Mittuniversitetet
MID SWEDEN UNIVERSITY

# IoT as Fog Nodes: An Evaluation on Performance and Scalability

Ishaq Ezaz

Bachelor Thesis

**Main field of study**: Computer Engineering

**Credits**: 15

**Semester**: 2023:

**Supervisor**: Stefan Forsström

**Examiner**: Patrik Österberg

**Course code**: DT099G

**Programme**: Master of Science in Engineering – Computer Engineering

Sundsvall 20/06/2023

**Location and date**

Master of Science in Engineering – Computer Engineering

**Programme/Course**

Ishaq Ezaz

**Name**

1998

**Year of birth**

# Abstract

With the exponential growth of the Internet of Things (IoT), managing the enormous amount of data generated has become a significant challenge. This study investigates the distributed paradigm of fog computing, using cost-effective IoT devices as fog nodes, as a potential solution for the centralized cloud. The scalability and performance of a fog computing system were evaluated under a range of workloads, using computationally intensive tasks reflective of real-world scenarios. Results indicated that with an increase in the number of fog nodes, system scalability improved, and the overall latency decreased. However, at lower workloads, configurations with fewer fog nodes outperformed those with more, highlighting the importance of the balance between computation and communication overheads. Overall, this study emphasizes the viability of fog computing as an efficient and scalable solution for data processing in IoT systems. Although the study primarily focused on latency, the insights gained could guide future design and implementation of fog computing systems and contribute to the ongoing discussions on IoT data processing strategies.

**Keywords:** Internet of Things (IoT), fog computing, fog nodes, distributed computing, cloud computing, scalability, latency, performance, data processing, communication overhead.

## Abstrakt

I takt med den exponentiella tillväxten av Internet of Things (IoT) har utmaningen att hantera den enorma mängden genererade data blivit allt större. Denna studie undersöker paradigmen med distribuerade dimdatorer, där kostnadseffektiva IoT-enheter används som dimnoder, som en potentiell lösning på de utmaningarna som det centraliserade molnet står inför. Skalbarheten och prestandan hos ett dimdatorsystem utvärderades under en rad olika arbetsbelastningar genererade av beräkningsintensiva uppgifter. Resultaten visade att en ökning av antal dimnoder förbättrade systemets skalbarhet och minskade den totala latensen. Dock visade det sig att konfigurationer med färre dimnoder presterade bättre vid lägre arbetsbelastningar, vilket understryker vikten av balansen mellan beräkningsuppgifter och kommunikationskostnaden. Sammantaget framhäver denna studie dimdatorkonceptets genomförbarhet som en effektiv och skalbar lösning för beräkningsintensiva databearbetning inom IoT. Trots att studiens fokus låg på latens, kan de insikter som vunnits vägleda framtida design och implementering av dimdatorsystem och bidra till de pågående diskussionerna om strategier för datahantering inom IoT.

**Nykelord:** Internet of Things (IoT), dimdatorer, dimnoder, distribuerad system, databearbetning, moln, skalbarhet, prestanda, latens, kommunikationskostnad.

# Acknowledgements / Foreword

I would like to express my appreciation to my supervisor, Stefan Forsström. His knowledge, guidance, and patience have been invaluable to me throughout this process. Despite having numerous students under his supervision, he always made time to provide each of us with equal attention and support, and for that I am thankful.

# Table of Contents

# Terminology / Notation

**Acronyms/Abbreviations**

| | |
|---|---|
| GUI | Graphical User Interface |
| IaaS | Infrastructure as a Service |
| IoT | Internet of Things. |
| I/O | Input/Output |
| IT | Information technology |
| PaaS | Platform as a Service |
| SaaS | Software as a Service |
| SSH | Secure Shell |
| VNC | Virtual Network Computing |

# 1 Introduction

This thesis aims to expand the knowledge on the topic of fog computing and its viability in various industries. It is expected to benefit organizations and individuals seeking to utilize computing resources with maximum efficiency. This project is proposed from the research group at Mid Sweden University.

## 1.1 Background and motivation

The Internet of Things (IoT) has become the foundation of our technology-driven world. With billions of interconnected devices, projected to reach nearly 75 billion by 2025 [1], IoT has revolutionized many industries by enabling new forms of interaction and communications between people and machines [2]. However, the vast amount of heterogenous data from IoT systems presents significant challenges, especially concerning data processing and data storage management [3].

Cloud computing, a prevalent approach in data processing, involves sending data from end device to central servers for processing [4]. While this model has served well for a while, it faces significant challenges when dealing with massive amounts of data generated by the end devices. These challenges include increased latency, high bandwidth usage, and energy consumption, as well as potential security and privacy vulnerabilities due to the data transmission over long distances [5].

It is evident that the need for more decentralized data processing methods is becoming increasingly important as the IoT continues to evolve. Advancements in fields such as autonomous vehicles, smart cities and homes, healthcare technologies, and industrial automation are driving the demand for real-time data processing capabilities. These applications require not only quick processing, but also the ability of handling big data ensuring data security and privacy [3]. Furthermore, the growing awareness of environmental sustainability is demanding a re-evaluation of existing processing methodologies [6].

The distributed computing paradigm, known as fog computing, has emerged as a promising potential solution to these challenges. Its practical implementation, particularly by leveraging the computational capabilities of cost-effective IoT devices as fog nodes, is a potential area of exploration. This forms the motivation for this study, which aims to investigate the performance, specifically in terms of latency, and scalability of a fog computing system where low-cost IoT devices serve as fog nodes. This objective is to contribute to the expanding knowledge in data processing methodologies.

This study could provide insights for designing efficient, responsive, and sustainable distributed systems for handling data from IoT devices. It could enhance machine learning, video processing, real-time analytics, among other computationally intensive tasks. Therefore, this project is motivated by both the technical challenges of processing methods and the potential of fog computing to supplement cloud computing in the era of IoT.

## 1.2 Overall aim and problem statement

The aim of the thesis is to evaluate the viability of using a fog computing system with IoT devices serving as fog nodes, as a potential alternative in data processing for IoT applications. This will be achieved by examining the performance and scalability of the fog computing system under varying workloads, with a particular focus on latency. The objective is to determine whether fog computing can provide a scalable and cost-effective solution for handling computation-intensive tasks IoT systems, thereby enhancing the reliability and real-time performance of these applications.

## 1.3 Scientific goals

By addressing the following research questions, this project aims to provide insights into the benefits of using Internet of Things (IoT) devices as fog nodes in a fog computing system:

1. What are the most prominent distributed computing approaches that utilize IoT devices as fog nodes?

2. How does the performance and scalability of a fog computing system, implemented using a distributed computing framework and IoT devices vary under different workloads?

3. What are the trade-offs between the latency and scalability?

## 1.4 Scope

The primary focus of this project is on the latency metrics of the fog computing system. This thesis will make sure that the resources of the fog nodes are maximized for the conducted tests. Although other aspects such as security and privacy are important to fog computing, they fall outside the scope of this project. They will not be directly addressed to maintain focus on the performance metric and simplify the system's set up.

Additionally, this project will not involve the use of cloud services. Given the primary focus on latency within a fog computing context, the exclusion of cloud services should not impact the evaluation of the system's latency and scalability.

## 1.5 Outline

Chapter 2 describes the different components of the problem, including IoT, cloud computing, edge and fog computing, and distributed computing. Chapter 3 outlines the methodology adopted for this project and elaborates on the work division. Chapter 4 covers the different approaches that was found from the literature review, a comparison of these approaches, and the approach selected. Chapter 5 discuss the implementation of the system used in the conducted test. The results from tests will be shown in Chapter 6. Chapter 7 will offer a discussion regarding these results, and the final chapter 8 will encapsulate the conclusion and suggest future work.

# 2 Theory

This section provides the theoretical background required for the understanding of the concepts involved in this study. This chapter will cover key topics such as cloud computing, the Internet of Things, fog and edge computing, and distributed computing frameworks. It presents the necessary context and relevant technological concepts, supported by examples and illustrative figures.

## 2.1 Internet of Things

The Internet of Things (IoT) refers to internet-connected devices capable sending and receiving data. This growing network of devices, equipped with software, sensors, and network connectivity, enables the collection and processing of data.

The Internet of Things foster a variety of advantages such as increasing productivity and enhancing operational efficiency. For example, in the context of smart homes, IoT technologies allow various applications to interconnect and automate numerous functions, including, environmental control like temperature and humidity adjustments, and security management. In healthcare, IoT technologies play an integral role in patient monitoring and management. Wearable devices can track vital signs and alert medical workers when irregularities occur, making it possible for preventative care and enhancing patient outcomes [7].

Despite its many advantages, IoT also introduces several challenges. The huge amount of data generated by these end devices need requires significant computing resources for processing and storage. Moreover, real-time data processing from these devices significantly increases the demand on computing resources.

This growth in data generation and processing demands impacts cloud computing infrastructure, as the distance between the end devices and cloud servers can cause processing latency. Particularly in time-sensitive applications like autonomous vehicles or healthcare systems as mentioned, this may be a huge concern. Additionally, as the number of connected devices increases security and privacy concern arises due to the increased vulnerability [8].

Consequently, the cloud computing model, which involves transmitting data from end devices to the cloud for processing, is found insufficient for the growing IoT network. These challenges have led to alternative computing paradigms such as fog and edge computing, aiming to bring computing resources closer to the data source [9].

## 2.2 Cloud computing

The cloud is an information technology (IT) service delivery model. It provides resources, such as applications, data storage, and other business resources whereby resources are retrieved from the internet through web-based tools, instead of a direct connection to a server [10]. Although data and packages are stored in servers, a cloud computing model allows access to information if an electronic device has internet access [10]. The illustration of the centralized model can be seen in Figure 1 below:

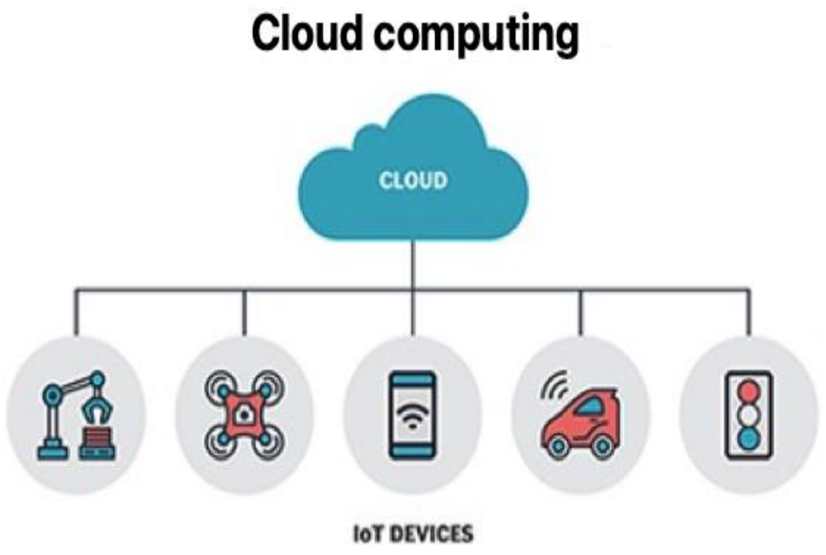

Figure 1: An illustration of cloud computing

Cloud computing provides three major services: Software as a Service (SaaS), which delivers software over the internet on demand; Infrastructure as a Service. (IaaS), which provides computer infrastructure like storage, networks, and operating systems; and Platform as a Service (PaaS), which offers a platform that includes both infrastructure and plat form layer resources like databases and web servers [10].

While cloud computing offers many benefits such as efficiency, scalability, and flexibility, it's also accompanied by limitations such as latency issues, the need for stable internet connectivity due to data transmission distance, and security concerns related to data privacy [11].

## 2.3 Edge computing

Edge computing is a distributed computing paradigm which brings computation and data storage closer to the data source, thereby improving response times [9]. The edge refers to the end of the network, nearest the devices producing or consuming data.

A significant advantage of edge computing is the ability to process data in real-time or near real-time. It removes the necessity of sending data to the cloud for processing. This is highly beneficial for time-sensitive applications. For instance, in autonomous vehicles, where delay may result in catastrophic outcomes, and in healthcare systems, where rapid response is vital.

Another benefit of edge computing is enhanced data security and privacy. When processing data closer to the source, it limits the transmission over networks, reducing exposure to potential cyberattacks. Moreover, edge computing reduces the bandwidth cost by minimizing the need for continuous communication with the cloud [12].

However, edge computing also introduces challenges. Managing and maintaining numerous edge nodes that processes data can be complex [13]. It requires a solid network infrastructure. Moreover, the distributed system of edge computing increases the potential failure within the system which requires security measures and fault tolerance protocols.

## 2.4 Fog computing

Fog computing is another computing paradigm originated by Cisco, that aims to bring the concept of cloud computing to the edge of the network [14]. While edge computing focuses on pushing the computing resources and applications to the edge, fog computing creates an intermediate layer of computing between the cloud and the edge devices, as illustrated in Figure 2.

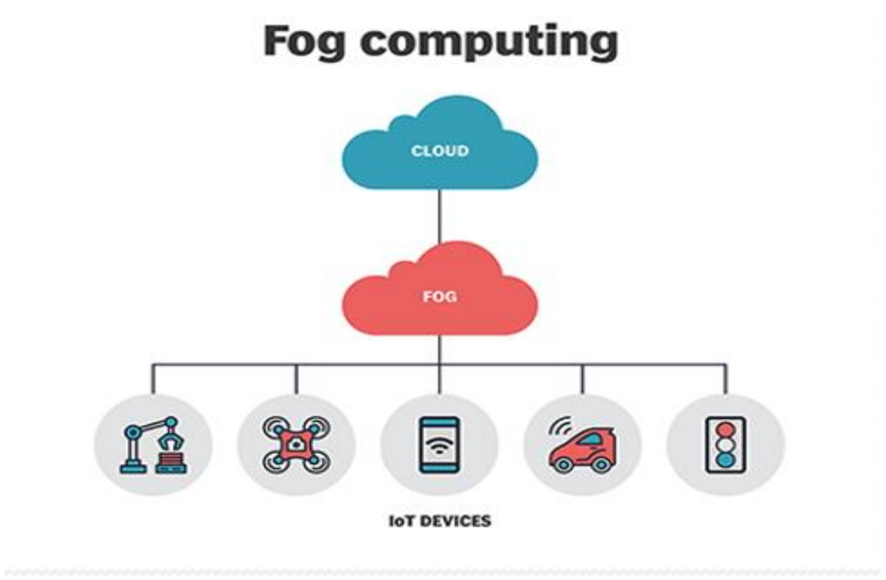

Figure 2: An illustration of fog computing [24]

This decentralized computing system extends from the edge of the network to the cloud. In fog computing, various nodes, known as fog nodes, are utilized that reside between the edge and the cloud servers, storing and processing data [14]. It allows for computation, applications, and data to be distributed in the most logical, efficient place between the data source and the cloud.

This reduced latency resulting from local processing is beneficial for scenarios where immediate computations are necessary, or when sending all data to the cloud would cause network traffic [14]. Additionally, in the context of data privacy, this computing paradigm enhances security by enabling local processing. Sensitive data can be processed closer to the data source, rather than being sent to the cloud, reducing exposure to potential cyberattacks.

Even though edge and fog computing have their differences, as will be discussed in the next section, fog computing poses similar challenges as edge computing. It requires adequate management and security measures for the fog nodes [13]. Additionally, it also requires a solid network infrastructure capable of handling increased processing demands and communication at the network's edge. Despite these challenges, fog computing appears to be a promising solution to the demands of the rapidly growing IoT.

## 2.5 The overlapping concepts

The definitions of fog and edge computing can sometimes overlap and be somewhat intangible. But these paradigms are complementary yet distinct. While both aim to bring computation closer to the data source to reduce latency, their approaches differ:

- Location of computation: Fog computing brings processing closer but not necessarily directly on the devices. It establishes an intermediate layer of computing infrastructure between the cloud and edge [14]. Edge computing on the other hand, involves processing data directly on the edge devices or a local server [12].
- Scope: Fog computing is network-centric, utilizing the network to distribute computing resources effectively [14]. In contrast, edge computing is typically device-centric and targets the individual device's computing capabilities [12].
- Data Flow and Management: In fog computing, processed data may be sent back to the cloud or other fog nodes depending on the needs of the application [14], allowing for more flexible data management. Conversely, in edge computing, data is processed locally on the device and typically not sent back to the cloud [12].

By understanding these differences, it helps identify the best solution based on the specific needs of an application. In some cases, a hybrid model using both paradigms may be most optimal.

## 2.6 Distributed computing

Software platforms for distributed computing serve as key components in fog and edge computing systems. They enable effective allocation and utilization of computing resources. These frameworks consist of libraries and tools that allow developers to build systems capable of running on a network of interconnected nodes, processing data in parallel across the network. This contrasts with the centralized cloud computing model where data is processed on a single machine. These frameworks perform several core functions:

1) Distributing computing frameworks manage the distribution of tasks across the nodes in the network. They ensure that tasks are allocated effectively to use available resources, balance load, and minimize latency.

2) They also oversee the management of computing resources across the network. They control allocation of processing power and storage capacity to ensure efficient resources usage and prevent any single node from becoming a bottleneck.

3) In larger-scale system, the probability of an individual component failing is high. Distributed computing frameworks include functionality for handling failures, such as fault tolerance to ensure that the system can continue to function even when individual nodes go offline [15].

4) Distributed computing frameworks are designed to be scalable, allowing for additional node configurations to handle increasing amounts of data and computational workload. This makes them well-suited to handling the enormous data generated by many IoT applications.

For fog computing, the distributed computing frameworks play an important role. They allow computation to be performed closer to the data sources, reducing latency and network traffic, while also leveraging the computational capabilities of many devices.

## 2.7 Related work

A brief review of two relevant studies focusing on their methodologies and the outcomes.

### 2.7.1 Parallel Computing: Performance of a clustered Raspberry Pi environment vs desktop processors

The study "Parallel Computing: Performance of a clustered Raspberry Pi environment vs desktop processors" [16] explore the use of Raspberry Pi cluster for computational tasks, as this study use of IoT Devices as fog nodes. However, the work differentiates in the focus on fog computing for IoT data processing, emphasizing system scalability and latency reduction. Their work tested Raspberry Pi clusters against traditional processors, while our study evaluates the performance of a fog computing system under various workloads, highlighting the balance between computation and communication overhead.

### 2.7.2 Evaluating Distributed Machine Learning for Fog Computing IoT scenarios

The thesis "Evaluating Distributed Machine Learning for Fog Computing IoT scenarios" [17] delves into the domain of distributed machine learning for IoT devices as alternative to cloud computing, which aligns with this projects investigation of using IoT devices as nodes. However, the difference is the focus on scalability and performance of the fog computing system under different workloads, rather than the application of distributed machine learning.

# 3 Methodology

The chapter will introduce the research methodology and approach used in this study.

## 3.1 Scientific method description

This thesis employs the design science methodology and a quantitative approach to gather and analyze data. Design science focuses on the development and evolution of artifacts or systems that aim to solve real-world problems. This methodology emphasizes the creation of new solutions, which are then evaluated based on the performance measures and criteria. A quantitative approach is a suitable method for measuring and evaluating the system's performance, which is a central focus of this project.

To address the first research question, a literature review will be conducted on distributed cluster computing with IoT devices. The review will identify the most prominent approaches along with their strengths and weaknesses, and the challenges and opportunities in this area. The findings from the review will be used to select the most suitable approach for evaluation using simulations and experiments.

The primary focus of the research will be measuring the system's performance and scalability under various workloads, following the literature review. The selected approach will be used to distribute the workload to the IoT cluster, and measurements will be taken to evaluate the system's performance.

Regarding the third question, the results of the simulations and experiments will be analyzed using quantitative measurements such as latencies. This analysis will account for the trade-offs between different factors and provide insights into the system's performance and scalability under different scenarios and workloads.

## 3.2 Project method description

The project's work will be partitioned into five achievable milestones to aid in organizing and structuring the project. These milestones align closely with the design science methodology and are detailed as follows:

### 3.2.1 Theory

During this phase, the goal is to deepen comprehension on distributed cluster computing, IoT devices, and fog computing environments. A literature review will be conducted to identify the most prominent approaches and frameworks for distributed cluster computing with IoT devices. This will establish a solid foundation of knowledge and understanding of the technologies and techniques relevant to the research.

### 3.2.2 Pre-study

In this phase, the project requirements, approach, design, and methodology will be further refined based on the foundational knowledge acquired from the preceding phase. Specific use cases and scenarios for the fog computing environment will be identified, and the requirements for the IoT devices and cluster computing will be defined. Additionally, this stage will involve the selection of suitable methods for evaluating the performance of the fog computing.

### 3.2.3 Implementation

This stage will be devoted to constructing a prototype of the cluster computing. Suitable hardware and software components will be chosen based on the requirements outlined during the previous milestone. Moreover, a scenario involving varying degrees of workloads will be deployed into the cluster. The performance of the system will then be evaluated based on these workloads.

### 3.2.4 Measurement

In this stage, data will be gathered at various stages to evaluate the performance of the IoT-cluster system. Various measurement tools and techniques will be used to measure the general performance, and other relevant metrics. These tools will include data visualization software or libraries. This stage is crucial for gathering the required data to determine the effectiveness of the implemented system.

#### 3.2.5 Evaluation

The results of the measurements collected in the previous stage will be analyzed and evaluated. The focus will be on the performance of the cluster in terms of latency, with the aim of identifying any bottlenecks and contributing to further research discussions. This evaluation will include data analysis and visualization. Additionally, the evaluation will involve a comparison of the performance of the cluster system under different workloads and conditions. The findings will then be discussed in relation to existing literature on fog computing and IoT-cluster computing to contribute to future research directions.

#### 3.2.6 Potential Challenges

A possible weakness of the project methodology is that it may not account for all the complex and unpredictable factors that may affect the performance and scalability of fog computing environments with IoT-devices. These factors may include network traffic, resource demands and unexpected failures or errors in the system. To address this challenge, a thorough testing and validation of the prototype will be conducted in different scenarios and conditions.

### 3.3 Project evaluation method

To evaluate the success of this project, an evaluation will be done covering the research methodology, the system implementation, the interpretation of results and lastly the discussion.

Methodology efficiency will be evaluated by its ability to generate reliable results. The implementation's effectiveness will be determined by its operation and the reflection of real-world scenarios. The interpretation of data will be considered successful if the results reflect the theoretical hypothesis. Moreover, the discussion's value will be assessed by its insights into IoT data processing. Finally, a reflection on challenges encountered and the strategies used to overcome them will be included. These considerations will determine the project's overall success.

# 4 Approach

This chapter delves into the different approaches presented in the literature review. The primary focus is on distributed computing framework and the IoT device, but it will also consider other critical components such as remote management tool and the scenario.

## 4.1 Distributed computing framework alternatives

Distributed computing frameworks play an essential role in processing the vast amount of data produced by various devices effectively. In this section explores and examines several frameworks, surrounding IoT-centric approaches, with the aim of determining the most appropriate one for the project's requirements.

Conventional distributed computing frameworks have been crucial in addressing the challenges associated with large-scale data processing operations. These frameworks employ distributed computing to effectively process and analyze significant volumes of data. Apache Spark and Apache Flink are among the most prevalent conventional distributed computing frameworks, having made significant contributions to the domain of larger-scale data and being broadly deployed in data centers and cloud environment.

The demand for distributed computing frameworks customized for the distinct attributes and needs of IoT-devices has expanded with their increasing prevalence. IoT-centric frameworks are specifically designed to tackle the challenges and limitations rooted in IoT-devices, such as limited resources. The emergence of distributed computing frameworks, such as Ray, Eclipse IoFog, and FlogFlow, addresses the unique challenges of these devices.

### 4.1.1 Ray.io

Ray.io is a high-performance distributed computing framework adapted to lightweight devices. It enables effortless execution of parallel and distributed algorithms, which is crucial for real-time analytics and machine learning applications. Its adaptable architecture allows for easy scalability and adaptability addressing the diverse nature of lightweight devices. Ray's Actor model and task-based API enable efficient resource allocation and granular control, reducing overall resource consumption and latency [18]. This makes Ray.io especially appropriate for IoT environments with constrained resources and real-time processing demands.

### 4.1.2 Eclipse IoFog

Eclipse IoFog is another distinct distributed computing framework with a specific focus on IoT devices. It offers a computing platform aiming at bringing the power of the cloud closer to the data source. Developers can easily deploy and manage services at the edge, enabling faster data processing. The flexible architecture of IoFog allows it to easily scale up, adapting to the changing demands of workload. This feature makes IoFog an ideal candidate for IoT environments that require fast and efficient data processing at the edge [19].

### 4.1.3 FogFlow

FogFlow is yet another IoT-focused distributed computing framework designed to facilitate integration between cloud and edge computing environments. It aims to optimize resource allocation, minimize latency, and enhance scalability through dynamic allocation of data processing tasks across IoT devices and the cloud. FogFlow's context entity programming model allows developers to define processing logic that adapts to different situations, thereby making it suitable for dynamic IoT environments [20].

## 4.2 IoT device alternatives

In this section, several options for IoT-devices that are suitable for integration with fog computing will be explored, considering the specific device models and their corresponding operating system. The two most prominent hardware models considered are the Raspberry Pi 4 model B and Odroid N2+.

### *4.2.1* Raspberry Pi 4 model B

The Raspberry Pi 4 with 4GB RAM is a suitable choice for fog and edge computing. Its processing capabilities make it ideal intensive computation. When coupled with a distributed computing framework, the quad-core CPU of the Raspberry Pi 4 can effectively distribute computations. Additionally, its affordability makes it a cost-effective choice for developers.

For the Raspberry Pi 4, the Raspberry Pi OS Lite (64-bit) is a compelling operating system option. It provides a lightweight platform with essential features, maximizing the available system resources for the application layer. The 64-bit version of the operating system takes full advantage of the Raspberry Pi architecture, enhancing the performance and allowing for more efficient resource management.

### *4.2.2* Odroid N2+

The Odroid N2+ is an impressive alternative to the Raspberry Pi 4 model B for fog and edge computing applications integrated with IoT devices. With an Amlogic S922X Rev.C processor and 4 GB RAM, the Odroid N2+ offers significant computational capabilities for distributed computing frameworks in fog and edge environments.

When considering operating systems for the Odroid N2+, DietPi is a viable option. DietPi is a highly optimized, extremely lightweight Debian-based OS. It is specifically designed for single board computers. It uses minimal disk space and RAM, and its highly configurable according to the specific needs. This makes it a strong candidate as it offers fast and secure platform that efficiently manages system resources.

## 4.3 Scenario alternatives

Evaluating the performance of the system requires employing effective analytical approaches. The focus is on using suitable computational algorithm that can generate a range of workload scenarios that reflects real-world conditions. Additionally, statistical methods are essential for the analysis and interpretation of the results. Using software like MATLAB for visual representation can further assist in understanding latency patterns and the system's overall performance under various workload.

### *4.3.1* Monte Carlo Pi Estimation

Monte Carlo Pi estimation is a computational algorithm that applies the Monte Carlo method to estimate Pi. The algorithm uses random sampling, which forms a computation-intensive task where several points are sampled within a defined space, and the ratio of points falling in a quarter-circle is calculated [21]. This computation-intensive task, when distributed across fog nodes helps creating different workloads to assess the system's performance.

### *4.3.2* Sensor Data Processing

In contrast to the computation-intensive nature of the Monte Carlo Pi estimation method, processing sensor data is more I/O intensive. This task typically involves the continuous collection, analysis, and possibly transformation of data gathered from various sensors. For instance, in a smart city scenario, an IoT device could be receiving data from environmental sensors such as temperature or air quality sensors. These readings may need to be averaged or otherwise processed before being sent on for further use or analysis.

## 4.4 Remote management alternatives

In the context of fog and edge computing, remote management and communication approaches are crucial. These methods allow the operator to manage and control the nodes in the network from a central location, reducing the need for physical interaction with the devices. These approaches can differ based on the requirements of the system, such as the need for secure communication, speed, and simplicity of use.

### *4.4.1* Secure Shell (SSH)

Secure Shell (SSH) is a protocol that provides secure and encrypted communication over the network in a client-server model. SSH allows for remote management of devices, enabling command executing and file transmission. It is widely used due to its security features and ease of use [22].

### *4.4.2* Virtual Network Computing (VNC)

Virtual Network Computing (VNC) is a graphical desktop sharing system that allows to remotely control another computer. This approach may be beneficial for managing node devices with a graphical user interface (GUI) or for applications that require visual feedback [23].

## 4.5 Comparison of Approaches

In this section, Pugh Matrices will be used for evaluating the different approaches based on the identified requirements. In the evaluation, the count of stars will serve as an indicator of better performance, while a minus sign will denote a negligible performance.

### 4.5.1 Comparison of distributed computing frameworks

The identified project requirements need an appropriate distributed computing framework that provides low latency, scalability, community support, and compatibility with lightweight devices. The frameworks are compared based on these requirements.

**Ray** [18].

Advantages:

- Lightweight compatibility: Supports a wide variety of IoT devices.
- Low latency: Ray's actor model and task-based API allow for efficient resource allocation and granular control, reducing overall resource consumption and latency.
- Community: Ray has active community, providing availability to resources and support.
- Scalability: Backed by Actor model for concurrency, which allows it to handle many tasks simultaneously. Its dynamic task scheduling and fault tolerance mechanism also contribute to its scalability.

**Eclipse IoFog**

Advantages:

- Lightweight compatibility: Designed with lightweight devices and edge computing in mind [19].
- Scalability: The management and coordination of edge microservices by the IoFog Controller and its supporting set of components allow for easy scalability [19].

Disadvantages:

- Community: Smaller community than other frameworks may result in less available resources and support.
- Low latency: Lack of documentation on mechanisms that contribute to reduced latency.

**FogFlow**

Advantages:

- Lightweight combability: Tailored to meet the distinct requirements of lightweight devices [20].
- Low latency: Backed by dynamic scheduling, which may reduce overall resource consumption and latency [20].
- Scalability: Enhanced by its parallel data transfer and dynamic scheduling [20].

Disadvantages:

- Community: FogFlow's smaller community may result in less available resources and support.

Table 1: Pugh matrix for the distributed computing framework comparison

| *Requirements* | *Ray* | *Eclipse IoFog* | *FogFlow* |
|---|---|---|---|
| Scalability | ** | * | * |
| Low latency | ** | - | * |
| IoT compatibility | * | * | * |
| Community Support | ** | * | * |

### 4.5.2 Comparison between IoT devices

Upon comparison, both Raspberry Pi 4 and Odroid N2+ possess feature that make them suitable as fog nodes. They both have powerful processing capabilities and compatible operating systems that are ideal for such applications. However, Raspberry Pi 4 has a slight edge due to its cost-effectiveness, striking a balance between performance and affordability. This can be a big advantage when dealing with larger-scale systems. Additionally, Raspberry Pi also benefits from a larger community and extensive support resources, which can be significantly beneficial when dealing with issues.

Table 2: Pugh matrix for the device comparison

| *Requirements* | *Raspberry Pi 4 Model B* | *Odroid N2+* |
|---|---|---|
| Processing Power | ** | ** |
| Operating System | ** | ** |
| Cost | ** | * |
| Community Support | ** | * |

### 4.5.3 Scenario Comparison

While both Monte Carlo Pi Estimation and Sensor Data Processing are illustrative of common tasks in distributed computing, the first mentioned aligns better with the goal of this study. This is primarily due to its computational intensity, which can more effectively leverage the distributed nature of fog and edge computing environments and therefore demonstrate the performance capacity of the IoT fog nodes.

In the case of Sensor Data Processing, its I/O intensity can be crucial in scenarios where data needs to be continuously streamed and processed. However, such a task would not fully engage the computation capabilities of the IoT devices serving as fog nodes, thus potentially underselling their true performance potential.

Moreover, the Monte Carlo Pi Estimation represents tasks that demand high computational resources such as machine learning or real-time analytics, common in many fog computing applications. These kinds of tasks align well with the project's aim of evaluating fog computing as an alternative solution for reducing latency and cost associated with the cloud.

### 4.5.4 Remote Management Comparison

The two primary methods for remote management and communication are SSH and VNC. Both methods are useful and applicable for different scenarios.

**SSH**

Advantages:

- Secure: SSH encrypts data sent over network, thereby providing a high level of security
- Lightweight: SSH does not consume significant computational resources, which is important when working with lightweight devices.
- Flexibility: Most Linux-based systems have inbuilt support for SSH

Disadvantages:

- Command-line interface: SSH does not support GUI-based interaction which may limit its usability for some applications.

**VNC**

Advantages:

- Graphical Interface: VNC allows control over desktop GUI which can be more user-friendly than a command-line interface.
- Flexibility: VNC servers and clients are available for many operating systems.

Disadvantages:

- Heavyweight: Rendering and transmitting graphical data can be resource intensive. This could interfere with the primary tasks of IoT devices as fog nodes.
- Network intensive: VNC can consume significant network bandwidth, especially at high screen resolutions.

Table 3: Pugh matrix for the remote management comparison

| *Requirements* | *SSH* | *VNC* |
|---|---|---|
| Security | * | * |
| Resource usage | ** | * |
| Interface | - | * |
| Setup | ** | * |

## 4.6 Chosen Approach

After a thorough consideration and comparative analysis of different approaches, the following components have been chosen to construct the distributed fog computing system for this project.

Ray has been selected due to its primary focus on flexible, high-performance distributed computing. Ray's support for dynamic task scheduling and its actor model allows for efficient resource allocation and granular control, reducing overall resource consumption and latency. Furthermore, its fault tolerance and decentralized nature make it well-suited for scalable applications.

The selected IoT device is the Raspberry Pi 4 model B. Its processing capabilities, including the quad-core CPU, make it well-suited for fog computing scenarios. Nevertheless, the decision was largely based on its affordability and widespread usage. These factors make it the preferred choice for the project's requirement of a cost-effective and scalable solution.

The Monte Carlo Pi Estimation method was chosen due to its computational intensity, which effectively reflects real-world workload scenarios. It's an ideal candidate to test the performance of the system, especially latency and scalability, which are the primary metrics for this project.

SSH was chosen for its minimal impact on network and CPU resources, making it a suitable choice for the system setup. Its efficient use of resources ensures that the network and CPU are not overloaded, allowing for smooth remote management of device while conducting the tests.

# 5 Implementation

The system implementation, as shown in Figure 3, operates within the same network. The client initiates the process by sending the tasks to the head node, which then distributes tasks across the workers. Once the workers complete their tasks, the head node collects the results and sends them back to the client. The sending latency is measured from when the client sends the tasks to the head node, while the processing latency includes the coordination time from the head node, processing duration at the worker nodes, and the retrieval time back to the client. The sum of these durations forms the total end-to-end latency.

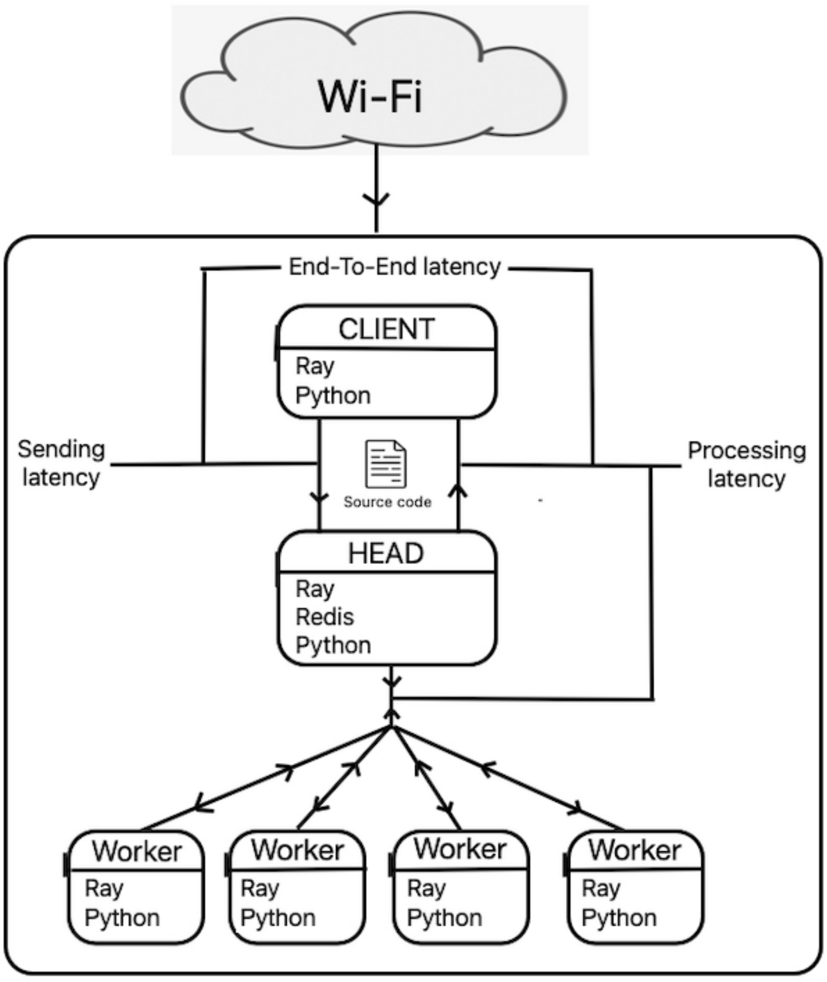

Figure 3: System overview

## 5.1 Wi-Fi

The system necessitates a network configuration that enables communication among all devices, including the client, head node, and worker nodes. A Wi-Fi network with data transfer speeds of 100Mbps was chosen to ensure reliable data exchange.

Following network configuration, the inter-device connectivity was verified by pinging each Raspberry Pi from the client hardware, confirming their network accessibility. To facilitate the setup, firewall was shut down, since it would not affect the performance. However, in a production environment, its crucial to consider security issues and implement appropriate firewall measures.

While using Wi-Fi allows for flexibility in device placement due to the lack of wired connections, care should be taken to minimize physical obstructions or significant distances that could weaken the signal and impact data transfer rates.

## 5.2 Device Configuration

The device configuration process involves the preparation of the Raspberry Pi devices, including the installation of Raspberry Pi OS Lite 64-bit and the configuration for Wi-Fi connectivity.

### 5.2.1 Operating System

The device configuration involves installing the Raspberry Pi OS Lite 64-bit on each Raspberry Pi device. The Raspberry Pi imager was utilized to write the OS image onto the 32 GB SD cards that were to be used with the devices.

### 5.2.2 Network Configuration

To enable the devices to communicate with each other and the client over Wi-Fi, each Raspberry Pi was connected to the same Wi-Fi network. This was achieved by configuring the ‘wpa_supplicant’ file, which allows the Raspberry Pi to automatically connect to a specific Wi-Fi network upon booting.

## 5.3 Software Setup

The software setup involves the installation of Python 3, pip, Ray, and Redis on all devices. The installed versions were Python 3.9.2, and Ray 2.4. This section outlines the process followed to set up the necessary software on the devices.

### 5.3.1 Python and pip

Python and pip, a package manager for Python, are prerequisites for running the Ray framework. The installation of Python and pip on the Raspberry Pi’s was done by using sudo. It’s important to note that the same versions of Python and pip must be installed across all devices for consistency.

### 5.3.2 Ray

Ray was installed using pip. As with Python and pip, the same version of Ray must be installed across all devices.

### *5.3.3* Redis Server

Redis, an in-memory data structure store used as a database, cache, and message broker by Ray, is required only on the head node in the context of this project. Redis was installed on the head node using sudo.

## 5.4 Source Code

The Monte Carlo Pi estimation method, built with Python, is central to the distributed computing operation. It utilizes several Python libraries, including the Ray library, which facilitates distributed computation. For more details, see GitHub [25].

The script contains a function called ‘estimatePi’, decorated with ‘@ray.remote’. This Ray-specific command indicates that the function can be executed in a parallel and distributed manner across the cluster. The ‘estimatePi’ function performs the Monte Carlo estimation of Pi. It defines a nested ‘sample’ function that generates random points within a unit square in the 2D coordinate system and determines whether these points lie within a unit circle. The proportion of points that fell inside the unit circle, multiplied by 4, gives an estimate of Pi based on the Monte Carlo Method.

The main body of the scripts initiates a connection to the Ray cluster through ‘ray.init’ with the IP address of the head node, specifies the details of the Monte Carlo estimation task, and uses ‘ray.get’ to retrieve the results once all tasks are complete.

This setup, where the computational tasks are not only processed but also generated closer to the edge of the network, is consistent with the principles of fog and edge computing. It aims to reduce latency and improve efficiency. Through this script and integration of the Ray framework, the system can distribute the Monte Carlo Pi estimation across a Raspberry Pi cluster, demonstrating the potential of fog computing in efficiently processing complex tasks.

Before running the script, the initialization of Ray on both the head and the worker nodes must be done. The following subchapter will describe the initialization and the roles of the nodes.

## 5.5 Cluster configuration

The initialization of ray, both on the head and worker nodes, is a critical step in setting up the distributed computing environment. This section delves into the initialization process and the specific roles of the head and worker nodes.

### *5.5.1* Head Node

In the distributed computing setup, the head node, also known as the master node, plays the role of a central server. This node is responsible for managing the distribution of tasks, storing shared data, and coordinating the worker nodes.

When initializing Ray, the head node is set up first. This can be done from the command line. It is crucial to specify that head node serves only as a coordinator to prevent it from becoming a bottleneck.

The IP address of the head node is specified in the ‘ray.init’ function in the script, allowing it to serve as the main point of connection for the distributed system.

Once the head node is set up, it is ready to distribute tasks to the worker nodes and manage the computation process.

### *5.5.2* Worker Node

Worker nodes or slave nodes, in the context of distributed computing, refers to the devices that perform the computational tasks assigned by the head node. These nodes are instrumental in the system's ability to process complex tasks more quickly by splitting the workload and processing in parallel.

In the initialization process, worker nodes connect to the head node via its IP address, joining the Ray cluster. Each worker node then stands ready to receive and execute tasks dispatched from the head node.

In the case of the Monte Carlo Pi estimation script, each worker node runs the decorated function, which generates random samples and calculates the proportion of those samples within a unit circle. This data is then returned to the head node for aggregation and final estimation.

In essence, the worker nodes are where the computation takes place in distributed system making them crucial to the fog computing setup.

## 5.6 Evaluation Setup

To evaluate the performance of our distributed fog computing system, a series of tests were conducted involving varying the workload size and the number of worker nodes, while keeping the task size constant. The details of the setup are provided in the following sections.

### *5.6.1* Parameters

The number of samples used in the script was varied, representing the workload size. The task size was kept constant at 10, which were distributed among the worker nodes for execution, as shown in Figure 4. The number of worker nodes involved in the computation was also varied to understand its effect on the system's performance.

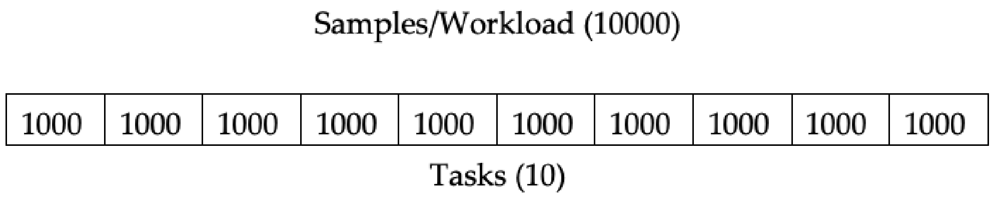

Figure 4: Illustration of the workload

### 5.6.2 Measurements

Three key latency metrics were measured in each test:

- Sending latency: The time it takes to generate all tasks (each with their share of the total workload) and send them to head node.
- Processing latency: The time it takes for the head node to distribute the tasks, and the worker nodes to complete all tasks and return the results.
- End-to-End latency: The total time from task dispatch to results receipt, which is the sum of the sending latency and the processing time.

The sending time for each task was measured as the time difference from the moment of task dispatch. This process was repeated 10 times, and the times for sending each task was summed up. Once the tasks were sent to the cluster, the processing time was measured as the time difference between the dispatch of all tasks and the retrieval of the results.

### 5.6.3 Testing Scenarios

Testing was conducted in a systematic manner, beginning with a minimal workload and a single worker node. Subsequently, the workload and the number of worker nodes were incrementally increased for each test scenario. In each instance, the workload was evenly divided among the tasks and distributed across the cluster. Additionally, each instance was tested 10 times.

### *5.6.4* Analysis

Test results were analyzed by graphing the average the end-to-end latency (y-axis) against the workload size (x-axis). Distinct curves were generated for each worker node configuration, visually representing their performance using MATLAB. Moreover, sending latency and processing latency were graphed in the same manner.

Through the analysis of these graphs, the impacts of varying workload size and scalability on the system's performance can be studied. Detailed test results and the subsequent analysis are provided in the following chapters.

# 6 Results

The results will be represented in both tabular formats, see Tables 5-10, and graphical formats, see Figures 5-7, with the values in the tables rounded to four decimal points. It's important to note that the average values were calculated individually. Therefore, the sum of the sending time and processing time may not always equate to the end-to-end time. For the whole detailed test, refer to the project's GitHub repository [25].

### *6.1.1* Measurements Results

Table 4: Latency comparison for a single worker node

| | ***End to end*** | | ***Send*** | | ***Processing*** | |
|---|---|---|---|---|---|---|
| Workload ($10^3$) | Average (s) | Stdev (s) | Average (s) | Stdev (s) | Average (s) | Stdev (s) |
| 50 | 2.2098 | 0.8919 | 0.0289 | 0.0021 | 2.1808 | 0.8923 |
| 100 | 2.3939 | 0.2811 | 0.0275 | 0.0021 | 2.3654 | 0.2822 |
| 300 | 2.9224 | 0.3538 | 0.0281 | 0.0018 | 2.8942 | 0.3531 |
| 600 | 3.8598 | 0.1527 | 0.0294 | 0.0041 | 3.8303 | 0.1531 |
| 900 | 4.7207 | 0.2005 | 0.0371 | 0.0064 | 4.6835 | 0.1980 |
| 1200 | 5.1371 | 0.3645 | 0.0298 | 0.0036 | 5.1072 | 0.3625 |
| 1500 | 5.8145 | 0.4167 | 0.0288 | 0.0032 | 5.7857 | 0.4159 |
| 1800 | 6.3191 | 0.5179 | 0.0280 | 0.0027 | 6.2911 | 0.5182 |
| 2100 | 7.0274 | 0.5240 | 0.0374 | 0.0119 | 6.9900 | 0.5166 |
| 2400 | 7.6181 | 0.5384 | 0.0289 | 0.0057 | 7.5892 | 0.5384 |
| 2700 | 7.9247 | 0.7044 | 0.0274 | 0.0027 | 7.8973 | 0.7035 |
| 3000 | 8.7420 | 0.6900 | 0.0272 | 0.0024 | 8.7147 | 0.6900 |

Table 5: Latency comparison for 2 worker nodes

| | ***End to end*** | | ***Send*** | | ***Processing*** | |
|---|---|---|---|---|---|---|
| Workload ($10^3$) | Average (s) | Stdev (s) | Average (s) | Stdev (s) | Average (s) | Stdev (s) |
| 50 | 1.0287 | 0.5335 | 0.0339 | 0.0048 | 0.9948 | 0.5320 |
| 100 | 1.7040 | 0.7076 | 0.0298 | 0.0035 | 1.6741 | 0.7098 |
| 300 | 2.6311 | 0.40131 | 0.0297 | 0.0039 | 2.6014 | 0.4023 |
| 600 | 3.1727 | 0.1418 | 0.0292 | 0.0027 | 3.1435 | 0.1402 |
| 900 | 3.4788 | 0.1288 | 0.0284 | 0.0034 | 3.4503 | 0.1276 |
| 1200 | 4.0352 | 0.5031 | 0.0376 | 0.0103 | 3.9976 | 0.5076 |
| 1500 | 4.9376 | 0.3625 | 0.0305 | 0.0076 | 4.9071 | 0.3613 |
| 1800 | 5.4811 | 0.0563 | 0.0320 | 0.0057 | 5.4491 | 0.0532 |
| 2100 | 5.5956 | 0.0880 | 0.0302 | 0.0019 | 5.5654 | 0.0877 |
| 2400 | 5.9383 | 0.2788 | 0.0358 | 0.0094 | 5.9024 | 0.2746 |
| 2700 | 6.0965 | 0.1872 | 0.0432 | 0.0249 | 6.0533 | 0.1820 |
| 3000 | 6.2119 | 0.1887 | 0.0296 | 0.0014 | 6.1823 | 0.1888 |

Table 6: Latency comparison for 3 worker nodes

| | ***End to end*** | | ***Send*** | | ***Processing*** | |
|---|---|---|---|---|---|---|
| Workload ($10^3$) | Average (s) | Stdev (s) | Average (s) | Stdev (s) | Average (s) | Stdev (s) |
| 50 | 0.7735 | 0.4321 | 0.0348 | 0.0052 | 0.7387 | 0.4319 |
| 100 | 1.1652 | 0.5968 | 0.0339 | 0.0030 | 1.1312 | 0.5966 |
| 300 | 1.7567 | 0.4756 | 0.0306 | 0.0030 | 1.7260 | 0.4764 |
| 600 | 2.4629 | 0.1509 | 0.0303 | 0.0042 | 2.43260 | 0.1500 |
| 900 | 2.7163 | 0.1320 | 0.0315 | 0.0046 | 2.6847 | 0.1326 |
| 1200 | 2.9573 | 0.1366 | 0.0341 | 0.0040 | 2.9232 | 0.1335 |
| 1500 | 3.7349 | 0.5243 | 0.0315 | 0.0055 | 3.7034 | 0.52380 |
| 1800 | 4.4094 | 0.2195 | 0.0308 | 0.0043 | 4.3785 | 0.2176 |
| 2100 | 4.8532 | 1.4482 | 0.032 | 0.0046 | 4.8209 | 1.4498 |
| 2400 | 5.3412 | 0.7242 | 0.0318 | 0.0041 | 5.3094 | 0.72352 |
| 2700 | 5.6260 | 0.2210 | 0.0343 | 0.0072 | 5.5916 | 0.2184 |
| 3000 | 5.6345 | 0.2578 | 0.0340 | 0.0065 | 5.6004 | 0.2559 |

Table 7: The latencies for 4 worker nodes

| | ***End to end*** | | ***Send*** | | *Processing* | |
|---|---|---|---|---|---|---|
| Workload | Average | Stdev | Average | Stdev | Average | Stdev |
| ($10^3$) | (s) | (s) | (s) | (s) | (s) | (s) |
| 50 | 0.2944 | 0.0465 | 0.0336 | 0.0037 | 0.2608 | 0.0463 |
| 100 | 0.6520 | 0.5583 | 0.0332 | 0.0036 | 0.6187 | 0.5575 |
| 300 | 0.7146 | 0.5546 | 0.0346 | 0.0031 | 0.6800 | 0.5550 |
| 600 | 1.3644 | 0.5161 | 0.0353 | 0.0032 | 1.3291 | 0.8263 |
| 900 | 1.3020 | 0.7932 | 0.0345 | 0.0034 | 1.2674 | 0.5182 |
| 1200 | 1.7898 | 0.4774 | 0.0343 | 0.0036 | 1.7554 | 0.7915 |
| 1 500 | 2.5318 | 0.7732 | 0.0342 | 0.0035 | 2.4975 | 0.4764 |
| 1800 | 3.1876 | 0.5790 | 0.0373 | 0.0035 | 3.1502 | 0.7738 |
| 2100 | 3.719 | 0.7675 | 0.0358 | 0.0031 | 3.6832 | 0.5806 |
| 2400 | 4.1030 | 0.7675 | 0.0363 | 0.0036 | 4.0667 | 0.7677 |
| 2700 | 4.6946 | 0.7619 | 0.0350 | 0.0031 | 4.6595 | 0.7623 |
| 3000 | 4.7328 | 0.5802 | 0.0351 | 0.0035 | 4.6976 | 0.5801 |

Table 8: The latencies for 5 worker nodes

| | ***End to end*** | | ***Send*** | | ***Processing*** | |
|---|---|---|---|---|---|---|
| Workload | Average | Stdev | Average | Stdev | Average | Stdev |
| ($10^3$) | (s) | (s) | (s) | (s) | (s) | (s) |
| 50 | 0.2517 | 0.0182 | 0.0352 | 0.0039 | 0.2164 | 0.0161 |
| 100 | 0.2748 | 0.0070 | 0.0364 | 0.0037 | 0.2384 | 0.0075 |
| 300 | 0.3674 | 0.0200 | 0.0358 | 0.0027 | 0.3315 | 0.0195 |
| 600 | 0.5518 | 0.0470 | 0.0359 | 0.0025 | 0.5158 | 0.0461 |
| 900 | 0.7171 | 0.1129 | 0.0359 | 0.0037 | 0.6812 | 0.1116 |
| 1200 | 1.3794 | 0.9429 | 0.0358 | 0.0037 | 1.3436 | 0.9443 |
| 1500 | 1.7281 | 1.1012 | 0.0341 | 0.0035 | 1.6940 | 1.1032 |
| 1800 | 2.2884 | 0.9352 | 0.0363 | 0.0034 | 2.2521 | 0.9349 |
| 2100 | 2.6637 | 1.5140 | 0.0366 | 0.0042 | 2.6271 | 1.5129 |
| 2400 | 3.7136 | 1.1105 | 0.0324 | 0.0032 | 3.6811 | 1.1106 |
| 2700 | 3.8663 | 0.9290 | 0.0343 | 0.0039 | 3.8319 | 0.9296 |
| 3000 | 4.3036 | 1.5255 | 0.0328 | 0.0018 | 4.2707 | 1.5256 |

Table 9: The latencies for 6 worker nodes

| | *End to end* | | *Send* | | *Processing* | |
|---|---|---|---|---|---|---|
| Workload ($10^3$) | Average (s) | Stdev (s) | Average (s) | Stdev (s) | Average (s) | Stdev (s) |
| 50 | 0.3204 | 0.1716 | 0.0353 | 0.0038 | 0.2851 | 0.1705 |
| 100 | 0.3392 | 0.1228 | 0.0340 | 0.0032 | 0.3052 | 0.1231 |
| 300 | 0.3728 | 0.0238 | 0.0378 | 0.0045 | 0.3350 | 0.0238 |
| 600 | 0.5918 | 0.0971 | 0.0353 | 0.0043 | 0.5564 | 0.0970 |
| 900 | 0.6974 | 0.0938 | 0.0351 | 0.0025 | 0.6623 | 0.0933 |
| 1200 | 1.0919 | 0.6026 | 0.0343 | 0.0033 | 1.0576 | 0.6033 |
| 1500 | 1.2487 | 0.7384 | 0.0359 | 0.0036 | 1.2128 | 0.7386 |
| 1800 | 1.5225 | 0.8132 | 0.0385 | 0.0031 | 1.4840 | 0.8128 |
| 2100 | 2.2889 | 1.3840 | 0.0381 | 0.0047 | 2.2516 | 1.3827 |
| 2400 | 2.4961 | 1.1232 | 0.0371 | 0.0045 | 2.4589 | 1.1251 |
| 2700 | 2.8968 | 1.0414 | 0.0370 | 0.0028 | 2.8598 | 1.0424 |
| 3000 | 3.1003 | 0.9930 | 0.0365 | 0.0033 | 3.0637 | 0.9939 |

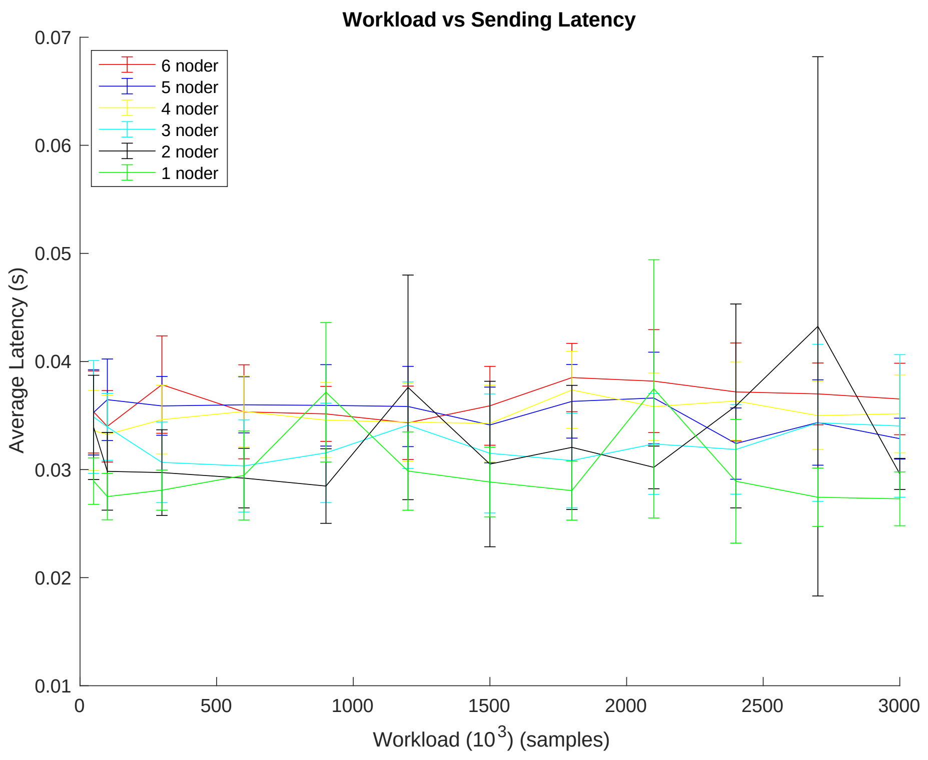

Figure 5: The average sending latencies with standard deviation by worker node configuration and workload.

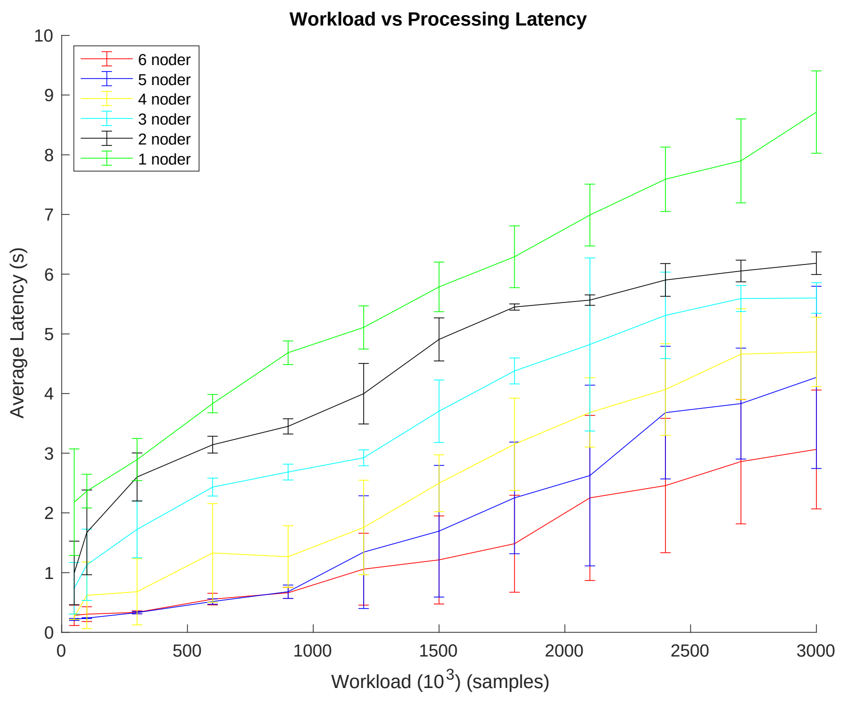

Figure 6: The average processing latencies with standard deviation by worker node configuration and workload.

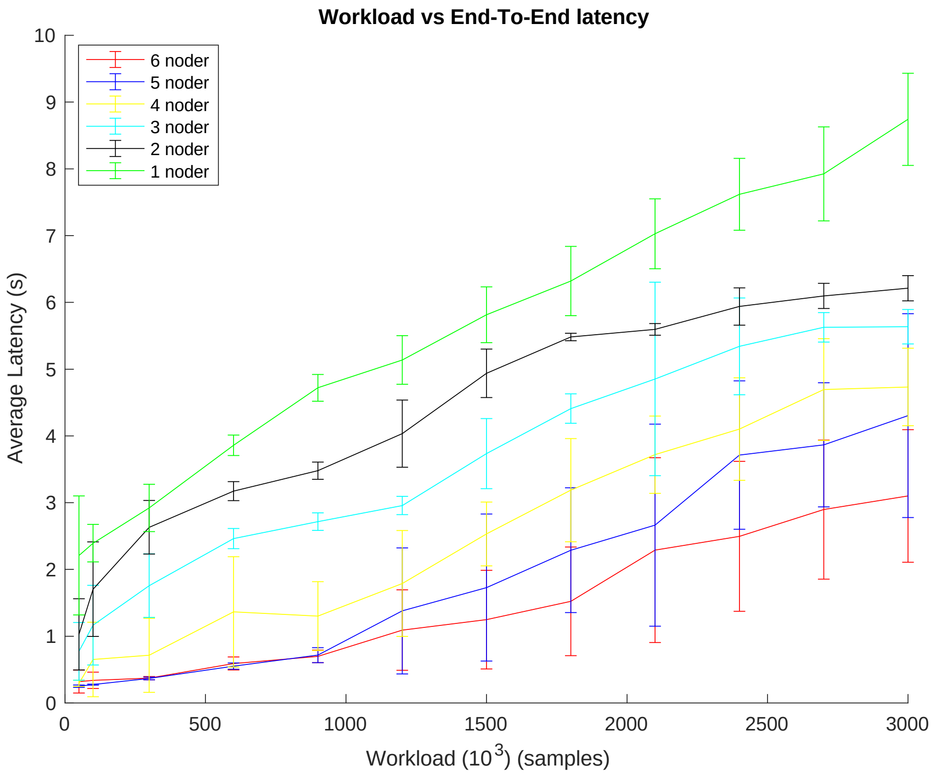

Figure 7: The average end-to-end latencies with standard deviation by worker node configuration and workload.

# 7 Discussion

The objective of this study was to investigate the incorporation of lightweight devices as fog nodes within a fog computing system. The aim thesis aimed to assess system's the performance under various workload conditions and provide insights into the potential advantages and challenges of the approach. Through a series of carefully conducted tests, several trends and outcomes were observed that could provide insights into the future development of fog computing systems.

## 7.1 Analysis and Discussion of Results

This section provides a comprehensive analysis and discussion of the test results, primarily focusing on the effects of workload variability and worker node configuration.

### *7.1.1* The Effect of Workload Variability

The results indicate that as the workload size increases, there is a corresponding increase in the end-to-end latency across the worker node configurations, see Tables 5-10. This increase can primarily be attributed to the enlarged processing time, which naturally increases with the workload size.

The standard deviation for the processing time does not show a clear correlation with the increasing workload size. Despite the workload increment, the standard deviation for the processing time varies seemingly randomly. This inconsistency may be influenced by other factors such as the randomness nature of the task and Rays dynamic scheduling mechanism.

A periodically latency pattern was observed in tests with high processing standard deviation time. This can be attributed to cache and Pythons garbage collection mechanism, which periodically clears memory, leading to increased processing time.

Additionally, back-to-back tests may have caused thermal throttling of the Raspberry Pi's devices. This, combined with potential competition for system resources, may have also contributed to the observed processing time fluctuations.

In contrast, the sending latency remains consistent across the increasing workload. This is expected given its dependence solely on network congestion. Thus, workload should not influence this metric. Although slight variability is observed, it is negligible and does not significantly affect the end-to-end latency.

However, it is critical to consider that real-world scenarios could present different outcomes. Changes in network configuration could potentially affect sending latency, and eventually the end-to-end latency.

Moreover, the quantity of the tasks sent to the cluster was not significant. To ensure some level of parallelism could occur, a consistent quantity of 10 tasks was maintained throughout the tests. Increasing the task quantity could lead to an increase in sending latency caused by the overhead on data serialization. While processing latency would have decreased, the elevated sending latency would counterbalance this benefit, making it a less optimal configuration.

It is important to find a balance between the number of tasks and worker node configuration. To fully leverage parallel computing, there should be at least as many tasks as worker nodes. If the quantity of tasks had been less than the number of worker nodes, some Raspberry Pi's would have been underutilized, leading to inefficiency in resource usage.

### 7.1.2 The Effect of worker node configuration

The test results clearly indicate the impact of worker node scalability on the end-to-end latency. As the number of worker nodes increases, a corresponding decrease in processing latency is observed, leading to a reduced end-to-end latency. This outcome is expected in distributed computing systems. This trend can be attributed to the nature of the tasks in the test, which were independent, asynchronous, and computationally intensive. Such tasks are convenient for distributed computing due to their coarse-grained granularity, which implies that they involve a high amount of computation to data transferred ratio. The coarse-grained nature of these tasks makes them ideal candidates for parallel processing across multiple nodes, maximizing the utilization of the distributed system.

Moreover, a 'cross-over' phenomenon is observed in the results. The phenomenon is characterized by several stages. Initially, configurations with fewer worker nodes deliver low latencies. As the workload increases, configurations with more worker nodes begin to outperform.

Finally, a potential second 'cross-over' occurs. In this stage, adding more workers or increasing the workload leads to configurations with fewer nodes outperforming again. This trend can be attributed to the trade-off between computation and communication overheads. It is typically observed in task that exhibit fine-grained granularity. In this case, a slight indication of this phenomenon was observed, especially between configurations with five and six worker nodes at a low workload range.

At the workload of 50.000 samples, the configuration with five workers had lower end-to-end time than the one with six worker nodes. As the workload increased, a crossover occurred, and the configuration with six nodes outperformed. This could be seen as an indication of the overhead cost, where managing six nodes outweighs the benefit at lower workloads. However, it is important to note that this could also be due to other factors previously mentioned. The latter explanation is likely, as this trend could not be observed across all configurations.

Contrary to expectations, the decrease in processing latency does not seem to be linear. This could be due to factors such as Ray's load balancing mechanism and other underlying mechanism in distributed system. A linear decrease would suggest each additional worker node contributes equally to reducing the processing time. In real-world systems and algorithms such ideal behavior would not be expected due to the complexity in task scheduling and other systemic factors.

As for the sending latency, a correlation between the worker node configuration and time can be argued. It appears that configurations with more worker nodes had an overall higher increased latency. This is reasonable since the sending latency is affected by the network. However, the fluctuations in the sending latency made it difficult to discern a clear trend. These fluctuations could be due to variety of factors, including network instability or intermittent network traffic.

## 7.2 Project Method Discussion

The Design Science methodology with a quantitative approach was chosen for this project. This methodology provided a solid framework for creating and evaluating the system, allowing for iterative development, and testing, which ensured practical functionally.

### *7.2.1* The Selected Approach

Ray was chosen as the distributed computing framework. The choice was influenced by its solid support for task parallelism and actor-based programming, making it a good fit. Furthermore, the extensive community support for ray provided an abundance of resources and solutions to common issues, which facilitated the implementation phase. While the alternative approaches might have been competitive with Ray, their implementation could have been more challenging due to a lack of available resources.

Raspberry Pi 4 model B devices were selected for their affordability, making them ideal candidates for a cost-effective system. They offer enough processing power for the computational tasks involved in the test, as well as network capability to form a distributed environment.

To manage the fog nodes, SSH was utilized. This provided an efficient way to remotely access and control each Raspberry Pi, thereby facilitating the overall system management and reducing resource and network consumption.

The selected scenario is a computationally intensive by nature, making it suitable fit for demonstrating the capabilities of distributed computing utilizing lightweight devices. This scenario mirrors real-world applications tasks where fog computing can provide significant benefits by reducing latency. A sensor reading scenario would indeed yield different results due to its different nature.

### *7.2.2* Project milestones

The overlapping definitions of edge and fog computing required time and effort, a solid understanding of the theories and concepts was achieved. The literature review guided the project, particularly in the selection of the chosen approaches. The complexity of the concepts revealed during the literature review presented unexpected challenges but also offered an opportunity to delve deeper into these concepts.

The initial planning and design took longer due to the overlap between the concepts. However, once the core purpose of latency reduction was understood, the project proceeded smoothly. The selected requirements for the different approaches were suitable. The Monte Carlo Pi estimation scenario used for testing was a fitting representation of real-world applications such as real-time analytics and machine learning.

The implementation phase was straightforward, with minor issues experienced such as inter-connectivity. Using the Raspberry Pi and Ray proved effective due to their simplicity and flexibility. The system's performance was lower than expected initially due to limited internet speed, but this issue was addressed by upgrading the internet connection.

The chosen tools, Python's time module for timing and MATLAB for graphing provided precise and detailed measurements. The system was tested extensively, with the implementation of automated back-to-back testing. However, to ensure accurate results, a 10-second interval was implemented between each test.

The evaluation phase provided insights into the system's performance. The data analysis was thorough, and the visualization of the results demonstrated the system's performance effectively. The results were examined in relation to existing literature on IoT distributed computing, contributing valuable insights for future research.

The project encountered several challenges, including network issues, and unexpected results during the back-to-back testing. However, each challenge was met with solutions. An unexpected issue arose when the head node was initially used both as coordinator and a worker node. However, this was resolved by adjusting the configuration ensuring it is used only as a coordinator to resist the overhead bottleneck situation.

## 7.3 Scientific Discussion

The acquired scientific knowledge has provided valuable insights into the domain of fog computing integrated with lightweight devices as fog nodes. The knowledge includes both specific details related to the approaches used and general concepts regarding concepts of fog computing and the performance versus scalability dynamics in such systems.

The first research question aimed to identify the most prominent distributed computing approaches that utilize lightweight devices as fog nodes. The literature review revealed several potential frameworks, including Ray, FogFlow, and Eclipse IoFog. Ray was selected due to its numerous advantages, such as simplicity, task parallelism based on the actor-model, flexibility across various applications, fault tolerance, and its extensive community support. Although the primary objective of this thesis was to evaluate a fog computing system's performance, typically inclusive of cloud-based elements, the scope of the project did not permit the incorporation of cloud platforms. Nonetheless, the cloud aspect was considered during framework selection to ensure a comprehensive understanding of fog computing. While the system implemented in this project could arguably align more with edge computing, the latency evaluation reflects the benefits of fog computing, as both models primarily aim to reduce latency.

The second research question explored how the performance and scalability of a fog computing system vary under various workloads. The test results demonstrated that as the system scales up with additional worker nodes, latency decreases, thereby affirming the advantages of distributed computing. This finding underscores the fundamental principle of distributed computing. However, it is crucial to maintain a balance between computation and communication to optimize this advantage.

The third research question examined the trade-offs between performance and scalability within a fog computing environment using. The results underscored the overhead cost associated with fine-grained tasks. Despite the scenario used being a great fit for distributed computing due to its coarse-grained and computation-intensive nature, this phenomenon was noticeable at lower workloads and higher scalability. Other contributing factors, such as those mentioned in Section 6.2, should also be considered.

While the results are promising, it is important to note that they are based on a specific set of components, such as Ray, and Raspberry Pi, and a particular scenario. Different approaches and scenarios might present a different result and challenges. Therefore, further research and experimentation are important to confirm the findings and discover new insights.

## 7.4 Consequence Analysis

The exploration of distributed computing with IoT devices fits within the broader scientific discussion around edge and fog computing. This project contributes to the understanding of the practical implementation of such systems highlighting both possibilities and the limitations. It demonstrates practical applicability of IoT devices in a fog computing context and illustrate the performance and scalability dynamics at play in these systems.

The findings suggest several potential directions for future research. One promising area of exploration would be the application of different distributed computing frameworks in a fog computing environment, to compare their performance, scalability, and suitability for different types of workloads. This could further broaden the understanding of the effective integration of fog computing and IoT devices and reveal the strengths and weaknesses of different frameworks.

Furthermore, more detailed investigation of the overhead cross-over phenomena, by focusing on I/O intensive tasks and varying the granularity or workload, could be valuable to cover more applications domains. It would be interesting to see how these variables influence the trade-offs between performance and scalability in a fog computing environment. Fundamentally this project provides a starting point, and there are numerous possibilities for expanding upon this work and further contribute to the domain of edge and fog computing.

## 7.5 Ethical and Societal Discussion

The implementation and application of distributed computing in fog computing context raise several ethical and societal considerations. One key consideration being data privacy and security. As more IoT devices are utilized in fog computing environments, there may be an increased risk of personal data being mishandled and misused. It is important that implementations of such systems apply solid security measures and follow the established data privacy standards. Moreover, developers in this domain have an ethical responsibility to ensure that their work does not create opportunities for threats that could exploit the system or the data.

From a societal perspective, the use of IoT devices as fog nodes can enhance the capabilities of local networks, reducing latency and improving scalability in many applications from smart homes to healthcare. The potential benefits in terms of efficiency and effectiveness can also raise questions around digital inequity caused by demographics and socioeconomics differences. Hence, it is important to ensure that the benefits of these advancements are accessible and distributed equitably across the world. The development in these domains should not only consider the technological and performance aspects but also the broader societal impact and ethical implications.

# 8 Conclusions

In conclusion, this project successfully achieved its goals, addressing the research questions and meeting the problem statement. Through a literature review, several prominent distributed computing frameworks with IoT compatibility were identified. Ray and Raspberry Pi stood out for their advantageous features. The project further demonstrated that as a fog computing system scales up with more fog nodes, system latency decreases, thereby confirming the benefits of distributed computing for computationally intensive tasks. Lastly, the study also identified a slight overhead cost, providing insights into the trade-offs between performance and scalability within a fog computing environment using IoT devices as fog nodes.

## 8.1 Future Work

Based on the findings of this project, several promising directions for future research have arisen.

### *8.1.1* Exploration of Different Distributed Computing Framework

A promising area of exploration involves the application of different distributed computing frameworks in the system. Future work could compare the performance, scalability, and suitability of different frameworks for different types of workloads. This could further broaden the understanding of how effective IoT devices serve as fog nodes and reveal the strengths and weaknesses of different approaches.

### *8.1.2* Investigation of Different Scenarios to Cover More Applications

Moreover, future research could delve into a more detailed investigation of the overhead costs, particularly in the context of I/O intensive tasks. While this approach observed a slight overhead when handling a coarse-grained, and computation intensive tasks at lower workload, many real-world applications of IoT devices in a fog computing environment deal with I/O intensive tasks, such as sensor data processing, and video streaming. Varying the task granularity and adjusting workload levels could provide additional insights into the trade-offs between performance and scalability.

### *8.1.3* Application in Real-World Scenario

This study was conducted under controlled conditions which allowed for more consistent results but might not fully reflect the complexities and variability in real-world environment. Therefore, future research could involve implementing and testing these systems in real-world scenarios where network conditions, data rates, and device availability may vary. For example, a fog computing setup could be deployed in an industrial environment, where the data sources are diverse and network conditions fluctuate. This would provide a solid test of the reliability and adaptability of the system. Understanding the behavior of these systems under more sensitive and unpredictable conditions would be valuable for their future deployment and development.

## References

[1] Statista (2016) 'Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025', 2016. Accessed June 18, 2023 .Available at: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] Maddikunta Reddy Kumar R, Pham QV, Nguyen C D, Huynh-The T, Aouedi O, Yenduri G. Incentive techniques for the Internet of Things: A survey. Journal of Network and Computer Applications. Available from: https://www.sciencedirect.com/science/article/abs/pii/S1084804522001138?via%3Dihub

[3] Vongsingthong S, Smanchat S. A Review of Data Management in Internet of Things. KKU Res J. 2015; 20(2). Available from: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=8f5c87eafb6cc948e125c75f3e0c6f2748d47d54

[4] Ward JS, Barker A. A Cloud Computing Survey: Developments and Future Trends in Infrastructure as a Service Computing. arXiv preprint arXiv:1306.1394. 2013. Available from: https://arxiv.org/abs/1306.1394

[5] Grønli TM, Fazeidehkordi E. A Survey of Security Architectures for Edge Computing-Based IoT. IoT. 2022;3(3):19. Available from: https://www.mdpi.com/2624-831X/3/3/19

[6] Ganesan M, Kor AL, Pattinson C, Rondeau E. Green Cloud Software Engineering for Big Data Processing. Sustainability. 2020;12(21):9255 https://www.mdpi.com/2071-1050/12/21/9255

[7] Mohammed M N, Desyansah SF, Al-Zubaidi S, Yusuf E. An internet of things-based smart homes and healthcare monitoring and management system. Journal of Physics: Conference Series. 2020;1450:012079. Available from: https://iopscience.iop.org/article/10.1088/1742-6596/1450/1/012079/pdf

[8] Atzori Rizzardi A, Grieco LA, Coen-Porisini A. Security, privacy, and trust in Internet of Things: The road ahead. Computer Networks 2015;76:146-164. Available from:

https://www.sciencedirect.com/science/article/abs/pii/S1389128614003971

[9] Laroui M, Nour B, Moungla H, Cherif MA, Afifi H, Guizani M. Edge and fog computing for IoT: A survey on current research activities & future directions. Computer Communications. 2021;180:210-231. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0140366421003327?via%3Dihub

[10] Zhang Q, Cheng L, Boutaba R. Cloud computing: state-of-art and research challenges. Journal of Internet Services and Applications. 2010;1(1):7-18. Available from: https://jisajournal.springeropen.com/articles/10.1007/s13174-010-0007-6

[11] Zissis D, Lekkas D. Addressing cloud computing security issues. Future Generation Computer systems. 2012;28:583-592. Available from: https://www.sciencedirect.com/science/article/pii/S0167739X10002554?via%3Dihub

[12] Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge Computing: Vision and Challenges. IEEE Internet of Things Journal. 2016;3(5):637-646. Available from: https://cse.buffalo.edu/faculty/tkosar/cse710_spring20/shi-iot16.pdf

[13] Roman R, Lopez J, Masahiro Mambo. Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges. Future Generation Computer Systems. 2018;78:680-698. Available from: https://www.sciencedirect.com/science/article/abs/pii/S0167739X16305635?via%3Dihubm

[14] Mahmud, R., Kotagiri, R., Buyya, R. (2018). Fog Computing: A Taxonomy, Survey and Future Directions. In: Di Martino, B., Li, KC., Yang, L., Esposito, A. (eds) Internet of Everything. Internet of Things. Springer, Singapore. https://doi.org/10.1007/978-981-10-5861-5_5

[15] Ledmi A, Bendjena H, Mounine Sofiane M. Fault Tolerance in Distributed Systems: A Survey. 2018 3rd International Conference on Pattern Analysis and Intelligent System (PAIS). 2018. Available from: https://www.researchgate.net/publication/330123117_Fault_Tolerance_in_Distributed_Systems_A_Survey

[16] Sandberg F, Hellman R. Parallel Computing: Performance of a clustered Raspberry Pi environment vs desktop processors. University West, Department of Engineering Science, Division of Mathematics, Computer and Surveying Engineering. 2021. Available from: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1612718&dswid=25

[17] El Ghamri H. Evaluating Distributed Machine Learning for Fog Computing IoT scenarios: A comparison Between Distributed and Cloud-Based Training on Tensorflow. Mid Sweden University, Faculty of Science, Technology and Media, Department of Information System and Technology. 2022. Available from: https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1673115&dswid=5484

[18] Wampler D. Ray for the Curious. Medium. 2019. Updated 2020. Accessed June 19, 2023. Available from: https://medium.com/distributed-computing-with-ray/ray-for-the-curious-fa0e019e17d3

[19] Eclipse ioFog. Core Concepts. 2019. Accessed June 19, 2023. Available from: https://iofog.org/docs/1.1.0/getting-started/core-concepts.html

[20] Cheng B, Solmaz G, Cirillo F, Kovacs E, Terasawa K, Kitazawa A. FogFlow: Easy Programming of IoT Services Over Cloud and Edges for Smart Cities. IEEE Internet of Things Journal. 2018;5(2):696. Available from: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8022859

[21] Gustafsen A. Estimating Pi Using Monte Carlo Simulation in R. Medium. 2021. Available from: https://towardsdatascience.com/estimating-pi-using-monte-carlo-simulation-in-r-91d1f32406af

[22] Loshin P, Cobb M. What is SSH (Secure Shell) and How Does it Work? Definition from TechTarget. SearchSecurity. Available from: https://www.techtarget.com/searchsecurity/definition/Secure-Shell

[23] RealVNC Limited. All You Need to Know About VNC Remote Access Technology. RealVNC. 2022. Accessed June 19, 2023. Available

from: https://discover.realvnc.com/what-is-vnc-remote-access-technology

[24] TechTarget. 'What is fog computing (fogging)?'. Accessed June 20, 2023. Available from: https://www.techtarget.com/iotagenda/definition/fog-computing-fogging

[25] Ezaz I. Examensarbete. Accesed June 20, 2023. Available from: https://github.com/Ishaqezaz/Examensarbete