud-netflix/multi/multi_spring-cloud-eureka-server.html
[16]https://aerokube.com/selenoid/
[17]https://www.postgresql.org/
[18]https://www.selenium.dev/
[19]https://www.docker.com/
[20]https://docs.docker.com/compose/

services, networks, and volumes using YAML files. While Docker Compose was employed for this experiment, alternative solutions such as Kubernetes[21] or other container orchestration mechanisms could also be utilized for deploying the target applications. In any case, it is required that the systems used as subjects in this study are developed using the microservices architectural style and the required metrics can be collected from such systems. There are no other requirements, such as a specific programming language used to develop the application or a certain application size or complexity level. This approach allows for flexibility in the choice of a solution for container orchestration and underlying application while ensuring adherence to the core principles of microservices and the successful collection of relevant metrics.

5.2 Experimental Variables

In this study, we explore whether AD and RCA techniques can utilise energy consumption data to achieve their core functionality and detect anomalies or root causes related to energy consumption anomalies. To assess this ability, anomalies need to be present within a system that potentially trigger the AD and RCA cycle, allowing us to generate statistics about the tools' performance. To better understand the degree to which these tools can operate properly using energy metrics, we decided to inject anomalies of various levels for various hardware components, namely CPU, memory (RAM), and file system for each service. In other words, the presence of a hardware resource anomaly constitutes the independent variable of our experiment. The combination of the resource that is stressed and the stress' intensity are the characteristics that differentiate between the treatments of this factor variable.

Anomalies are introduced into the target system in the form of stress of individual system services. The solution that is employed for the stress is stress-ng[22], which is further discussed in Section 6. Each hardware component is stressed at two intensity levels. These levels correspond to 50% and 90% utilisation of the target hardware component's max capacity that is assigned to the stressed microservice. These levels of hardware stressing were used to explore how efficiently AD and RCA perform on both moderate and high intensity anomalies. Their effectiveness is evaluated after combining the results they generate regardless of the intensity level of the stress applied.

Each service within SockShop has 1Gb of RAM and of 1Gb of disk, while each service in Zahori is given 2Gb of RAM and 2Gb of disk. Regarding processing power, each service in both of the systems is allocated 1 CPU that is stressed at different levels. These resources were found to be the minimum amount needed by the systems to operate properly.

A 50% stress test in SockShop increases the RAM usage by 512Mb when a memory-related anomaly is injected. The same amount of bytes is written on the disk of the target application service when a file system anomaly is injected. These values increase to 1024Mb of excess RAM or disk utilisation when considering Zahori, as in this case the allocated resources are 2Gb for memory and disk capacity. Accordingly, when an anomaly of high intensity is injected to the system (i.e. increase of resource utilisation of 90%) the RAM or disk usage of the target SockShop service is increased by 921Mb, when the type of stress targets the memory or disk of the service, respectively. Stressing the Zahori services at 90% capacity results to an increase of 1.8Gb in the utilisation of the allocated RAM or disk, based on whether bytes are written using the memory or file system stress-ng stressor. Lastly, the CPU anomalies refer to the percentage of the maximum load the processor can take that is directed towards the CPUs. We stress the CPU of each of the target services at levels 50% and 90%.

While the different treatments are applied on the target services, monitoring tools are utilised to extract metrics about resource utilisation and power consumption of these services. This data is then provided as input to AD and RCA techniques. This process allows the tools to yield outcomes that undergo statistical analysis to assess their effectiveness. We quantify this performance using the statistical metrics of precision, recall and F1-score (F1 for short) in the case of anomaly detection. Specifically, we use the *adjusted* version of these metrics, which means that an interval of metrics, rather than a single point, is considered to compute these metrics. This approach is followed because anomalies often occur continuously, and not just on a certain moment, forming contiguous segments of time in which a system is affected by the anomaly. This leads to point-adjusted (range-based) evaluation metrics for anomaly detection [38]. The evaluation of the results that RCA produces is conducted using the metrics of precision at top k, signified as PR@k, and the average precision at top k, signified as AP@k [21]. These are the most commonly used metrics to evaluate ranked results. [39]. All of these metrics are used to answer the research questions defined in Section 4. To answer RQ1 we consider the process of anomaly detection, whereas answering RQ2 involves analysing the performance of the root-cause analysis process.

Following the concept of point-adjust metrics, the adjusted precision (P) of anomaly detection is measured by considering the number of correctly and wrongly identified anomalous segments during the operation of the system. Formula 1 refers to this metric. The focus is on the detection of anomalies related to the energy consumption of system services. A segment is considered to be correctly detected as anomalous if any point in its range is deemed correctly as anomalous according to the ground truth. In this case, all other points in this interval are treated as they can be properly detected as anomalies as well. If a segment that does not contain any anomalous data points is marked as anomalous by anomaly detection, the whole segment is incorrectly deemed anomalous. This way we get True Positives (TP) and False Positives (FP), respectively. If no points within a segment are marked as anomalous in the ground truth, the whole segment is deemed non-anomalous. An illustration of the decision on true positives, false positives, and false negatives (discussed below) is depicted in Figure 4

$$adjusted(P_{AD}) = \frac{TP}{TP + FP} \tag{1}$$

For RCA, precision at the top k measures the proportion of correct predictions among the top K items the RCA solution identifies. Per Li et al. [21], the calculation of this metric is achieved using formula 2. In this formula, $R[i]$ denotes the rank of a detected root cause, and gt signifies the actual, known root cause of the system

[21]https://kubernetes.io/

[22]https://wiki.ubuntu.com/Kernel/Reference/stress-ng

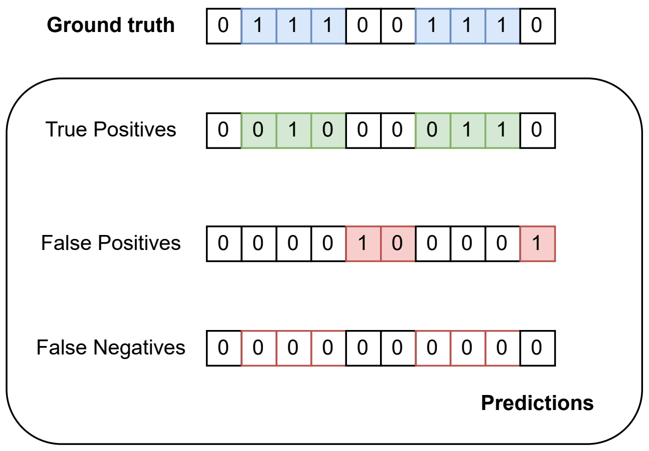


Figure 4: An illustration of the way true positives, false positives, and false negatives are derived using the notion of segments and point-adjust metrics. Green signifies correct identification of anomalous segment. Red indicates incorrect characterisation of non-anomalous segment as anomalous. Red border shows a miss in detecting anomalous segments as so.

anomalies, constituting the ground truth. Moreover, A is the set of given anomalies, i.e. the sum of all the considered datasets that correspond to trials during which anomalies are injected to services' hardware components. In our experiment, the ground truth for RCA is always comprised of only one entry, the injected anomaly in the form of the stress of hardware resources. For this reason, when calculating this metric, the denominator in formula 3 is always going to have a value of 1.

$$P@k_{RCA} = \frac{1}{|A|} \sum_{\alpha \in A} \frac{\sum_{i<k} (R[i] \in gt)}{(min(k, |gt|))} \tag{2}$$

For root cause analysis, the average precision at k quantifies the overall performance of the process, as follows:

$$AP@k = \frac{1}{k} \sum_{1 \leq j \leq k} P@j \tag{3}$$

In terms of adjusted recall (R), we examine the quantity of TP and FN as follows:

$$adjusted(R) = \frac{TP}{TP + FN} \tag{4}$$

In the case of AD, a false negative (FN) indicates the number of times an anomalous segment is not marked as such. For RCA, false negatives are not relevant.

Lastly, the evaluation of AD's accuracy is conducted by considering the f-score (F) metric. The F1-score is a metric that represents the harmonic mean of precision and recall. It is defined as follows:

$$adjusted(F1) = 2 \times \frac{P \times R}{P + R} \tag{5}$$

Again, for anomaly detection, we evaluate performance using the adjusted F1-score. This means that the computation of its value is conducted using the adjusted precision and recall metrics.

At this point, it is important to specify what is considered to be an anomaly in the collected metrics for the target system. In other words, we need to lay down the rules that define the ground truth about the anomalous state of a certain segment of evaluated points. This is crucial to allow for the appropriate labelling of the results of the anomaly detection process as correctly marked or not. Subsequently, this affects the calculations for the dependent variables defined for AD in this experiment, namely the adjusted versions of precision, recall, and F1-score.

The need for this arises from the fact that an anomaly injected into the target system might not be the only one that is present. It is possible that the injected anomaly affects other services of the application. In particular, as we focus on energy consumption, a metric that is dependent on the utilisation of hardware resources, we need to be able to define what is considered an anomaly in the collected metrics for this variable.

For this reason, a normal operation of the target systems is first conducted to collect metrics for them during a period of five minutes without injected anomalies. These metrics are used as the baseline for evaluating a certain data entry from the collected metrics and its segment as anomalous or not. When comparing newly retrieved data with the baseline, we consider a data entry anomalous when the value of the data entry has a difference of at least the standard deviation of the metric in the baseline data from the average (mean) value for the same metric that is logged in the baseline dataset, either upward or downward. The product of this process is the ground truth that is later used for the deduction of the metrics for AD.

5.3 Experiment Design

The application of the various treatments of the dependent variable of this experiment, described in subsection 5.2, is conducted using a cross-over design [40]. Every combination of the type of system stressing and the intensity level is applied to each of the target application's services each time. This is done by introducing a spike on the utilisation of just one hardware component for just one of the application services at a time, while ensuring that the traffic directed to the system is at relatively consistent levels between experiment runs and trials. To achieve this, we have created a custom locust[23] script for SockShop that generates predictable traffic to the system, by sending requests to endpoints of the application that we specify. For Zahori, we run a test process that has been developed and made available by the software developers that build and maintain Zahori.

Additionally, the order in which the system stressors (i.e. treatments) are applied are randomised between experiment trials. This is necessary to ensure that no bias exists in the order of execution of the stressors and that any factors that could affect the experiment's

[23]https://locust.io/

results, such as network and system conditions, are minimised and spread out across different trials.

After metrics about hardware utilisation and energy consumption of an application's services have been collected from Prometheus and SmartWatts, the energy metrics are passed as input to the anomaly detection solution utilised for this experiment, namely PyCaret. Then, the results generated by AD determine whether there is a need for the execution of root-cause analysis on the collected metrics. The presence of detected anomalies in the application metrics on energy consumption indicates the need for further analysis of the data using RCA to identify the root of these anomalies. This is done by utilising the RCD tool. A detailed description of the setup and execution of the experiment is given in section 6. Finally, the results of these two phases, AD and RCA, produce the material for analysing the effectiveness in utilising energy consumption metrics to detect energy consumption anomalies and combining energy consumption metrics with performance-oriented metrics to identify the root causes of such anomalies in the target application.

As described in 5.2, the combination of stress type and level that is applied on the target system defines the type of treatments for our experiment. We applied each treatment 20 times on the most actively engaged services of the applications. For SockShop, we targeted four services, while for Zahori, we focused on two services, as described in Section 6. Moreover, we ran the target system 20 times without injecting any anomalies, i.e. without applying stress, to capture the normal operation metrics of each system.

In total, the SockShop experiment involved 500 trials, with 4 services getting injected with anomalies of 3 different types and 2 intensity levels for twenty trials, in addition to 20 non-anomalous trials. Regrading Zahori, the experimentation involved 2 services getting stressed with 3 types of stressing and 2 levels of intensity for each stress. Each trial is executed 20 times to smooth out outliers that may occur in certain trials. Additionally, 20 non-anomalous trials are conducted, resulting to a total of 260 trials for the Zahori experiment. When the experiments for SockShop and Zahori are completed, the DDAD pipeline is executed offline on the datasets collected from each experiment trial, running three AD models on every dataset. For this reason, the final count of AD results is three times the number of experiment trial for each system. Subsequently, the number of times RCD runs on the anomalous data is almost tripled as well, depending on whether an anomaly has been detected in a certain dataset that triggers the execution of RCD, or whether RCD managed to find the root cause.

The progression from the initial exploration and refinement using the research-oriented SockShop system to the subsequent deployment within Zahori represents a crucial step in our research. It allows us to transition from an application-oriented for testing purposes to a realistic, operational setting, facilitating a comprehensive assessment of the efficacy of AD and RCA and the pipeline's capabilities and highlighting the pipeline's potential for real-world applications. This pipeline has been designed to be conducted in a partially automated manner, with certain phases requiring the intervention of a human operator to move forward. The process flow graph of this pipeline is depicted in Figure 5 and described further in Section 6.

6 EXPERIMENT EXECUTION

In this section, the arrangement of the experiment's software and hardware components is delineated. Additionally, the approach to conducting the experiment and acquiring system metrics for the designated applications is presented.

The anomaly detection and root cause analysis operations that are described in this section are executed as part of the DDAD pipeline that is designed and proposed as part of the current study. This way, the functionality and reliability of the pipeline is properly demonstrated. Furthermore, we have conducted a thorough exploration of the different techniques available for AD and RCA. Our investigation has involved analysing a diverse range of tools that utilise algorithms and adopt unique approaches that cover a wide range of concepts.

6.1 Preparation

6.1.1 ***SockShop***. A key requirement for this experiment was to assess the performance of anomaly detection and root-cause analysis tools under stress conditions. To achieve this, SockShop's Docker images were modified to incorporate stress-ng[24], a versatile solution for imposing stress on system resources. This stress testing tool can effectively generate a significant system load by deploying various stressors such as CPU-intensive tasks, memory allocation, disk I/O, and more. The modified Docker images[25] used in this experiment, which introduce stress-ng, were created by Lilly Wu. This modification was made as part of Wu's previous research titled "MicroRCA: Root Cause Localization of Performance Issues in Microservices"[16].

In total, four critical user-facing microservices - catalogues, orders, payment, and shipping - were subjected to stress using modified images out of the 13 microservices. Since the system is interconnected, any changes made to the performance of these services directly impact the other microservices. These four chosen microservices are an accurate representation of the larger system in terms of functionality and communication patterns. The purpose of this experiment was to create real-world scenarios where stress conditions could impact the overall performance of the microservices ecosystem.

Additionally, it was crucial to introduce realistic user traffic to the services. To achieve this, we employed locust[26], a tool designed to simulate user behaviour. In this regard, we simulated interactions from 100 simultaneous users throughout the experiment. We determined that this number was adequate for introducing some load to the services without causing undue stress while also ensuring that they were not left idle. Each simulated user engaged in a specific sequence of actions, including logging in, browsing the product catalogue and picking 1-9 products, viewing the cart, and finally ordering the products.

6.1.2 ***Zahori***. To effectively utilize Zahori for the experiment, some minor adjustments were required. Specifically, Selenoid and Zahori-server were seamlessly integrated with stress-ng. Unlike Sockshop, Zahori is designed to cater to a limited number of users at a time, as its main objective is to automate processes and repetitive

[24]https://wiki.ubuntu.com/Kernel/Reference/stress-ng
[25]https://hub.docker.com/r/lillywu/sock-shop
[26]https://locust.io/

tasks. To achieve this, an Automated Process was initiated via an API call, which involved conducting a search on Wikipedia and gathering logs, screenshots, documents, and videos while simultaneously generating a Fibonacci sequence for four minutes. This process was executed during each normal and anomalous trial and enabled the creation of more realistic scenarios, as Zahori services remain mostly idle when a process is not running.

6.1.3 *Anomaly Detection*.

The software solution PyCaret is utilised for anomaly detection. It is an open-source, low-code machine-learning library in Python that automates the process of anomaly detection. PyCaret offers different modules for various specialised applications, including anomaly detection. Three algorithms are available in pycaret to perform anomaly detection on system metrics, namely, Local Outlier Factor (LOF)[27], k-Nearest Neighbors (KNN)[28], and Isolation Forest[29]. For our research, a separate model is trained for each algorithm using the data from normal, non-anomalous runs of the experiment.

When analysing the provided dataset, PyCaret only provides one indication regarding the presence of an anomaly for each entry in the dataset. There are no hints about the anomaly status of individual columns in the dataset, which, in our case, refer to the combination of each application service and the metric captured for them. For this reason, a model for each algorithm is trained on each column of the normal dataset. The trained models are stored locally and later utilised to perform AD on the anomalous datasets captured during the experiment execution.

Each data entry in the provided dataset is analysed by the PyCaret model, resulting in two newly created columns that PyCaret uses to indicate the anomalous status of that data entry, namely "Anomaly" and "Anomaly Score". The former column is an indication of whether a data entry is considered to be anomalous (value of 1) or not (value of 0). The latter is the score assigned to a data entry, based on which the decision is made as to whether it is anomalous. This is a continuous value that can take a value from 0 up to 1, with higher values indicating a stronger suggestion that a specific data entry is considered anomalous. The anomaly status assigned by PyCaret is compared to the anomaly status signified in the ground truth to analyse the performance of AD on energy consumption data.

6.1.4 *Root-Cause Analysis*.

For RCA, the RCD tool is employed, designed and developed by Ikram et al. [9]. This localised causal discovery algorithm identifies the root cause of anomalies detected in system metrics. The algorithm pinpoints the root causes without learning the causal structure of the complete graph. RCD is run on a certain dataset of captured system metrics only when the execution of AD results in the discovery of anomalies in the dataset. Then, the same dataset is used as input for RCD which analyses the data and generates a ranked list of the most probable causes of the detected anomalies. During the assessment of RCA's ability to detect the root causes of energy anomalies, this list is used to derive the performance metrics for RCA, as described in 5.

[27]https://www.sciencedirect.com/topics/computer-science/local-outlier-factor
[28]https://www.sciencedirect.com/topics/biochemistry-genetics-and-molecular-biology/k-nearest-neighbor
[29]https://www.analyticsvidhya.com/blog/2021/07/anomaly-detection-using-isolation-forest-a-complete-guide/

6.1.5 *Energy Consumption*.

As this study aims to explore the ability of anomaly detection and root-cause analysis processes to handle energy consumption data and detect energy anomalies and the root causes of these anomalies, this information must be available at the beginning of the experimentation. However, the metrics that SmartWatts produces are in the form of power consumption, measured in watts. Thus, it is necessary to translate the power metrics to energy metrics measured in joules, or watts per hour. This is done by multiplying the average power consumption of the system over 5 seconds with the number of seconds over which the average was computed (i.e. 5 seconds) to get the average energy consumption for periods of that length for the duration of each experiment trial. This action was conducted before any other data pre-processing and analysis was performed. All following operations were performed using the data coming out of this phase. The original data collected from the target systems, SockShop and Zahori, are not discarded and can be used as a reference for the initially collected power metrics.

6.1.6 *Ground Truth*.

To create the ground truth used as the baseline for deriving the correctness, completeness, and accuracy indicators for AD and RCA, both the data collected during the normal trials of the experiment and the trials with injected anomalies are considered. Specifically, we calculate the average value for each collected metric and service combination stored in the normal dataset. Then, for each metric and service combination in the datasets corresponding to the trials with injected anomalies, we compare each entry with the corresponding mean value from the normal data. As described in section 5, an entry is marked as anomalous in the ground truth when its value differs from the mean normal value by at least the standard deviation of the metric being evaluated in the baseline data, regardless of direction.

6.2 Setup

The experimentation that was conducted as part of the current study was executed on an external server hosted by Green Labs [30] at the Vrije University of Amsterdam. The server is equipped with an Intel(R) Xeon(R) Silver 4208 CPU @ 2.10GHz, 376GB of RAM and operated on Ubuntu 22.04.2 LTS operating system. Furthermore, the pipeline was developed using Python version 3.9.17, and the subjects utilised Docker version 24.0.2.

6.3 Execution and Measurement

The hardware utilisation for each service of each target application during the experiment runs is captured using the Prometheus monitoring solution. The services deployed as Docker containers are measured across different metrics, namely CPU usage in seconds, memory usage in bytes and filesystem reads in bytes. Along with these metrics, the power consumption of each service during experiment trials is also captured. After completing all the experiment runs, the power consumption values are converted to energy consumption values to accommodate the needs of this study in answering the research questions defined in 4.

For each trial, we either select one of the application services to stress or no service is injected with an anomaly for the duration of

[30]https://s2group.cs.vu.nl/pages/greenlab/

the trial. This is done at random. The data gathered during trials that do not include any injected anomalies are used to construct the ground truth to analyse the results that AD and RCA produce. When a service is selected to be stressed, three available stressors can be executed, one for each type of hardware utilisation metric captured for the application services. The stress types target each application service's CPU, memory, or file system. Only one stress type and one service is stressed at a time.

To enhance the accuracy of each new trial, we have implemented a system that allows for a 4-minute break between experiment trials. This pause eliminates any previous stress-inducing operations before a new trial commences. Additionally, constantly stressing the hardware components can cause the server hosting the application to produce more heat, leading to increased power usage over time. However, allowing the system to rest between runs gives it enough time to cool down and return to its usual temperature and power consumption levels before starting a new trial. This ensures that the data we collect remains reliable and accurate. The duration of this break was found to be the shortest amount of time that leads to no residuals from previous experiment trials, after exploring various duration options. This way the experiment data remain unaffected in this aspect, while keeping the experiment execution time short.

6.4 Pipeline

Here, we provide an overview of the various steps involved in the AD and RCA pipeline created for and utilised in this study, along with the expected inputs and outputs that every step of the pipeline should have. The pipeline is split into 4 phases, namely *Preparation*, *Context Elicitation*, *Processing*, and *Assessment*. The process flow graph of the pipeline is shown in Figure 5.

The words technique and tool are used interchangeably. They are used to refer to the solution that is employed to process and analyse the data the system produces and that uses such data to perform anomaly detection or root cause analysis (or both) on it. As such, the pipeline described in this subsection can incorporate solutions for anomaly detection, root cause analysis or both.

6.4.1 ***Preparation.***

In this phase, the target system is analysed and the appropriate solution is selected and prepared accordingly.

System Classification The user identifies aspects of the system that are essential for the selection of the appropriate technique. Such characteristics include the level of monitoring artefact the system is able to produce (logs, traces, metrics) when executed and the type of the system configuration, which, in this study, consists of Docker.

Technique Selection Once the system has been classified, it is necessary to analyse the available techniques and choose the one that most appropriately matches the identified characteristics of the system and best suits the objectives of the system administrators. This depends on the granularity of the produced artefact (application- or service-level) and the type of the discovered anomaly (functional or performance) that the administrators require, but also the classification and the characteristics of the system. For this study, the focus is set on energy-related anomalies.

Technique Setup Before the selected tool can be used it is necessary to prepare the tool based on the author's instructions. This setup is highly dependent on the chosen technique but may consist of installing different dependencies, setting up log traces, etc.

6.4.2 ***Context Elicitation.***

In this phase, the necessary data to tune the behaviour of the selected AD or RCA solution to the current setup are gathered and provided as input to the tool.

Acquisition of Required Input The choice of tool in a previous step will determine the specific data requirements based on the input parameters the tool requires, such as runtime and previous system logs, previous system traces, (Kubernetes) application deployment, test cases, workload generators, previously monitored system KPIs, failure injectors, application specifications, anomaly patterns and SLOs. This data is used by the tool to obtain the ground truth, which provides an accurate depiction of the microservice system's state and behaviour during normal or abnormal operations, and is crucial for identifying anomalies and their root causes. This input data and the ground truth are generated and stored as a system artefact. In particular, the system artefact is a set of data consisting of the ground truth for the behaviour of the system under normal or abnormal conditions (as needed by the employed solution), and any additional input parameters that the tool requires.

Technique Tuning Upon acquiring the system artefact it is necessary to tune the tool to create a ground truth in the form of, for example, dependency or causality graphs. This information is later used, in step *Analysis of Produced Data*, by the employed solution to determine whether a system is behaving anomalously or not.

6.4.3 ***Processing.***

After the solution has been tuned according to the needs of the underlying system and of the system operators, metrics regarding the operation of the system are gathered while the system accepts traffic. Such metrics could take the form of hardware utilisation data, energy consumption data, or any other type of metric that is required for the analysis of the system for anomalies and their root causes.

System Execution The operation and running of the selected software system. This means that the software abandons its stale state and starts processing workload going into execution state. At the same time, the appropriate monitoring tools are deployed and configured to continuously collect the required data, according to the technique that has been selected.

System Monitoring in Real Time While the system is in the execution state, it is monitored continuously and several metrics (e.g. CPU usage, power consumption, etc.) are recorded and stored into a monitoring artefact to be passed for further analysis. Other than metrics, the monitoring artefact consists of logs or traces the system produces while it is in execution state. Monitoring can also be performed in cycles, with each cycle lasting a predetermined duration, in case of offline analysis for anomaly detection or root cause analysis. For example, one such cycle could last 60 minutes, after which the monitoring artefact is taken offline for further processing and analysis. The monitoring of the target systems in this study is conducted using the tools Prometheus, Grafana, and SmartWatts.

6.4.4 ***Assessment.***

In this last phase of the pipeline, the collected system metrics are analysed by the chosen AD or RCA tool, after which they generate certain results. If necessary, the operators of the process intervene to review and evaluate the results.

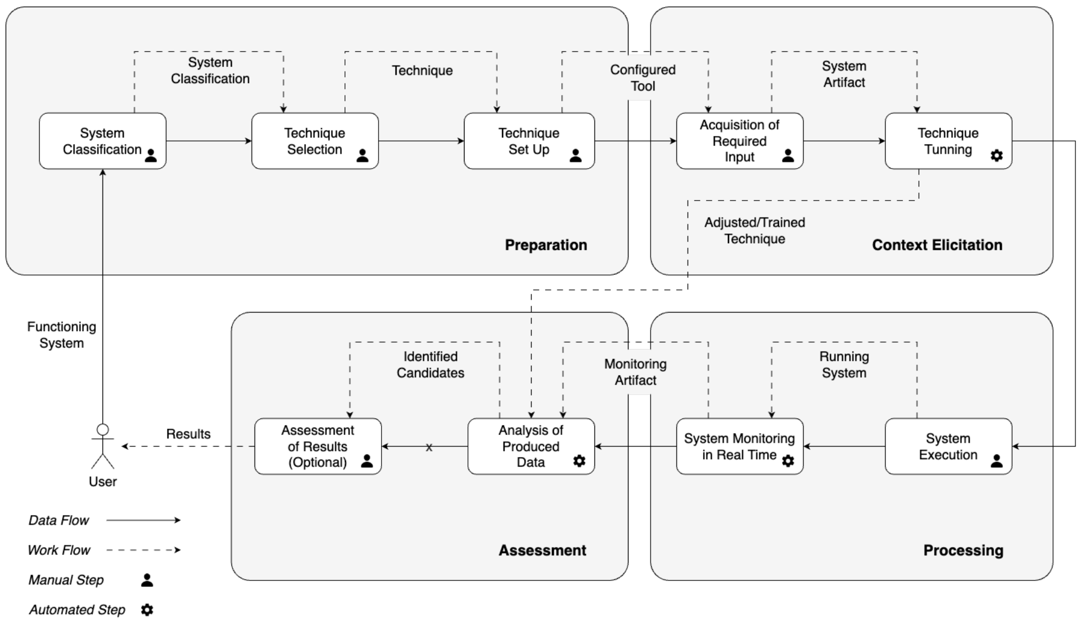


Figure 5: Data-Driven Anomaly Diagnosis: Semi-automated Pipeline for Anomaly Detection and Root-Cause Analysis

Analysis of Produced Data The data collected from the system monitoring and the system artefacts (coming from the Input Analysis step) are combined and provided to the selected technique. The solution then performs an analysis and provides a verdict which can be either a set of candidates of potential anomalies or root causes for an anomaly. This process can take place either online - while the system is running - or offline - when data from a previous system execution cycle is being processed.
Assessment of Results When a tool produces multiple candidates of anomalies, for anomaly detection, or root causes, for root cause analysis, it can be challenging to determine which one is the most likely anomaly or cause. This is where human intervention comes into play. The system administrators must carefully review each candidate and evaluate its relevance to the system. Ultimately, the goal is to identify the anomaly that happened or the root cause of an observed anomaly and take appropriate corrective action to address it.

6.5 Anomaly Detection Models

To conduct anomaly detection in our research we employ 3 algorithms that are available within the utilised solution, namely PyCaret. Here, we provide the basics of each of these algorithms.

6.5.1 ***Local Outlier Factor (LOF).*** LOF is a model used for anomaly detection. It works by comparing the density of data points in a given data set. It identifies outliers by measuring how isolated a data point is compared to its neighbors – if a point has significantly lower density around it, it's likely an outlier.

6.5.2 ***k-Nearest Neighbors (KNN).*** This is a simple and versatile algorithm used for classification and regression tasks. For classification, it assigns a label to a data point based on the majority class of its nearest neighbors. For regression, it predicts a value by averaging the values of its closest neighbors. The 'k' parameter represents the number of neighbors considered.

6.5.3 ***Isolation Forest (IForest).*** Isolation Forest is an anomaly detection algorithm that separates anomalies (outliers) from normal data points by building a random forest of decision trees. Anomalies are easier to isolate and require fewer splits to be separated from the majority of data points. By measuring the average path length of a data point in the forest, anomalies can be identified – shorter paths indicate potential anomalies.

7 RESULTS

In this section, we present the outcomes of the analysis conducted on the correctness, completeness, and accuracy of the anomaly detection and root-cause analysis processes. These results are derived from the performance of the AD and RCA processes on the metrics collected during all the trials of the experiment. For AD, the metrics refer to the adjusted versions of them, regardless if this is explicitly stated or not. Moreover, these performance metrics serve as the basis for addressing the research questions defined in Section 4.

7.1 Data Exploration

The first step in addressing the research questions involves an inspection of the collected data to shed light on their quality. This analysis helps give a better insight into the underlying characteristics of the dataset and informs the subsequent stages of the research process.

In Figures 6 and 7, we observe the resource utilisation during the stressing of services for both SockShop and Zahori. In our case, the duration of the stress test was 300 seconds (5 minutes). The first noticeable observation is an initial spike in CPU and memory utilisation within the first few seconds of the test. This spike is present because the invocation time of the stressor, which in this case is the anomaly being injected, coincides with the moment when we begin collecting data from the target system.

Furthermore, it is observed that there is no tangible surge in the file system utilisation metrics when introducing an anomaly into the file system. Interestingly, the file system utilisation for the payment and catalogue services in SockShop appears to remain constant. Nevertheless, the reason for this could be attributed to varying base levels for file system reads that take place during normal system operations. Consequently, when the stressor is introduced, the file system utilisation fluctuates in line with the typical file system usage.

When examining the data presented in Figures 8 to 10, a clear trend emerges between the resource utilisation and energy consumption patterns for each of the tested services within the SockShop application. Notably, the figures provide a visual representation of the relationship between these two critical aspects. An intriguing observation is the noticeable surge in energy consumption concurrent with the occurrence of anomalies. This suggests that anomalies may trigger higher energy usage, but further analysis reveals a more complex pattern. While some instances of high energy usage coincide with anomalies, there are also instances where resource utilisation remains stable despite fluctuations in energy consumption. This suggests that there may be additional factors at play beyond just resource utilisation. Similar patterns are observed when Zahori is considered, as shown in Figures 11, 12, and 13. Further studies could be conducted to shed light on this situation and these patterns.

7.1.1 ***Anomaly Detection***

We utilised three algorithms, namely Local Outlier Factor (LOF), k-Nearest Neighbors (KNN), and Isolation Forest (IF), to classify our data entries as anomalous or not. The adjusted precision, recall, and F1-score for each of these algorithms across all the collected data for the SockShop experiment are listed in Table 1. The same metrics for the Zahori experiment are presented in Table 2. In this case, we count and sum up the TP and FP in the datasets collected from each experiment trial, for each application separately, and calculate the performance metrics using the final count. Table 3 presents descriptive statistics about the same metrics, this time taking into account each dataset retrieved during the data collection phase of the research separately, instead of examining all the data together as a whole. These results give us insight into how well the AD algorithms can detect anomalies in energy consumption data in both research-oriented and production-level applications. Moreover, since our study's primary focus is to evaluate the overall performance of the anomaly detection process, Tables 1, 2, and 3 also provide an overview of these metrics across all the algorithms used, showing the average performance that AD had on each examined system.

Looking at Table 1 we observe that in the case of SockShop the Local Outlier Factor model exhibited superior performance in terms of precision in detecting anomalies within energy consumption metrics, compared to the other two employed models, namely IF and KNN. The key factor contributing to the Local Outlier Factor's superiority was its ability to maintain a particularly low number of false positives (FP) compared to true positives when identifying anomalous sections. Specifically, the LOF model made only 182

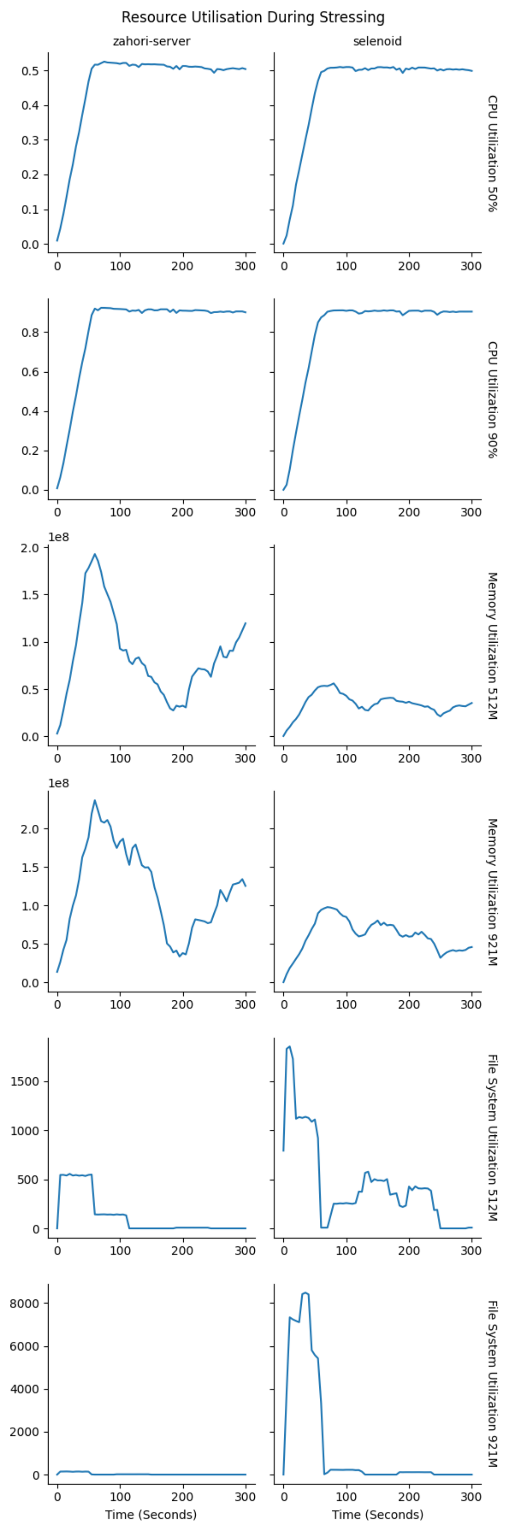


Figure 6: Zahori: Resource Utilisation of Each Service During Hardware Stress Testing

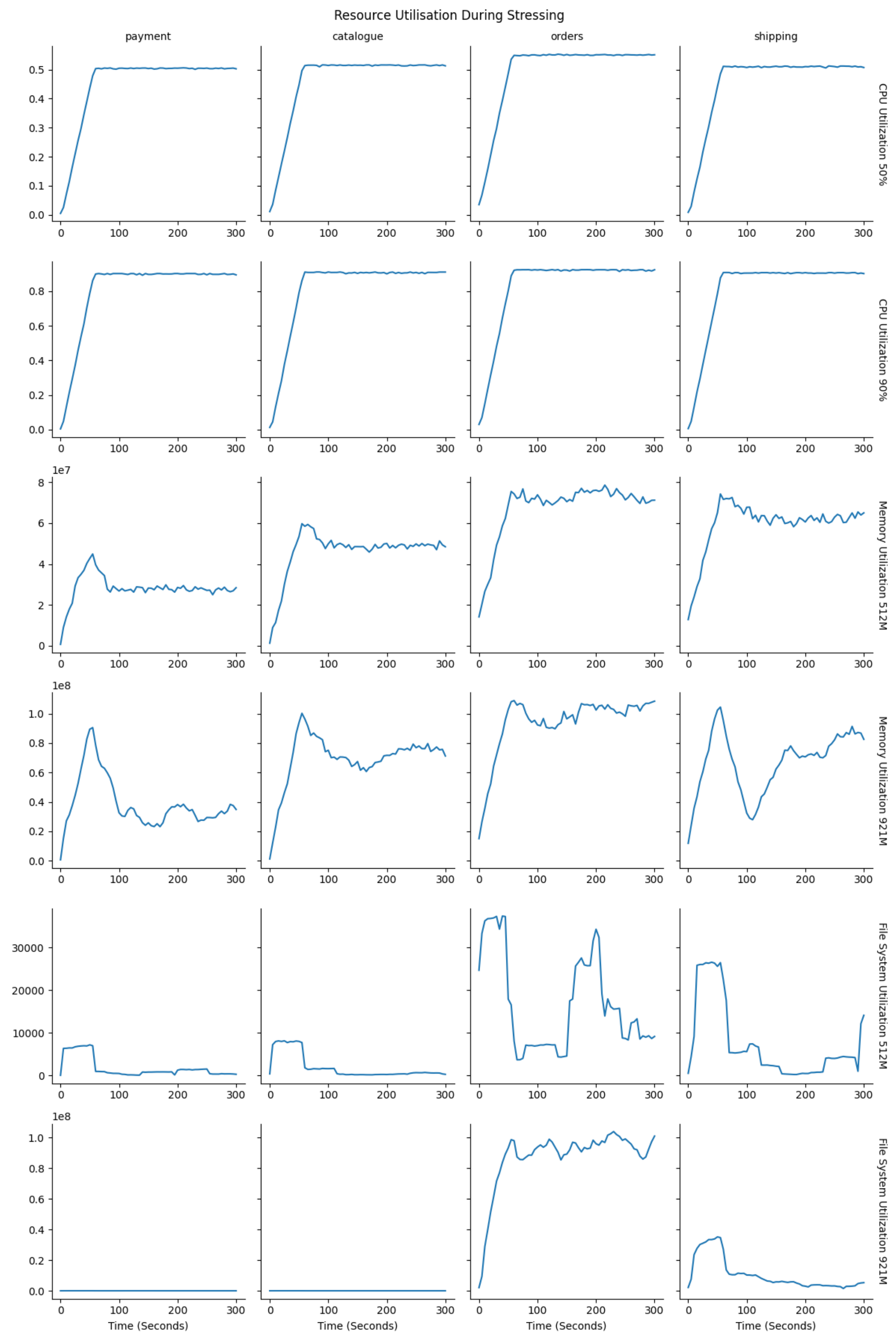


Figure 7: SockShop: Resource Utilisation of Each Service During Hardware Stress Testing

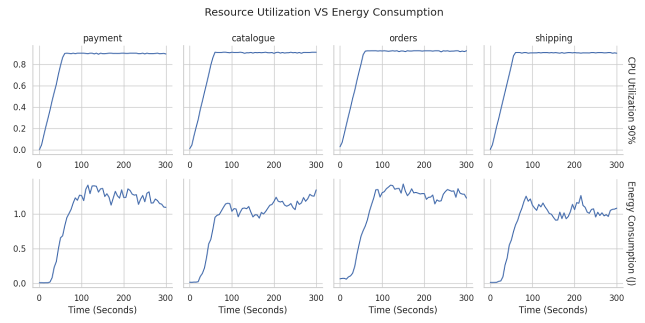


Figure 8: SockShop: Comparison of Energy Consumption and CPU Utilisation

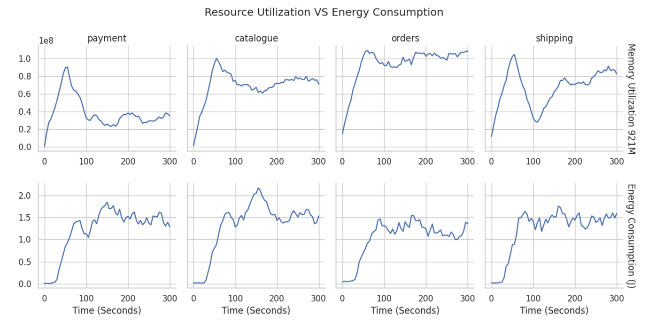


Figure 9: SockShop: Comparison of Energy Consumption and Memory Utilisation

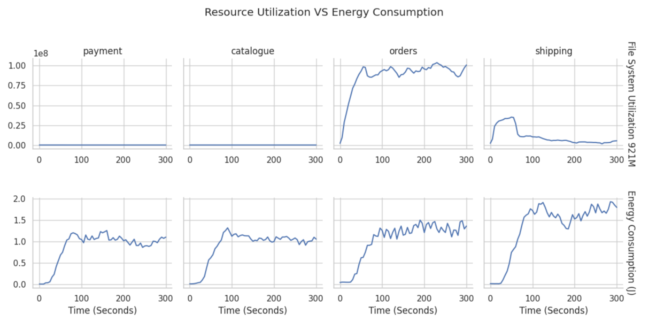


Figure 10: SockShop: Comparison of Energy Consumption and File System Utilisation

Table 1: Anomaly Detection Performance Metrics per Model and Overall - SockShop

Model	Precision	Recall	F1-Score
Isolation Forest	0.934	0.208	0.341
k-Nearest Neighbours	0.954	0.338	0.499
Local Outlier Factor	0.97	0.36	0.525
Overall	0.956	0.302	0.459

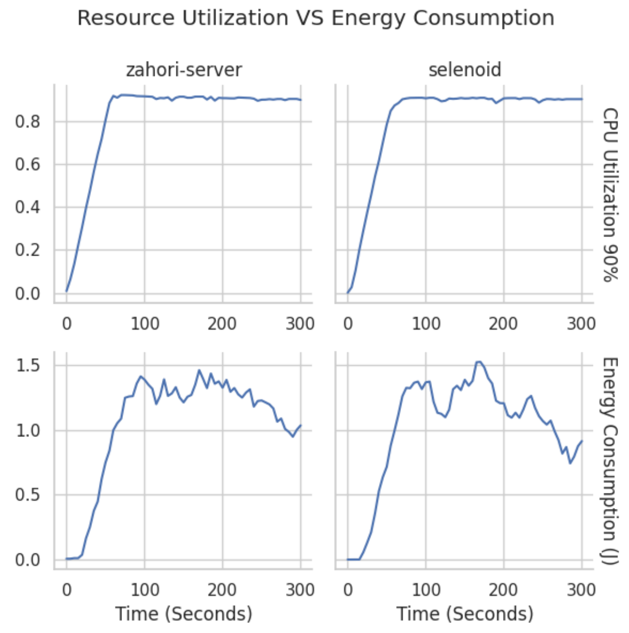


Figure 11: Zahori: Comparison of Energy Consumption and File System Utilisation

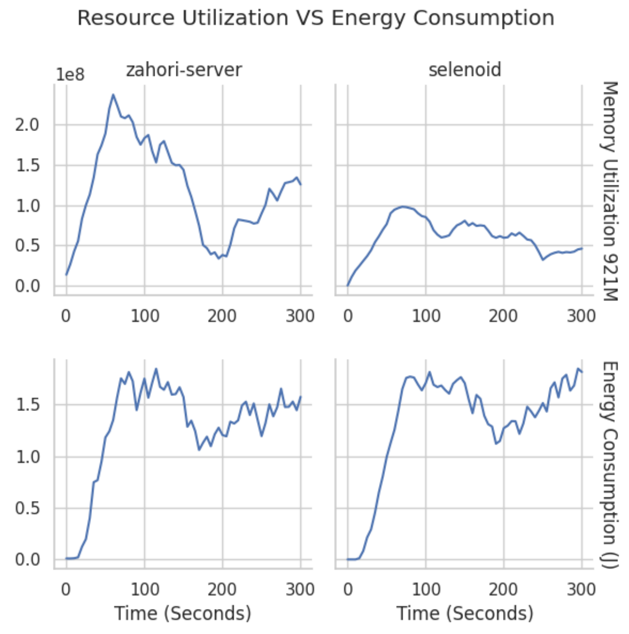


Figure 12: Zahori: Comparison of Energy Consumption and File System Utilisation

incorrect classifications as false positives for SockShop. On the contrary, the IF model reported 242 false positives, and the KNN model falsely detected anomalies on 271 occasions.

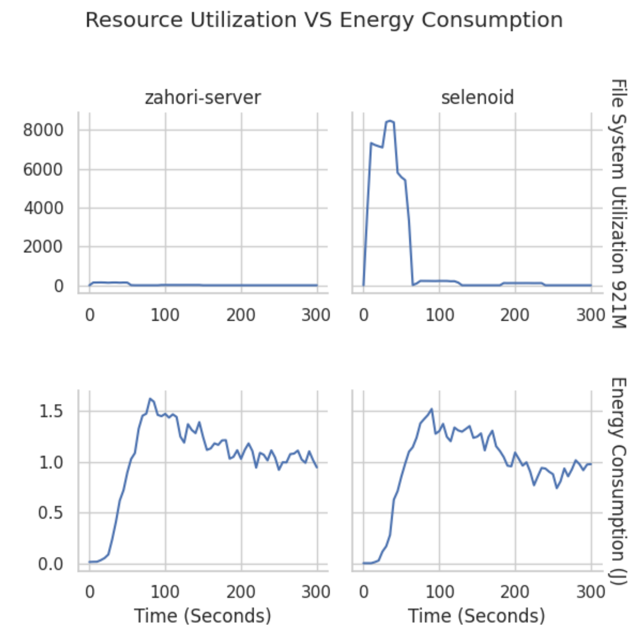


Figure 13: Zahori: Comparison of Energy Consumption and File System Utilisation

Table 2: Anomaly Detection Performance Metrics per Model and Overall - Zahori

Model	Precision	Recall	F1-Score
Isolation Forest	1	0.414	0.585
k-Nearest Neighbours	1	0.472	0.641
Local Outlier Factor	1	0.639	0.78
Overall	1	0.508	0.674

Furthermore, we analysed the number of true positives (TP) for each model. In the case of SockShop, the LOF model demonstrated approximately 32.5 times more true positives than false positives. In comparison, the IF model had a ratio of a little over 14 times, while the KNN model's ratio was approximately 20 times. These findings are reflected in the precision values for each model, providing additional support to the conclusion that the Local Outlier Factor model outperforms both IF and KNN in terms of precision.

Similarly, when evaluating recall and F1-score, the Local Outlier Factor model displayed the highest values, with a recall of 0.36 and an F1-score of 0.525. In comparison, the adjusted recall and F1-score values for Isolation Forest are 0.208 and 0.341. The KNN model yielded a recall of 0.338 and an F1-score of 0.499, having the second best performance out of the 3 considered models.

It is evident that all the considered AD models exhibited relatively low performance in completeness and accuracy. This can be attributed to the significantly high number of false negatives (FN) across all models. The number of false negatives is approximately 1.5 times greater than the number of true positives for both LOF

Table 3: Summary of Descriptive Statistics for Anomaly Detection

Metric	**Precision**					
Subject	**SockShop**			**Zahori**		
Algorithm	**iforest**	**knn**	**lof**	**iforest**	**knn**	**lof**
Min	0.0	0.0	0.0	0.0	0.0	0.0
Max	1.0	1.0	1.0	1.0	1.0	1.0
Mean	0.44	0.615	0.695	0.517	0.59	0.773
SD	0.493	0.483	0.458	0.5	0.492	0.419
CV	1.12	0.785	0.659	0.967	0.834	0.542
Metric	**Recall**					
Min	0.0	0.0	0.0	0.0	0.0	0.0
Max	1.0	1.0	1.0	1.0	1.0	1.0
Mean	0.329	0.465	0.506	0.466	0.532	0.701
SD	0.425	0.437	0.427	0.476	0.473	0.42
CV	1.292	0.94	0.844	1.021	0.889	0.599
Metric	**F1-Score**					
Min	0.0	0.0	0.0	0.0	0.0	0.0
Max	1.0	1.0	1.0	1.0	1.0	1.0
Mean	0.356	0.502	0.552	0.481	0.55	0.724
SD	0.431	0.437	0.42	0.478	0.473	0.412
CV	1.211	0.871	0.761	0.994	0.86	0.569

and KNN models, while it is nearly 4 times larger in the case of the IF model.

When examining the outcomes for Zahori, the situation does not change a lot. The precision values for all the models employed for anomaly detection are interestingly high, with a value of 1 in each case. The high precision observed in the AD results for Zahori, as shown in Table 2 can be rationalised by the non-existent amount of FP. All the AD models managed to avoid incorrectly marking non-anomalous segments as anomalous, an astonishing performance that inspires a high level of confidence on the results that AD generates.

In terms of recall, the ranking of the algorithms for Zahori is the same to the findings for SockShop. LOF exhibits the highest recall value of 63.9%. Isolation Forest continues to exhibit the weakest performance, registering a recall score of 41.4%. KNN now stands at 47.2% for adjusted recall in the case of Zahori. Subsequently, the overall recall for AD in Zahori is greater compared to that of SockShop, as it now stands more than 20% higher. Concerning accuracy and adjusted F1-score, the overall effectiveness of AD is, again, greater to that observed for SockShop, with the overall adjusted accuracy standing at 67.4% for Zahori and hovering around 46% in the case of SockShop.

Additionally, as shown in Table 3, when the individual datasets are considered, the LOF model demonstrates the best mean performance across all metrics, with 69.5% and 77.3% mean adjusted precision, 50.6% and 70.1% mean adjusted recall, and 55.2% and 72.4% mean adjusted F1-score for SockShop and Zahori each time respectively. This model has also the least dispersion across all performance metrics, as shown by the standard deviation (SD) and the coefficient of variation (CV). That said, the dispersion of the data is relatively to extremely high for the results of all of the AD models employed. This is shown by a CV value that is consistently greater

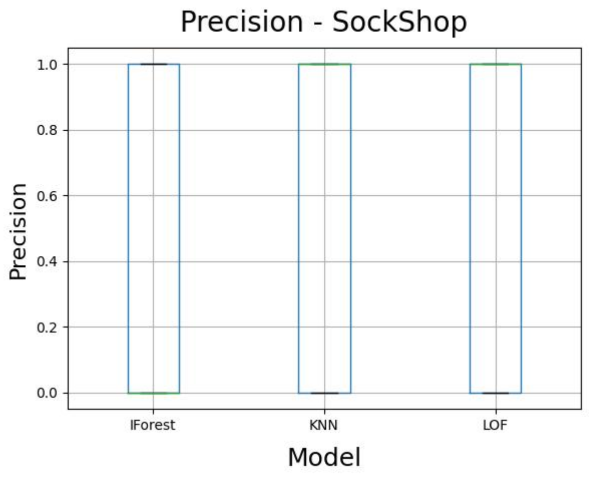


Figure 14: Distribution of precision for every anomaly detection model - SockShop

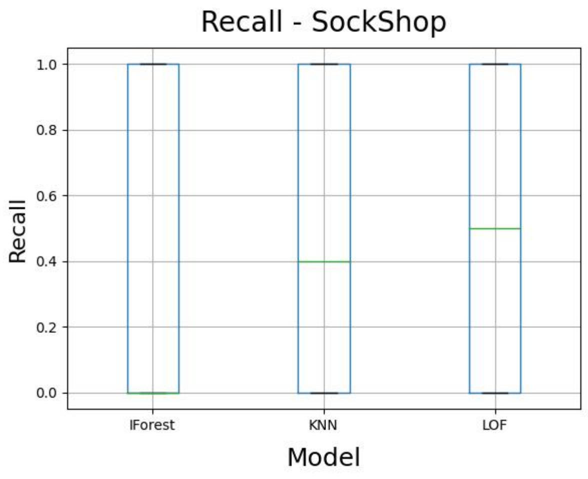


Figure 15: Distribution of recall for every anomaly detection model - SockShop

than 50% for all models and systems and a SD that is hovering around the 45% mark.

The boxplots in Figures 14-19 illustrate the distribution of all the AD models used in our research across all of the datasets collected in the experimentation for SockShop and Zahori. These visual representations complement the descriptive statistics provided in Tables 1, 2, and 3. Looking at these boxplots we confirm the inferior performance AD had when applied on metrics gathered from the experimentation on Zahori. Furthermore, we observe that the Isolation Forest model has consistently poor performance, weaker than the other AD models employed, with its high value for F1-score being an outlier and the median for this metric hovering around 0%. Moreover, the performance of LOF is kept always at high levels, with the mean being consistently lower than the media, across all metrics. The low values for LOF, reported in Table 3 for Zahori are only outliers and do not comprise a large portion of the computed performance metric data. This indicates a negative skewness in the data distribution that correspond to the performance metrics for each individual dataset, where there are some lower values that are impacting the mean while the majority of the data is clustered around higher values. This suggests that LOF is capable of detecting anomalies in energy consumption data effectively

7.1.2 *Root Cause Analysis.*

When an anomaly is detected in the energy consumption metrics gathered from an experiment execution, the process of RCA is initialised. This includes all the times AD determines an anomaly is present, regardless if the detection was a TP or FP. As discussed in Section 6, the tool used for this operation is RCD. The performance of this phase is evaluated using the metric $P@k$ (precision at k) for levels 1, 2, and 3. Additionally, the average precision across all considered precision levels, up to level 3 ($AP@3$), is taken into consideration to assess RCD's, and more generally RCA's, effectiveness in detecting root causes of energy consumption anomalies.

As shown in Table 4, the precision levels of RCD for the SockShop experiment are relatively low across all precision levels, never exceeding the 0.5 mark. As expected, the highest value is observed for level 3 ($P@3$), with the precision having a value of 0.452. This means that in more than 50% of the cases, RCD fails to identify the true root cause of the injected anomalies in the top 3 predictions that it makes. This value suggests that RCD, and, in this case, root cause analysis in general, is not very reliable in identifying true root causes and employing this tool would result in a large number of irrelevant predictions. This conclusion becomes even more relevant when a lower number of top RCA predictions are considered. The precision at level 1 ($P@1$) yields a value of 0.412, with 593 correct top-1 predictions out of the 1440 datasets that were examined, and $P@2$ has a value of 0.443. It is obvious and expected, that at higher levels of precision, the value of the metric increases. This increase is minor, though, and does not suggest a strong ability to detect

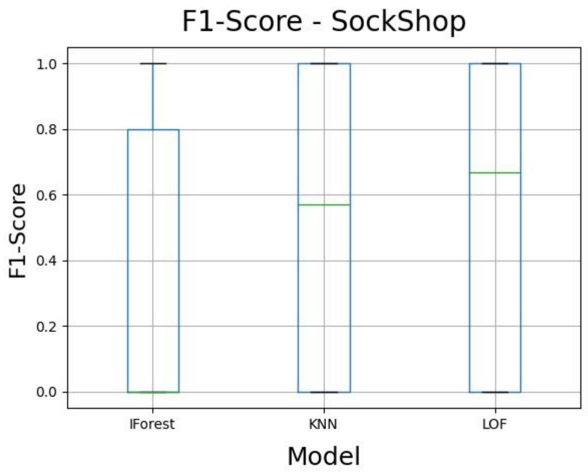


Figure 16: Distribution of F1-score for every anomaly detection model - SockShop

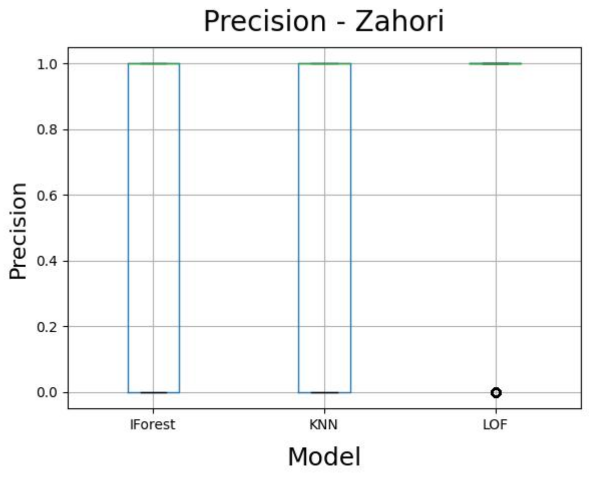


Figure 17: Distribution of precision for every anomaly detection model - Zahori

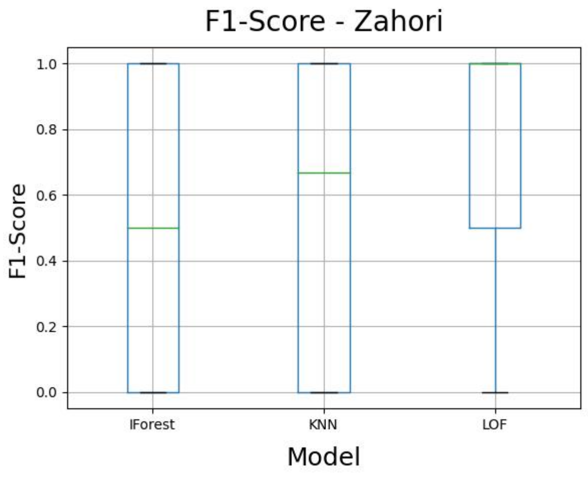


Figure 19: Distribution of F1-score for every anomaly detection model - Zahori

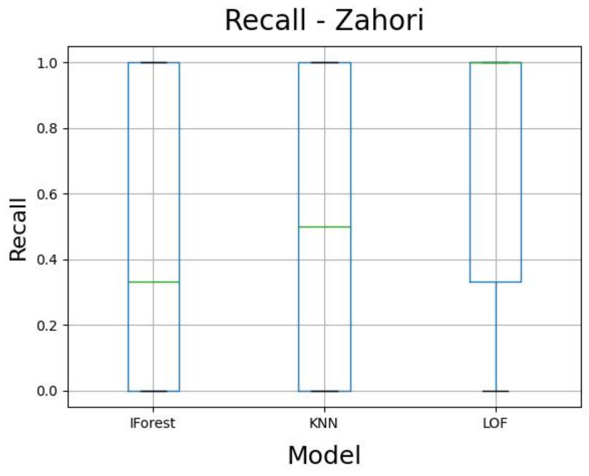


Figure 18: Distribution of recall for every anomaly detection model - Zahori

true root causes with confidence as the number of considered predictions increases. Overall, the average precision at level 3 (*AP*@3) exhibits a value of 0.436. This imprints the average performance that RCA exhibits in identifying root causes correctly.

Regarding the results concerning the experiment in which Zahori is the target application, the RCA results exhibit lower values across all levels of precision. Now, more than 60% of the real root causes of the anomalies detected in the system fail to be pinpointed within the top 3 predictions generated by RCD. Specifically, P@3 stands at 0.384, with only 279 correct predictions out of the total 726 datasets that were examined. Predictably, for lower levels of precision, the associated values decrease. At the second precision level (P@2), the value is recorded at 0.375 with 272 correct identifications. Similarly, at the first precision level (P@1) the value is 0.358 reflecting 260 correct root cause identifications within the overall pool of 726 instances. As a result, the average precision (AP@3) stands at a low value of 0.372.

That said, achieving a high P@1 value may be challenging in many real-world scenarios, especially in complex systems or situations with multiple contributing factors to a problem. There the P@1 that RCD exhibits can still be considered reasonably good, indicating that the analysis is successfully identifying relevant root causes in a significant number of cases. Furthermore, Figures 20 and 21 show the performance that RCA had for all combinations of anomaly injected and anomaly (or stress) intensity across all system services that were injected with anomalies, for SockShop and Zahori, respectively. Specifically, these bar plots show the number of times RCD included the correct root cause in the top k, for all cases of k we considered, namely k values equal to 1, 2, and 3. In this way, we explore what types of system anomalies can be appropriately identified and in which cases effectiveness is weak.

Looking at these graphs, it is prominent that a large portion of the root causes that correspond to CPU anomalies are detected correctly, regardless of the intensity of the anomaly. On the contrary, RCA is practically unable to find root causes that are associated with injected anomalies in the file system of the target service. In this case, regardless of the intensity level, RCD was unsuccessful in identifying any of the correct root causes. Finally, the root causes that are associated with memory anomalies tend to be correctly detected at a considerable rate, however, far less compared to the CPU-associated root causes. These findings are very similar for both examined applications, SockShop and Zahori. Moreover, we observe that the intensity level does not make a big difference in the number of correct root causes that RCA detects, rather it is the type of resource that is injected with an anomaly, and that constitutes the root cause, that is the differentiating factor in terms of the RCA performance. RCA tends to perform very well for root causes associated with CPU anomalies, less so for memory anomalies, and poorly for file system-related anomalies.

Table 4: Root Cause Analysis Precision per Level and Overall - SockShop

Metric	Correct Detections	Value
P@1	593	0.412
P@2	638	0.443
P@3	651	0.452
AP@3	-	0.436

Table 5: Root Cause Analysis Precision per Level and Overall - Zahori

Metric	Correct Detections	Value
P@1	260	0.358
P@2	272	0.375
P@3	279	0.384
AP@3	-	0.372

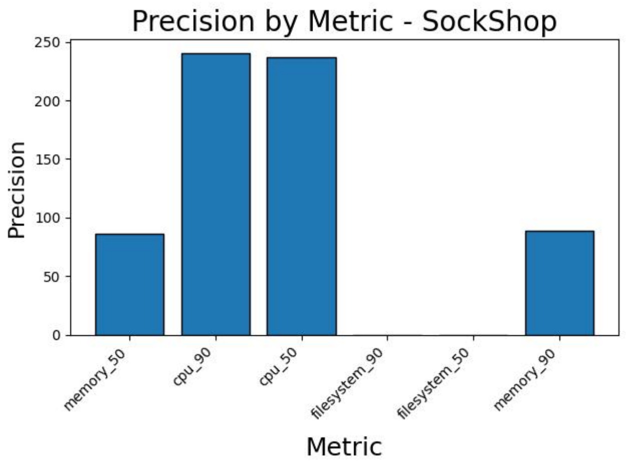


Figure 20: Quantity of correctly detected root causes by RCA - SockShop

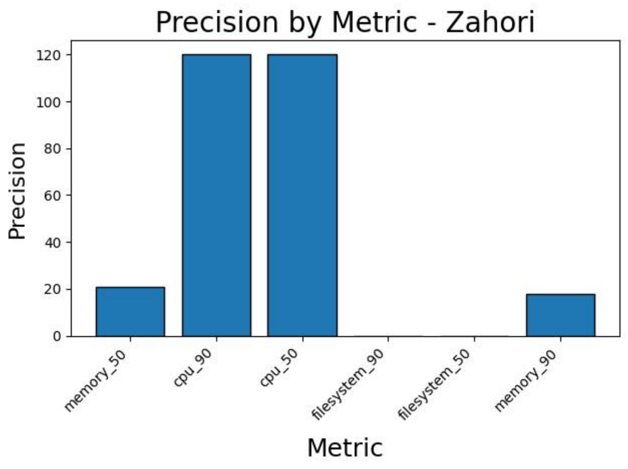


Figure 21: Quantity of correctly detected root causes by RCA - Zahori

8 DISCUSSION

In this section, we elaborate on our research questions using the results presented in Section 7. The implications for all interested parties are outlined, too.

RQ1: ***What is the effectiveness of anomaly detection techniques in detecting energy consumption anomalies in microservice-based applications?***

Based on our empirical results, the performance of the AD process in detecting anomalies in energy consumption data in terms of precision is observed to be high, with a value of around 95%. This indicates that AD has a low rate of false positives. It also suggests that the AD process is effective in identifying anomalies in energy consumption data with minimal false alarms. In other words, when anomaly detection flags an instance of energy metrics as an anomaly, it is more likely to be a true anomaly rather than a false alarm. Additionally, subsequent calls to the RCA process and for human intervention are unlikely to be triggered when they are not needed, refraining from binding valuable resources to perform RCA or time for human operators to investigate the system.

Furthermore, the recall and F1-score values for all the considered models are relatively low, thus the overall AD values for these metrics take relatively low values as well, with 0.302 (SockShop) and 0.508 (Zahori) for recall and 0.459 (SockShop) and 0.674 (Zahori) for F1-score. This indicates that when anomaly detection operates on energy consumption data it is conservative in its approach, leading to a high level of confidence when it flags an instance as an anomaly (because of high precision levels), but it may lack the ability to detect a broader range of anomalies present in the data. As a result, the use of AD on energy data needs to be accompanied by other techniques that can detect energy issues to boost recall and accuracy by reducing the number of false negatives.

Additionally, we observe that the coefficient of variation is remarkably high for all three metrics and all three AD models employed. This is the case for both target systems, signifying significant variability, where the data points are widely dispersed from the mean. This can suggest a high level of uncertainty or inconsistency in the results that AD generates across individual datasets.

Overall, we suggest that practitioners, including system operators, make use of anomaly detection tools to discover when the energy consumption of a system deviates from its normal situation and patterns. The results that AD generates are reliable and accurate, as shown by the significantly high adjusted precision levels we observed during analysis in Section 7. Nevertheless, anomaly detection techniques might not be enough to detect a substantial portion of all the energy consumption anomalies a system might experience. The high number of false negatives, leading to low levels of recall and F1-score, means that many cases of anomalous segments are overlooked by AD. For this reason, we encourage practitioners to complement the use of AD tools with other solutions capable of detecting anomalies and deviations in collected energy consumption metrics.

The previously discussed weaknesses that AD shows make the need for further research and experimentation regarding AD solutions that are capable of detecting anomalies in energy consumption data prominent. The findings of this study can be used as a first indication of the current status of the AD process in regards to energy metrics, on which future studies can be based and expand from.

RQ2: ***What is the effectiveness of root cause analysis techniques in identifying the root causes of energy consumption anomalies in microservice-based applications?***

Regarding root cause analysis, the poor results that RCD exhibits, presented in Section 7, indicate a weak RCA solution in general or a poor performance when incorporating energy consumption data to identify the sources of a system anomaly. According to Ikram et al. [9], RCD outperforms other popular root cause analysis solutions. In their evaluation of RCD, Ikram et al. utilised solely hardware utilisation metrics. As a result, we suggest that RCD, and subsequently RCA, cannot incorporate energy consumption data very efficiently. This is obvious when we consider that in both experiments, more than 50% of the time RCD failed to identify the real cause of anomalies (i.e., the application service stressed) in system metrics. It is essential to note that the conclusions regarding RCA might not be representative of the currently available techniques for RCA, as only one technique has been examined. This limitation is also addressed in Section 9. However, this is the first time such a technique is applied to energy consumption data, making these results a good starting point for further exploration.

We recommend that system operators and other practitioners refrain from using RCA solutions to detect the root causes of anomalies in energy consumption metrics gathered from microservice-based applications. The findings of our analysis show that more often than not, the results RCA generates are not trustworthy and the predictions made might not include the actual root cause of an anomaly in the energy consumption metrics of a system. An exception to this statement is the case in which the root cause is related to CPU anomalies, as discussed in Section 7. However, this is not enough to justify the use of RCA on system metrics, as the CPU is only one of the various hardware components that are used by application services to perform their functionality, all of which might experience anomalies in their utilisation. Instead, practitioners should turn to other types of solutions to achieve the goal of detecting root causes of energy anomalies, until improvements in the ability of RCA on energy consumption data have been made.

Furthermore, we found the field of root cause analysis on energy consumption data and related anomalies to be mostly unexplored. No particular studies have focused on using energy consumption metrics for such processes. Moreover, the difficulties in the monitoring of microservice-based systems, the growing use of this style of application development and deployment and the potential benefits of promptly and effectively detecting energy anomalies in such systems enhance the need for solutions specialised in energy consumption data. In future studies, researchers should focus on achieving this objective, while at the same time minimising the overhead for deployment of such solutions on target systems. This way their usage will be encouraged and the results can be helpful for usage by practitioners.

Overall, both anomaly detection and root cause analysis exhibit poor performance in accurately detecting a large percentage of anomalies or their root causes, according to data collected in this study. This is despite the high precision that AD yields, which enhances the trustworthiness of the generated results. This could result in systems with anomalies that are left unattended and untreated for long periods before they become apparent either by system performance degradation or even the execution of AD on another type of metrics, such as hardware utilisation metrics. This is an indication that both tools are potentially not good on their own and their use should be complemented or substituted by other tools that conduct the same functionality on non-energy data or complementary actions that increase discovery of issues in microservice-based systems. Thus, system operators should not rely solely on energy consumption metrics to promptly and accurately discover issues in their systems. Moreover, the deployment of AD and RCA tools that utilise energy metrics might introduce an overhead that does not justify any related expenses, both financially and time-wise. For this reason, researchers need to conduct further studies and development of anomaly detection and root cause analysis tools that are capable of performing adequately well to justify their employment and promote sustainability by supporting the functionality of AD and RCD processes to integrate energy consumption data. The findings outlined in the present paper can be used as the foundation and the starting point of such research and analysis.

9 THREATS TO VALIDITY

In this section, we aim to assess the overall reliability and applicability of the experiment results by examining potential threats to their validity. To achieve this, we adopt the threat classification approach introduced by Cook and Campbell (1972) [41].

9.1 Internal Validity

This category comprises threats strongly related to the design and execution of the experiment.

9.1.1 ***History***. The paper presents the implementation of two experiments. The first experiment utilised a medium-complexity research-oriented reference application called SockShop, while the second experiment employed a industrial system named Zahori. Both experiments followed the same design, which included randomisation of the order of the trial execution, spreading the effects of uncontrolled factors across different subjects and treatments.

In the experiment involving SockShop no interruptions occurred after the experiment began, ensuring the reliability and integrity of the results were not compromised. However, in the Zahori experiment, the experiment was disrupted several times throughout its run. The cause of these interruptions was the crash of the SmartWatts tool, which runs alongside the applications to measure their power consumption. It was observed that the rapid creation of multiple containers, essential for Zahori's operation during process execution, led to the SmartWatts tool crashing. The exact reason behind this phenomenon has yet to be identified. These interruptions have introduced unreliability within the environment and, for the experiment, have introduced a selection bias as the failed attempts have been ignored. As soon as the script detected SmartWatts had crashed, we reset its functionality and continued the experiment from its last successful execution.

9.1.2 ***Reliability of Measures.*** The are no environmental factors that could interfere with the measurements that are collected from the target applications. Before the experiment started, we ensured that no other services or intensive processes were running, apart from the necessary OS processes and the monitoring tools that capture hardware utilisation and power consumption. This means that all the metrics for the application services reflect the actual utilisation of the services and are not affected by external processes. The interference of the monitoring tools is considered negligible and always the same across treatments and systems. Moreover, the same stress tests were applied to all the services, for the same duration and under consistent application traffic.

To avoid any potential periodic interference, the combination of services stressed, type of stress and stress test intensity was randomly selected for each run. As a result, any factors that could affect the experiment results are minimised.

9.2 External Validity

Threats to external validity restrict the level of generalisability of the experiment results.

9.2.1 ***Interaction of selection and treatment.*** For the first experiment conducted for this study, we selected to use one research-oriented application to refine and validate the reliability of the experiment design and the process that is followed to capture application service metrics. The second part of the study involves the employment of an industrial system in order to generalise our findings on a real-world application and systems developed with external end-users in mind. This setup enhances the reliability and generalisability of the results and findings of this research.

It is essential to note that both systems are not excessively large or complex, which might limit the relevance of our study's results for more extensive, distributed systems with intricate interactions between services [42]. To address this limitation, future research could explore the application of our experiment design and process on more substantial and complex systems to validate or refute our findings for applications of a larger scale. That said, this study's findings apply to a wide range of microservice-based applications that do not rely on extremely complicated architectures and service interactions.

9.2.2 ***Interaction of setting and treatment.*** In order to mitigate threats to the validity of our findings stemming from the environment in which the systems were deployed and the experiments were run, we eliminated all non-necessary processes running on the server that hosts our applications before executing the experiments. Both system employed in this study were deployed on a Unix-based OS to approximate real-world conditions, as Unix is one of the most popular OS to deploy cloud applications on [43]. Moreover, predictable traffic levels were directed towards the application services, emulating real-world application traffic. In the case of SockShop, a locust[31] script generates traffic to specific endpoints of the application that affect most other connected services. Regarding Zahori, a small process is run repeatedly while system services are stressed. The stress of application services, which leads to anomalies in the metrics, is achieved using the same stressor for the same type of stressing, i.e. CPU, memory and file system stressing, every time.

9.3 Construct Validity

Factors that threaten the construction of the experiment constitute threats to the construct validity.

9.3.1 ***Inadequate pre-operational explication of constructs.*** Applying the GQM approach, we laid the groundwork for our research by establishing its constructs even before the execution phase. This early definition of the objective, research questions, and measurement methods provided a solid basis for subsequent steps. Furthermore, in the experiment planning stage (Section 5), we determined the research subjects, identified the dependent and independent variables, and outlined the treatments for our factors in the study. By adhering to the GQM approach, we ensured a clear and well-defined framework for designing and operating the experiments and collecting relevant measures to answer our research questions.

9.3.2 ***Mono-method bias.*** To measure the power consumption of the applications employed in this study, we used the SmartWatts tool. This is a software-based solution that takes advantage of specific OS and hardware features to make a highly-accurate estimation of the real consumption a service makes. However, the metrics collected using this solution are not verified using another software-based tool. Furthermore, the SmartWatts readings are expected to be less accurate than the ones generated by hardware-based power meters. In order to partially mitigate these concerns, we performed 20 runs for each distinct experiment trial to verify the reliability of the captured measures. That said, SmartWatts has been assessed to be fairly accurate when measuring the power consumption of monitored systems when compared to other power consumption profilers [44].

Similarly, to support the feasibility of the study, a single solution was utilized for anomaly detection and another for root cause analysis. Consequently, the conclusions drawn from this research may only apply to the specific tools employed and might not reflect the broader practices of anomaly detection and root cause analysis. For this reason, further studies could explore the effectiveness of alternative techniques and algorithms for anomaly detection and root cause analysis. This would enhance comprehension regarding the topic of anomaly detection and root cause analysis using energy consumption metrics, thereby either corroborating the findings observed in this study or rejecting them. That said, the algorithms used in this study for anomaly detection are widely used solutions for research and development purposes [45–49].

10 CONCLUSION

In this research, we explore and evaluate the effectiveness of the processes of anomaly detection and root cause analysis applied to

[31]https://locust.io/

data related to energy consumption that microservice-based applications exhibit. Specifically, we employ two distinct systems, SockShop and Zahori, to validate the robustness and reliability of our findings and expand their scope to systems of varying complexities and purposes. This evaluation involves computing adjusted metrics, in particular precision, recall and F1-score for AD to gauge its correctness, completeness and accuracy. The rationale behind utilising adjusted versions of these metrics stems from the fact that anomalies often occur continuously, forming contiguous segments of abnormal behaviour rather than isolated instances. For RCA, we assess its correctness using the precision at the top k level (P@k) metric, a widely accepted measure for assessing ranked results.

The results derived from the analysis reveal that anomaly detection can produce accurate predictions about the existence of anomalies within metric segments. However, both AD and RCA tend to overlook many instances that should be flagged as either anomalous or the cause of the detected anomalies, respectively. In particular, the average precision of AD, across all the models employed, is approximately 99%, while it exhibits an average recall of roughly 40% and an average adjusted F1-score of 57%. Furthermore, when evaluating the top 3 predictions generated by the RCD, and subsequently, for our study, the RCA process, the average precision stands at approximately 40%. Such performance levels fail to justify the operational overhead associated with deploying and operating AD and RCA tools that utilise energy consumption data and necessitate infrastructure deployment to monitor the power or energy usage of individual services comprising microservice-based applications. In the case of AD, the overhead is potentially increased by the need to deploy a complementary solution to reduce the missed anomalous cases that AD cannot detect by itself. Regarding RCA, a completely different solution is essential at the time of this research, as the results were found not sufficient to detect root causes of energy consumption anomalies.

It is imperative that further research endeavours be undertaken in this field to produce tools and methodologies capable of effectively utilising energy consumption metrics stemming from microservice-based applications in a way that provides adequate performance. This pursuit aims to establish techniques that offer satisfactory performance, thereby fostering the adoption of such methods and encouraging a sustainability-oriented approach to system management. Ultimately, this would facilitate the prompt identification of anomalies and their root causes within systems employing the microservice architecture leading to more accurate alerts and faster treatment of system malfunctions. Furthermore, similar studies to this one must be conducted on new systems and using a wider range of tools for anomaly detection and root cause analysis. This way, a better understanding of the current state of the field is achieved, prompting appropriate actions to be taken.

REFERENCES

[1] L. De Lauretis, "From monolithic architecture to microservices architecture," pp. 93–96, 2019.

[2] A. Singleton, "The economics of microservices," *IEEE Cloud Computing*, vol. 3, no. 5, pp. 16–20, 2016.

[3] J. Soldani, D. A. Tamburri, and W.-J. Van Den Heuvel, "The pains and gains of microservices: A systematic grey literature review," *Journal of Systems and Software*, vol. 146, pp. 215–232, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218302139

[4] J. Soldani and A. Brogi, "Anomaly detection and failure root cause analysis in (micro) service-based cloud applications: A survey," *ACM Comput. Surv.*, vol. 55, no. 3, feb 2022. [Online]. Available: https://doi.org/10.1145/3501297

[5] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth, "Performance anomaly detection and bottleneck identification," *ACM Comput. Surv.*, vol. 48, no. 1, jul 2015. [Online]. Available: https://doi.org/10.1145/2791120

[6] M. Jin, A. Lv, Y. Zhu, Z. Wen, Y. Zhong, Z. Zhao, J. Wu, H. Li, H. He, and F. Chen, "An anomaly detection algorithm for microservice architecture based on robust principal component analysis," *IEEE Access*, vol. 8, pp. 226 397–226 408, 2020.

[7] T. F. Düllmann, "Performance anomaly detection in microservice architectures under continuous change," Master's thesis, 2017.

[8] I. Kohyarnejadfard, D. Aloise, S. V. Azhari, and M. R. Dagenais, "Anomaly detection in microservice environments using distributed tracing data analysis and nlp," *Journal of Cloud Computing*, vol. 11, no. 1, pp. 1–16, 2022.

[9] A. Ikram, S. Chakraborty, S. Mitra, S. Saini, S. Bagchi, and M. Kocaoglu, "Root cause analysis of failures in microservices through causal discovery," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 31 158–31 170. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/c9fcd02e6445c7dfbad6986abee53d0d-Paper-Conference.pdf

[10] J. Bogatinovski, S. Nedelkoski, J. Cardoso, and O. Kao, "Self-supervised anomaly detection from distributed traces," pp. 342–347, 2020.

[11] Y. Gan, Y. Zhang, K. Hu, D. Cheng, Y. He, M. Pancholi, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of performance debugging in cloud microservices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 19–33. [Online]. Available: https://doi.org/10.1145/3297858.3304004

[12] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, D. Liu, Q. Xiang, and C. He, "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 683–694. [Online]. Available: https://doi.org/10.1145/3338906.3338961

[13] P. Wang, J. Xu, M. Ma, W. Lin, D. Pan, Y. Wang, and P. Chen, "Cloudranger: Root cause identification for cloud native systems," in *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2018, pp. 492–502.

[14] Y. Gan, M. Liang, S. Dev, D. Lo, and C. Delimitrou, "Sage: Practical and scalable ml-driven performance debugging in microservices," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 135–151. [Online]. Available: https://doi.org/10.1145/3445814.3446700

[15] S. Ahmad, A. Lavin, S. Purdy, and Z. Agha, "Unsupervised real-time anomaly detection for streaming data," *Neurocomputing*, vol. 262, pp. 134–147, 2017, online Real-Time Learning Strategies for Data Streams. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217309864

[16] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," in *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, 2020, pp. 1–9.

[17] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," New York, NY, USA, p. 215–224, 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939712

[18] S. Nedelkoski, J. Cardoso, and O. Kao, "Anomaly detection from system tracing data using multimodal deep learning," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 179–186.

[19] Z. Li, Y. Zhao, R. Liu, and D. Pei, "Robust and rapid clustering of kpis for large-scale anomaly detection," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–10.

[20] C. Wang, K. Wu, T. Zhou, G. Yu, and Z. Cai, "Tsagen: Synthetic time series generation for kpi anomaly detection," *IEEE Transactions on Network and Service Management*, vol. 19, no. 1, pp. 130–145, 2022.

[21] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Causal inference techniques for microservice performance diagnosis: Evaluation and guiding recommendations," pp. 21–30, 2021.

[22] B. Familiar, *What Is a Microservice?* Berkeley, CA: Apress, 2015, pp. 9–19. [Online]. Available: https://doi.org/10.1007/978-1-4842-1275-2_2

[23] M. Viggiato, R. Terra, H. Rocha, M. T. Valente, and E. Figueiredo, "Microservices in practice: A survey study," 2018.

[24] Flexera, "State of the cloud report," Flexera, Tech. Rep., 2019. [Online]. Available: https://info.flexera.com/CM-REPORT-State-of-the-Cloud

[25] M. Jay, V. Ostapenco, L. Lefevre, D. Trystram, A.-C. Orgerie, and B. Fichel, "An experimental comparison of software-based power meters: focus on cpu and gpu," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2023, pp. 106–118.

[26] A. Noureddine, R. Rouvoy, and L. Seinturier, “A review of energy measurement approaches,” *Operating Systems Review*, vol. 47, no. 3, pp. 42–49, Dec. 2013. [Online]. Available: https://inria.hal.science/hal-00912996

[27] G. Fieni, R. Rouvoy, and L. Seinturier, “Smartwatts: Self-calibrating software-defined power meter for containers,” in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. Melbourne, Australia: IEEE, 2020, pp. 479–488.

[28] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A.-C. Orgerie, and B. Fichel, “An experimental comparison of software-based power meters: focus on CPU and GPU,” Bangalore, India, p. 13, May 2023. [Online]. Available: https://inria.hal.science/hal-04030223

[29] X. Jiang, Y. Pan, M. Ma, and P. Wang, “Look deep into the microservice system anomaly through very sparse logs,” New York, NY, USA, p. 2970–2978, 2023. [Online]. Available: https://doi.org/10.1145/3543507.3583338

[30] E. Capra, C. Francalanci, and S. A. Slaughter, “Measuring application software energy efficiency,” *IT Professional*, vol. 14, no. 2, pp. 54–61, 2012.

[31] L. Aversano, T. Bodhuin, G. Canfora, and M. Tortorella, *A framework for measuring business processes based on GQM*. Big Island, HI, USA: IEEE, 2004.

[32] H. Wang and H. Zheng, *True Positive Rate*. New York, NY: Springer New York, 2013, pp. 2302–2303. [Online]. Available: https://doi.org/10.1007/978-1-4419-9863-7_255

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

[34] Y. Li, Y. Lu, J. Wang, Q. Qi, J. Wang, Y. Wang, and J. Liao, “Tadl: Fault localization with transformer-based anomaly detection for dynamic microservice systems,” in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2023, pp. 718–722.

[35] L. Wu, J. Bogatinovski, S. Nedelkoski, J. Tordsson, and O. Kao, “Performance diagnosis in cloud microservices using deep learning,” in *Service-Oriented Computing – ICSOC 2020 Workshops*, H. Hacid, F. Outay, H.-y. Paik, A. Alloum, M. Petrocchi, M. R. Bouadjenek, A. Beheshti, X. Liu, and A. Maaradji, Eds. Cham: Springer International Publishing, 2021, pp. 85–96.

[36] L. Yang, J. Li, K. Shi, S. Yang, Q. Yang, and J. Sun, “Micromilts: Fault location for microservices based mutual information and lstm autoencoder,” in *2022 23rd Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2022, pp. 1–6.

[37] J. Lin, P. Chen, and Z. Zheng, “Microscope: Pinpoint performance issues with causal graphs in micro-service environments,” in *Service-Oriented Computing*, C. Pahl, M. Vukovic, J. Yin, and Q. Yu, Eds. Cham: Springer International Publishing, 2018, pp. 3–20.

[38] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng, J. Chen, Z. Wang, and H. Qiao, “Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications,” Republic and Canton of Geneva, CHE, p. 187–196, 2018. [Online]. Available: https://doi.org/10.1145/3178876.3185996

[39] R. Xin, P. Chen, and Z. Zhao, “Causalrca: Causal inference based precise fine-grained root cause localization for microservice applications,” *J. Syst. Softw.*, vol. 203, no. C, jul 2023. [Online]. Available: https://doi.org/10.1016/j.jss.2023.111724

[40] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[41] T. D. Cook and D. T. Campbell, *Quasi-experimentation: Design &#38; Analysis Issues for Field Settings*. Wadsworth Publishing Company, 1979.

[42] D. Huye, Y. Shkuro, and R. R. Sambasivan, “Lifting the veil on {Meta’s} microservice architecture: Analyses of topology and request workflows,” in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 419–432.

[43] R. B. William Voorsluys, James Broberg, *Introduction to Cloud Computing*. John Wiley & Sons, 2011.

[44] M. Jay, V. Ostapenco, L. Lefèvre, D. Trystram, A.-C. Orgerie, and B. Fichel, “An experimental comparison of software-based power meters: focus on cpu and gpu,” in *CCGrid 2023-23rd IEEE/ACM international symposium on cluster, cloud and internet computing*. IEEE, 2023, pp. 1–13.

[45] V. Hautamaki, I. Karkkainen, and P. Franti, “Outlier detection using k-nearest neighbour graph,” in *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, vol. 3, 2004, pp. 430–433 Vol.3.

[46] M.-Y. Su, “Real-time anomaly detection systems for denial-of-service attacks by weighted k-nearest-neighbor classifiers,” *Expert Systems with Applications*, vol. 38, no. 4, pp. 3492–3498, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417410009450

[47] Z. Cheng, C. Zou, and J. Dong, “Outlier detection using isolation forest and local outlier factor,” in *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, ser. RACS ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 161–168. [Online]. Available: https://doi.org/10.1145/3338840.3355641

[48] Z. Ding and M. Fei, “An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window,” *IFAC Proceedings Volumes*, vol. 46, no. 20, pp. 12–17, 2013, 3rd IFAC Conference on Intelligent Control and Automation Science ICONS 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016314999

[49] C. Li, L. Guo, H. Gao, and Y. Li, “Similarity-measured isolation forest: Anomaly detection method for machine monitoring data,” *IEEE Transactions on Instrumentation and Measurement*, vol. 70, pp. 1–12, 2021.