Fog computing task scheduling

a multi-objective AI-based approach

Théo FIGINI
M2 Computer Science
Academic year 2023-2024

Hosting organization: *LAAS-CNRS*

February, 1st 2024 - June, 25th 2024

Referring teacher:
Emmanuel BIABIANY

Tutors:
Tom GUÉROUT
Andrei DONCESCU

June, 18th 2024

Contents

Résumé

Abstract

1 Introduction

As the number of devices grows exponentially, the demand for computing resources increases. The traditional cloud computing model is not sufficient to handle the massive amount of data generated by these devices. Fog computing is an alternative model that extends the computing resources to the edge of the network, closer to the devices. This model reduces the latency and bandwidth usage, and improves the overall performance of the system. However, the increasing demand for computing resources also raises concerns about the energy consumption and the environmental impact of fog computing systems.

In this context, task scheduling plays a crucial role in optimizing the performance of fog computing systems. Task scheduling algorithms aim to allocate the available resources efficiently to the tasks, while still meeting the Quality of Service (QoS) requirements of the users. Several task scheduling algorithms have been proposed in the literature, ranging from simple heuristics to complex optimization techniques. However, most of these algorithms focus on minimizing the energy consumption or maximizing the performance of the system, without considering the environmental impact of the energy sources used to power the system.

In this paper, we will propose an AI-based task scheduling algorithm that integrate the use of green energy sources while satisfying the QoS requirements of the users. The proposed algorithm will consider the availability of green energy sources, mainly solar panels, and will schedule the tasks to maximize the use of green energy while minimizing the energy consumption from the grid.

The rest of this paper is organized as follows. In Chapter 2, we present an overview of the existing research on task scheduling, energy efficiency and AI in the context of fog computing. In Chapter 3, we describe the proposed task scheduling algorithm and the integration of green energy sources. In Chapter 4, we present the experimental results and the performance evaluation of the algorithm. Finally, in Chapter 5, we conclude the paper and discuss the future work.

2 Related Work

There are many existing studies and researches on the topics task scheduling, energy efficiency and AI in the context of fog computing. The concept of fog computing was introduced by Cisco in 2012 [8], and since then, many researches have been conducted to improve the performance of fog computing systems and reduce their environmental impact.

2.1 Fog computing

Fog computing is a paradigm that extends cloud computing and services to the edge of the network. [18] and [2] respectively investigate the concepts surrounding fog and its applications in the context of IoT. [16] offers an extensive review of the existing work on fog computing.

2.1.1 Task Scheduling in Fog Computing

Genetic Algorithms

The task scheduling problem in fog computing has been widely studied in the literature. Many researchers have proposed different algorithms and techniques to optimize the task scheduling process in fog computing systems. [15] offers an analysis of the existing algorithms while identifying the challenges and research gaps. In [19], the authors propose a task scheduling algorithm based on Swarm Intelligence and Machine Learning. This hybrid approach minimizes the execution time and the makespan and improves the performance of the load balancing scheduling. In [9], an optimization model is proposed for the problem of mapping data stream over fog nodes while considering the load of those nodes and the latency between the sensors and the nodes. A heuristic based on genetic algorithms is then presented to address the complexity of the problem. In [26], the authors propose a task clustering and scheduling mechanism based on Differential Evolution to find the optimal execution time for the tasks. The mechanism is compared to the Firefly Algorithm and Particle Swarm Optimization, and the results show that the proposed mechanism outperforms the other two algorithms in terms of execution time, system efficiency and stability.

Fuzzy Logic

[7] proposes a ranking-based task scheduling method that combines fuzzy logic and user preferences. In [4], the authors propose an approach based on Fuzzy Logic for real-time task scheduling in IoT applications.

2.1.2 Energy Efficiency in Fog Computing

The energy efficiency of fog computing systems has been a major concern for researchers. [3] introduces a mathematical framework to evaluate the trade-off of fog computing systems, especially in terms of power consumption and energy efficiency. In [12], the authors propose two Integer Linear Programming models, where the second one aims at minimizing the energy consumption while maximizing successfully provisioned tasks. The authors of [14] provides an overview of the energy efficiency challenges in fog computing and presents a comprehensive survey of the existing energy-efficient techniques and algorithms. An energy-aware Metaheuristic algorithm based on the Harris Hawks Optimization algorithm, itself based on a local search strategy for task scheduling in fog computing, is proposed in [1]. [22] proposes an energy efficient algorithm through an integrated computation model.

2.1.3 AI in Fog Computing

The use of AI in fog computing has been a growing trend in the literature. An efficient binary CNN with numerous skip connections is proposed by the authors of [23]. In [25], the authors designed an intelligent energy-saving model based on CNN and a task scheduling model is designed based on the policy gradient algorithm. [13] present an improved convolutional neural network so solve the value function of a Continuous Markov Decision Process model, so it can be applied to a multi-user system.

3 Problem statement

3.1 Context

3.2 Work environment

3.3 Objective

4 Internet of Things (IoT)

The Internet of Things (IoT) is a network of interconnected devices that can communicate with each other and exchange data. These devices can be anything from smartphones and laptops to sensors and actuators. The IoT is a rapidly growing field with many applications in various industries like healthcare, agriculture, transportation, and manufacturing. The goal of the IoT is to make our lives easier by automating tasks and providing real-time information. It is made possible by advances in wireless communication, sensor technology, and cloud computing. These technologies allow devices to connect to the internet and share data with each other and the cloud. The IoT has many benefits like improved efficiency, reduced costs, and increased safety. However, it also has some challenges like security, privacy, and interoperability.

4.1 History

The concept and the term "Internet of Things" can be traced back to the 1985 during a speech by Peter T. Lewis to the Congressional Black Caucus Foundation.[20] He said: "The Internet of Things, or IoT, is the integration of people, processes and technology with connectable devices and sensors to enable remote monitoring, status, manipulation and evaluation of trends of such devices." The term was later popularized by Kevin Ashton in 1999.[5]

The first internet-connected (previously known as ARPANET) device was a Coca-Cola vending machine at Carnegie Mellon University in 1982,[21] it was able to report its inventory and temperature to the network. Since then, the IoT has grown rapidly with the advent of wireless communication, sensor technology, and cloud computing.

Cisco Systems estimated that the number of internet-connected devices exceeded the number of people on Earth between 2008 and 2009,[10] marking the beginning of the IoT era. Today, there are billions of IoT devices around the world, and the number will continue to grow as more devices become connected.

4.2 Architecture

The IoT architecture consists of three layers: the perception layer, the network layer, and the application layer.

- The perception layer is where the devices are located, it includes sensors, actuators, and other devices that collect data and interact with the physical world.
- The network layer is where the devices communicate with each other and the cloud, it includes wireless communication technologies like Wi-Fi, Bluetooth, and Zigbee.
- The application layer is where the data is processed and analyzed, it includes cloud computing services like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform.

The IoT architecture is designed to be scalable, flexible, and secure. It allows devices to connect to the internet and share data with each other and the cloud. This enables new applications and services that were not possible before. The IoT architecture is still evolving with new technologies like edge computing and fog computing. These technologies bring the cloud closer to the devices and reduce latency and bandwidth requirements.

4.3 Applications

4.3.1 Home automation

One of the most popular applications of the IoT is home automation, it allows homeowners to control many aspects of their home like lighting, heating, and security. Smart devices like thermostats, light bulbs, and cameras can be connected to the internet and controlled remotely using a smartphone or voice assistant.

Home automation can help homeowners save energy, improve security, and make their lives more convenient. It is also a growing market with many companies offering smart home products and services, like Google Home, Amazon Echo, and Apple HomeKit. Non-proprietary solutions are also available, like Home Assistant, OpenHAB, and Domoticz.

4.3.2 Healthcare

IoT has many applications in healthcare, it can be used to monitor patients, track medications, and manage chronic diseases. Wearable devices like smartwatches and fitness trackers can collect data on a patient's heart rate, blood pressure, and activity level. This data can be sent to a healthcare provider for analysis and diagnosis. The IoT can also be used to monitor patients in hospitals and nursing homes, it can alert healthcare providers if a patient's condition changes. The IoT can help improve patient outcomes, reduce costs, and increase access to care.

However, it also raises concerns about privacy, security, and data ownership. The Health Insurance Portability and Accountability Act (HIPAA)[HIPAA privacy rule] in the United States and the General Data Protection Regulation (GDPR)[article 4-15] in the European Union are two regulations that govern the use of healthcare data.

4.3.3 Energy management

Homeowners can monitor and control their energy usage using with many IoT devices, this principle can be extended to a larger scale with smart grids. Smart grids are electrical grids that use IoT devices to monitor and control the flow of electricity. They can help utilities reduce costs, improve reliability, and increase efficiency. They can also help reduce greenhouse gas emissions and promote renewable energy sources. It is a growing field with many companies offering smart grid products and services, like Siemens, ABB, and Schneider Electric.

5 FOG computing

FOG computing was introduced by Cisco in 2012 as a way to extend cloud computing to the edge of the network. The goal of FOG computing is to bring the cloud closer to the end-users, this is done by placing the cloud's resources at the edge of the network thanks to a wider geographical distribution.

5.1 Concept

FOG computing is a virtualized and decentralized computing platform that acts as an intermediary between the cloud and the edge devices. The word "fog" is used to describe the cloud-like properties being closer to the "ground", the ground being the edge devices. Many of these devices will generate a huge amount or raw data that needs to be processed in real time, this is where FOG computing comes in. Instead of sending all the data to the cloud for processing, the data is processed on FOG nodes. The nodes can be anything from resource-rich servers to resource-constrained devices like gateways, routers or other end devices.

It is designed to reduce latency, provide computing resources, storage and network services to the edge devices. It also provides location awareness, mobility support, and real-time data processing. FOG computing is ideal for applications that require real-time processing, like autonomous vehicles, industrial automation, and augmented reality.

However, FOG computing is not to be confused with edge computing, while they share some similarities, they are not the same. Edge computing is a subset of FOG computing, the processing is done at the edge of the network, while FOG computing allows the processing to be done anywhere between the edge and the cloud.

5.2 Architecture

The architecture of a FOG network is shown in figure 1. It consists of three layers:

- The edge layer is where the end devices are located, it includes sensors, actuators, and other devices that generate data.
- The FOG layer is where the FOG nodes are located, it includes servers, gateways, and other devices that process data.
- The cloud layer is where the cloud data centers are located, it includes servers, storage, and network services that store and process data.

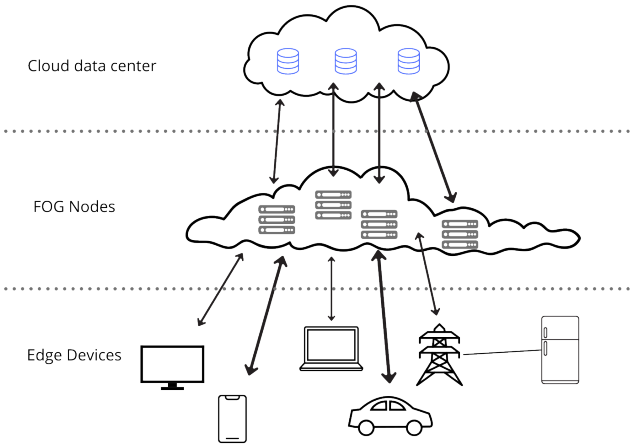

Figure 1: FOG computing architecture.

5.3 Challenges

While FOG networking has many benefits, it also has some challenges. Some of these challenges are inherited from cloud computing, while others are unique to FOG computing. We will a few of these challenges in the following sections.

5.3.1 Security and privacy

As the number of connected devices and fog nodes increases, so does the complexity of the network. This complexity makes it difficult to ensure date security and privacy. The data generated by the end devices can be sensitive and needs to be protected from unauthorized access. Also, as some fog nodes are making use of virtualized infrastructure, they introduce vulnerabilities in hypervisors and virtual machines.

5.3.2 Energy efficiency

FOG computing requires many devices to be connected to the network, this can lead to increased energy consumption. The amount of nodes and servers needed to process the data can be significant, this can lead to increased cooling and power costs. To address this issue, fog nodes can be equipped with energy-efficient hardware and software, like low-power processors and energy-aware scheduling algorithms.

5.3.3 Quality of service (QoS)

FOG computing requires real-time processing of data, this can be challenging as the network conditions can change rapidly. It requires a reliable and low-latency network to ensure that the data is processed in a timely manner. The bandwidth and latency requirements need to be met to ensure that the data is processed correctly. To address this issue, fog nodes can be equipped with QoS mechanisms that prioritize traffic based on its importance.

6 Offloading using Fuzzy logic

6.1 Explaining fuzzy logic

Introduced in 1965 by Lotfi Zadeh,[27] fuzzy logic is based on a "degree of truth" instead of a finite value, usually 0 or 1, it aims to represent the vagueness of human language and thought. Fuzzy systems are the means to implement fuzzy logic, they are two types of fuzzy systems: Mamdani and Sugeno, both are similar but differ in the way the output is determined. The most common one Mamdani and it's the one we will be using in this project. The Mamdani fuzzy system follows three steps:

- The inputs are fuzzified into fuzzy membership functions,
- a set of rules are applied to the fuzzy inputs to determine the fuzzy output,
- the fuzzy output is defuzzified to get a crisp value.

While Sugeno includes the defuzzification step in the rule evaluation step, this system works well with optimization algorithms and is more computationally efficient than the Mamdani system. But for this project we will be using the Mamdani system.

6.1.1 Fuzzification

Fuzzification is the process of converting a crisp value into a fuzzy value, this is done by assigning a membership function to the input value. The membership function is a curve that defines how much the input value belongs to a certain fuzzy set. The most common fuzzy sets are the triangular and trapezoidal functions, they are defined by three or four parameters respectively. The triangular function is defined by the parameters a, b and c and is given by:

$$\mu(x) = \begin{cases} 0 & \text{if } x \leq a, \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b, \\ \frac{c-x}{c-b} & \text{if } b \leq x \leq c, \\ 0 & \text{if } x \geq c. \end{cases} \tag{1}$$

The trapezoidal function is defined by the parameters a, b, c and d and is given by:

$$\mu(x) = \begin{cases} 0 & \text{if } x \leq a, \\ \frac{x-a}{b-a} & \text{if } a \leq x \leq b, \\ 1 & \text{if } b \leq x \leq c, \\ \frac{d-x}{d-c} & \text{if } c \leq x \leq d, \\ 0 & \text{if } x \geq d. \end{cases} \tag{2}$$

6.1.2 Rule evaluation

The rule evaluation is the process of determining the fuzzy output based on the fuzzy inputs and a set of rules. The rules are defined by two parts: the "if" or antecedent part and the "then" or consequent part. The antecedent part deals with inputs, it can either be a single input or a combination of inputs, the combination can be done using the logical operators "and" and "or". The consequent part deals with the output. In the context of this project, the rules are usually of the form "if Bandwidth is low then processing is local". The rules are generally defined by the user and are based on their knowledge of the system.

6.1.3 Aggregation

The aggregation is the process of combining the fuzzy outputs from the rules to get a single fuzzy output. This process relies on T-conorms, and must satisfy the following properties:

- Commutativity: $x * y = y * x$,
- Associativity: $x * (y * z) = (x * y) * z$,
- Monotony: $x \leq y \implies x * z \leq y * z$,
- Neutrality of 0: $x * 0 = x$ for $x \in [0, 1]$.

They are also *positive reinforcement* operators:

$$f(x_1, \cdots, x_n) \leq max[x_i] \forall x_i \geq 0.5 \tag{3}$$

As opposed to T-norms which are *negative reinforcement* operators:

$$f(x_1, \cdots, x_n) \geq min[x_i] \forall x_i \leq 0.5 \tag{4}$$

The most common T-conorms are the maximum, the probabilistic sum and the bounded sum. They are defined as follows:

$$\text{Maximum: } x \oplus y = \max(x, y) \tag{5}$$

$$\text{Probabilistic sum: } x \oplus y = x + y - x \cdot y \tag{6}$$

$$\text{Bounded sum: } x \oplus y = \min(x + y, 1) \tag{7}$$

6.1.4 Defuzzification

Defuzzification is the process of converting a fuzzy output into a crisp value, there are several methods to do this, the most common one is the centroid method. The centroid method calculates the center of mass of the fuzzy output, this is done by taking the weighted average of the output values. The weighted average is calculated by taking the sum of the product of the output value and its membership value divided by the sum of the membership values. The formula for the centroid method is given by:

$$y = \frac{\sum_i \mu_i \cdot y_i}{\sum_i \mu_i} \tag{8}$$

Where y is the crisp output, μ_i is the membership value of the output value y_i. The centroid method is the most common method because it is simple and easy to implement. However, it is not always the best method, other methods like the mean of maximum and the largest of maximum can be used depending on the application.

6.2 Fuzzy system for offloading

6.2.1 Setting up the engine

To demonstrate the use of fuzzy logic in offloading, we wrote two simple programs in Python. The goal was to recreate the experiment done by Hari et al.[11] The first program uses the *pyfuzzylite*[17] library to implement a fuzzy engine with all the variables and rules needed to determine the offloading decision. The second program uses the NumPy library to generate random values for the inputs and then uses the fuzzy engine to determine the offloading decision (remote execution or local execution, remote execution means offloading the task to the cloud or a fog node). The fuzzy engine is defined by the variables shown in table 1 and 2.

Name	Range	Fuzzy set	Membership function	Parameters
Bandwidth (in Mbps)	[0, 100]	bw_low	trapezoidal	0, 20, 30, 40
		bw_med	trapezoidal	35, 45, 60, 70
		bw_high	trapezoidal	65, 75, 90, 100
Data size (in KB)	[0, 600]	data_low	trapezoidal	0, 0, 230, 360
		data_med	trapezoidal	250, 350, 470, 590
		data_high	trapezoidal	450, 540, 600, 600
Residual battery charge (in %)	[0, 100]	bat_low	trapezoidal	0, 0, 25, 35
		bat_med	trapezoidal	25, 40, 60, 75
		bat_high	trapezoidal	60, 75, 100, 100
Load (in %)	[0, 100]	load_low	trapezoidal	0, 0, 25, 40
		load_med	trapezoidal	35, 45, 60, 70
		load_high	trapezoidal	65, 80, 100, 100
Memory (in %)	[0, 100]	mem_low	trapezoidal	0, 0, 25, 40
		mem_med	trapezoidal	35, 45, 60, 70
		mem_high	trapezoidal	65, 80, 100, 100
Virtual machines available	[0, 50]	vm_low	trapezoidal	0, 0, 15, 20
		vm_med	trapezoidal	15, 22, 37, 40
		vm_high	trapezoidal	30, 35, 50, 50
Number of concurrent users	[0, 100]	user_low	trapezoidal	0, 0, 25, 40
		user_med	trapezoidal	30, 40, 60, 70
		user_high	trapezoidal	60, 75, 100, 100

Table 1: Input variables for the fuzzy engine.

Name	Range	Fuzzy set	Membership function	Parameters
Offloading decision	[0, 100]	local	trapezoidal	0, 12, 24, 48
		remote	trapezoidal	36, 60, 72, 100

Table 2: Output variable for the fuzzy engine.

The original paper by Hari et al. did not provide all the rules used, so we had to come up with our own rules. We established them based on our understanding of the system and the variables. Unlike the original paper, where the rules only combined the bandwidth with one other variable, we decided to combine the bandwidth with all the relevant variables. The rules are shown in table 3.

	Rules
R1	IF Bandwidth is bw_low THEN Processing local_processing
R2	IF Bandwidth is not bw_low and Datasize is not data_low THEN Processing is remote_processing
R3	IF Bandwidth is not bw_low and Datasize is data_low and (Load is load_high or Memory is mem_low) THEN Processing is remote_processing
R4	IF Bandwidth is not bw_low and Datasize is data_low and (NB_concurrent_users is user_low or Memory is mem_low) THEN Processing is remote_processing
R5	IF Bandwidth is not bw_low and Datasize is data_low and NB_concurrent_users is not user_low and Memory is not mem_low and Load is load_low THEN Processing is local_processing
R6	IF Bandwidth is not bw_low and Datasize is data_low and NB_concurrent_users is not user_low and Memory is not mem_low and Load is not load_low THEN Processing is remote_processing

Table 3: Rules for the fuzzy engine.

From these rules we can see that two variables have no impact on the offloading decision, the residual battery charge and the number of virtual machines available. So we decided to remove them from the fuzzy engine.

6.2.2 Mean 3Π aggregation operator (M3Π)

The standard 3Π operator was introduced by Yager and Rybalov [24] in 1988, it is a generalization of the probabilistic sum. The 3Π operator is defined by the following formula:

$$\Pi(x_1, \cdots, x_n) = \frac{\Pi_{j=1}^{n} x_j}{\Pi_{j=1}^{n} x_j + \Pi_{j=1}^{n} (1 - x_j)} \tag{9}$$

Unlike T-norms and T-conorms, the 3Π operator is *full reinforced* which means that it satisfies the properties of both positive and negative reinforcement operators.

We decided to implement a M3Π aggregation operator which is derived from the 3Π operator, it is defined by the following formula:

$$M3\Pi(x_1, \cdots, x_n) = \frac{\Pi_{j=1}^{n} (x_j)^{(1/n)}}{\Pi_{j=1}^{n} (x_j)^{(1/n)} + \Pi_{j=1}^{n} (1 - x_j)^{(1/n)}} \tag{10}$$

The goal was to improve the decision-making process. However, we were unable to implement the M3Π operator as it would require to modify a large part of the *pyfuzzylite* library. We decided to shift our focus towards other parts of the project.

6.2.3 Partial results

Despite the absence of the M3Π operator, we still ran the fuzzy engine with three different aggregation operators: maximum, probabilistic sum and bounded sum. Samples of the results are shown in tables 4, 5 and 6.

Bandwidth	Data size	Number of concurrent users	Memory	Load	Crisp output	Fuzzy output
30	180	42	38	66	21.575	1.000/local_processing + 0.000/remote_processing
77	462	44	96	8	67.539	0.000/local_processing + 1.000/remote_processing
3	21	45	75	50	60.732	0.150/local_processing + 0.850/remote_processing
38	229	51	31	66	57.565	0.200/local_processing + 0.600/remote_processing
50	302	17	96	92	68.116	0.000/local_processing + 0.554/remote_processing

Table 4: Sample results from the fuzzy engine with the Maximum aggregation operator.

Bandwidth	Data size	Number of concurrent users	Memory	Load	Crisp output	Fuzzy output
30	180	42	38	66	21.575	1.000/local_processing + 0.000/remote_processing
77	462	44	96	8	67.539	0.000/local_processing + 1.000/remote_processing
3	21	45	75	50	60.443	0.150/local_processing + 0.850/remote_processing
38	229	51	31	66	61.27	0.200/local_processing + 0.904/remote_processing
50	302	17	96	92	68.314	0.000/local_processing + 0.863/remote_processing

Table 5: Sample results from the fuzzy engine with the Probabilistic sum aggregation operator (note: the library calls it Algebraic sum).

Bandwidth	Data size	Number of concurrent users	Memory	Load	Crisp output	Fuzzy output
30	180	42	38	66	21.575	1.000/local_processing + 0.000/remote_processing
77	462	44	96	8	67.539	0.000/local_processing + 1.000/remote_processing
3	21	45	75	50	60.275	0.150/local_processing + 0.850/remote_processing
38	229	51	31	66	61.855	0.200/local_processing + 1.000/remote_processing
50	302	17	96	92	67.903	0.000/local_processing + 1.000/remote_processing

Table 6: Sample results from the fuzzy engine with the Bounded sum aggregation operator.

For the rest of the project, we will use the Maximum aggregation operator as it gave the best results. We will also focus on improving the decision-making process by using a decision tree.

7 Decision trees

In an attempt to improve the offloading decision-making process, we decided to use the Weka[29] software to create a decision tree. We will use the J48 algorithm which is a Java implementation of the C4.5 algorithm. The goal is to was to improve upon the decision-making process of the fuzzy engine.

7.1 The Weka software

Weka is a collection of machine learning algorithms for data processing and predictive modeling. It is written in Java and is distributed under the GNU General Public License. The software is designed to be user-friendly and easy to use, thanks to its graphical user interface. It is widely used in academia and industry for data mining, machine learning, and predictive analytics. Weka provides a wide range of algorithms for classification, regression, clustering, etc. The software also provides access to SQL databases with Java Database Connectivity (JDBC) and can also accept multiple data formats like ARFF, CSV, C4.5, etc.

Figure 2: Weka logo.

Figure 3: Weka software icon.

7.1.1 The J48 algorithm

The J48 algorithm is a decision tree algorithm, it allows the creation of a decision tree from a set of training data. The algorithm works by recursively splitting the data into subsets based on the values of the input variables. The goal is to create a tree that can predict the value of the output variable based on the input variables. The algorithm uses the information gain to determine the best split at each node, the information gain is a measure of the reduction in entropy.

The training data is a set of already classified data, it contains the values of the input variables and the corresponding value of the output variable and the class it belongs to. The algorithm uses this data to build the decision tree, at each node, it selects the attribute that split the data into the best subsets. The process is repeated until all the data is classified. The decision tree can then be used to predict the value of the output variable for new data.

Decision trees are used to represent the decision-making process in a graphical form, they are easy to understand and interpret. They can also be used to identify the most important variables in the data and to detect patterns and relationships between the variables.

7.1.2 The decision tree

The resulting decision tree is composed of nodes, leaves and branches:

- The nodes represent the attributes where the data was split,
- the branches represent the value of the attribute,
- the leaves represent the class that the data belongs to.

When building the tree, the missing values are ignored but can be predicted by using the attribute value that is most common in the training data. The tree can be pruned to improve its accuracy and reduce its complexity. Pruning is the process of removing nodes that do not improve the accuracy of the tree. The decision tree can be visualized using the graphical user interface of Weka. The tree can also be exported as a text file for further analysis.

Tree pruning

To avoid overfitting, the decision tree can be pruned. Pruning is the process of removing nodes that do not improve the accuracy of the tree. There are two types of pruning: pre-pruning and post-pruning. Pre-pruning is done before the tree is built, it stops the tree from growing when a certain condition is met. Post-pruning is done after the tree is built, it removes nodes that do not improve the accuracy of the tree. Pruning can improve the accuracy of the tree and reduce its complexity.

7.2 Results

7.2.1 Generating the decision tree

For this test, we used the data we generated with the fuzzy engine. These data are in the form of a CSV file with a header and 21 rows. They contain the values of the input variables and the corresponding value of the output variable in both crisp and fuzzy form. The data was loaded into Weka and the J48 algorithm was applied to it. The resulting decision tree is shown in figure 4.

7.2.2 Updating the fuzzy engine

We then used this decision tree to modify the membership functions of our input and output variables, we also modified the input variables not appearing in the decision tree to match the other variables. The functions were changed to triangular, the original brackets for each variable were kept. The middle value for **bw_med** and **user_med** was changed according to the decision tree, for the other variables we took the average of the brackets. The same process was done for the output variable, the new values are shown in tables 7 and 8.

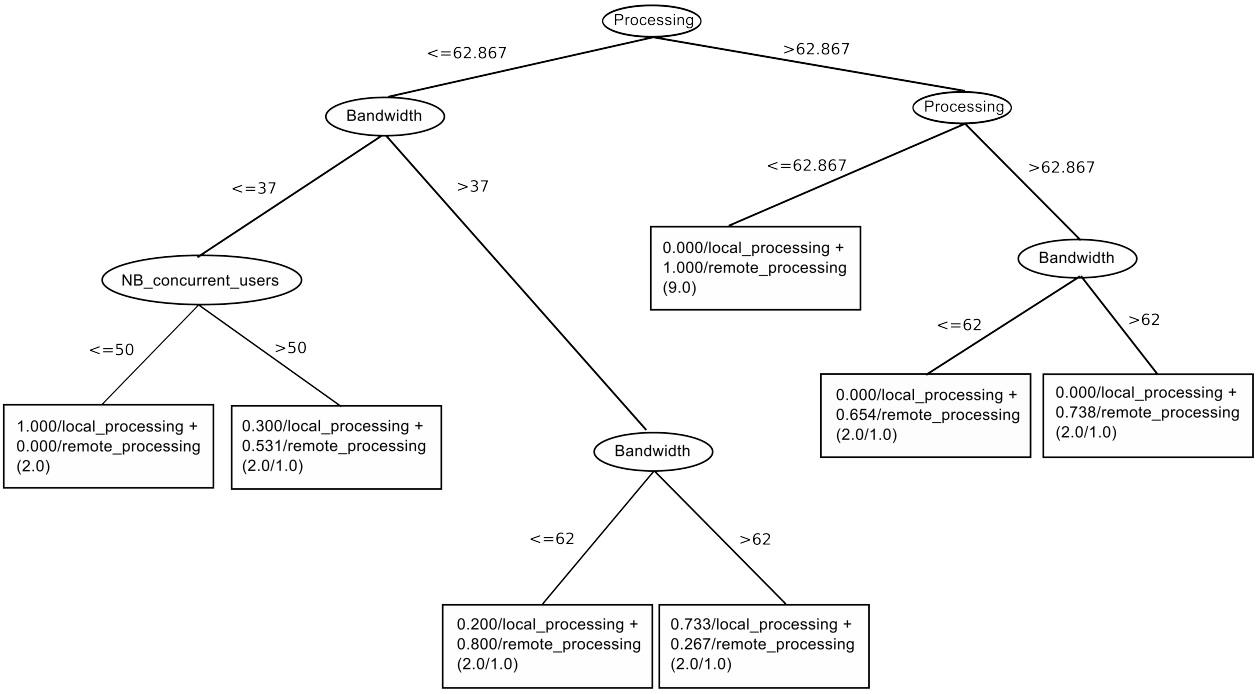

Figure 4: Decision tree generated by the J48 algorithm. (Recreated with Inkscape)

Name	Range	Fuzzy set	Membership function	Parameters
Bandwidth (in Mbps)	[0, 100]	bw_low	triangular	0, 20, 40
		bw_med	triangular	35, 37, 70
		bw_high	triangular	65, 82, 100
Data size (in KB)	[0, 600]	data_low	triangular	0, 180, 360
		data_med	triangular	250, 420, 590
		data_high	triangular	450, 600, 600
Load (in %)	[0, 100]	load_low	triangular	0, 20, 40
		load_med	triangular	35, 52, 70
		load_high	triangular	65, 82, 100
Memory (in %)	[0, 100]	mem_low	triangular	0, 20, 40
		mem_med	triangular	35, 52, 70
		mem_high	triangular	65, 82, 100
Number of concurrent users	[0, 100]	user_low	triangular	0, 20, 40
		user_med	triangular	30, 50, 70
		user_high	triangular	60, 80, 100

Table 7: Input variables for the fuzzy engine.

Name	Range	Fuzzy set	Membership function	Parameters
Offloading decision	[0, 100]	local	triangular	0, 24, 48
		remote	triangular	36, 62.867, 100

Table 8: Output variable for the fuzzy engine.

When defining the membership functions, we usually want them to be symmetrical but in this case, we decided to keep the values as they were in the decision tree. This resulted in **Bandwidth** and **Processing** not having symmetrical membership functions, **Data size** is also in this case but for a different reason and is not as noticeable. We can use the graphic interface of QtFuzzyLite to visualize the new membership functions and compare them to the original ones. The results are shown in figures 5, 6, 7 and 8.

7.2.3 Generating new results with the updated fuzzy engine

After updating the fuzzy engine, we made a script capable of running both the old and new fuzzy engines with the same input data. We then compared the results to see if the decision tree improved the accuracy of the fuzzy engine. The results are shown in tables 9 and 10.

Bandwidth	Data size	Number of concurrent users	Memory	Load	Crisp output	Fuzzy output
43	505	44	92	69	67.228	0.000/local_processing + 1.000/remote_processing
20	285	87	35	93	21.6	1.000/local_processing + 0.000/remote_processing
83	193	71	71	36	56.42	0.267/local_processing + 0.733/remote_processing
45	353	9	77	72	67.251	0.000/local_processing + 0.946/remote_processing
70	61	93	64	17	21.6	1.000/local_processing + 0.000/remote_processing
28	418	60	35	22	21.6	1.000/local_processing + 0.000/remote_processing
60	345	31	53	15	62.239	0.115/local_processing + 0.885/remote_processing
37	400	77	71	18	55.221	0.300/local_processing + 0.700/remote_processing

Table 9: Sample results from the old fuzzy engine.

Bandwidth	Data size	Number of concurrent users	Memory	Load	Crisp output	Fuzzy output
43	505	44	92	69	66.289	0.000/local_processing + 1.000/remote_processing
20	285	87	35	93	24.0	1.000/local_processing + 0.000/remote_processing
83	193	71	71	36	57.64	0.200/local_processing + 0.800/remote_processing
45	353	9	77	72	66.294	0.000/local_processing + 0.961/remote_processing
70	61	93	64	17	53.206	0.339/local_processing + 0.661/remote_processing
28	418	60	35	22	45.829	0.600/local_processing + 0.400/remote_processing
60	345	31	53	15	62.223	0.083/local_processing + 0.917/remote_processing
37	400	77	71	18	59.483	0.150/local_processing + 0.850/remote_processing

Table 10: Sample results from the new fuzzy engine.

We can see that, while some results are nearly identical, others are quite different. The new engine either reinforced the decision made by the old engine or changed it. Overall, the new engine seems to yield better and more consistent results. This is worth expending upon in the future by using more data and refining the decision tree. However, we lack the time to do so in the context of this project.

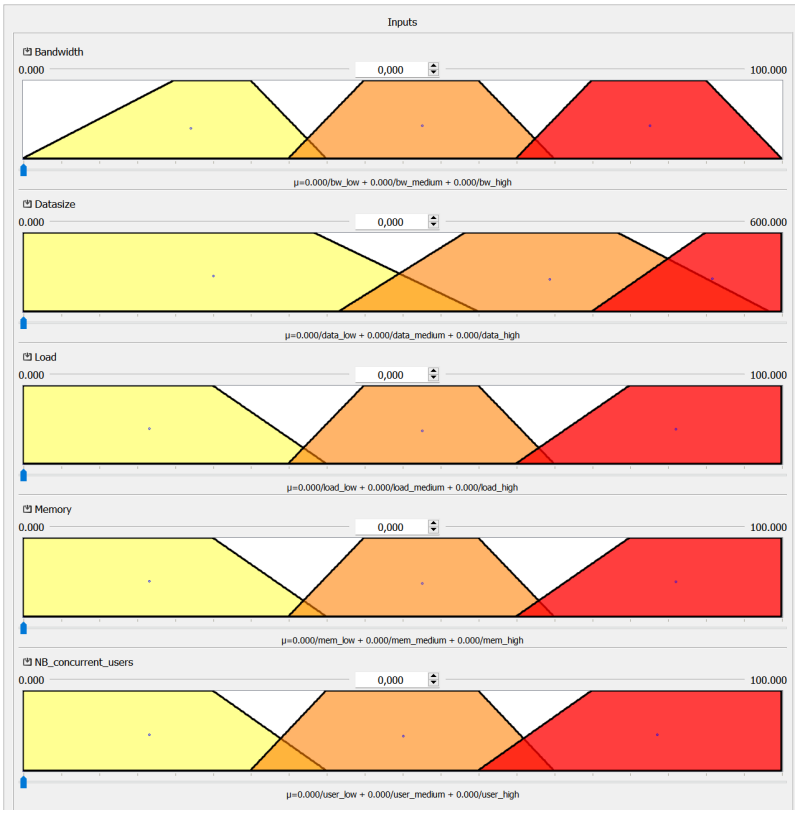

Figure 5: Input variables for the old fuzzy engine.

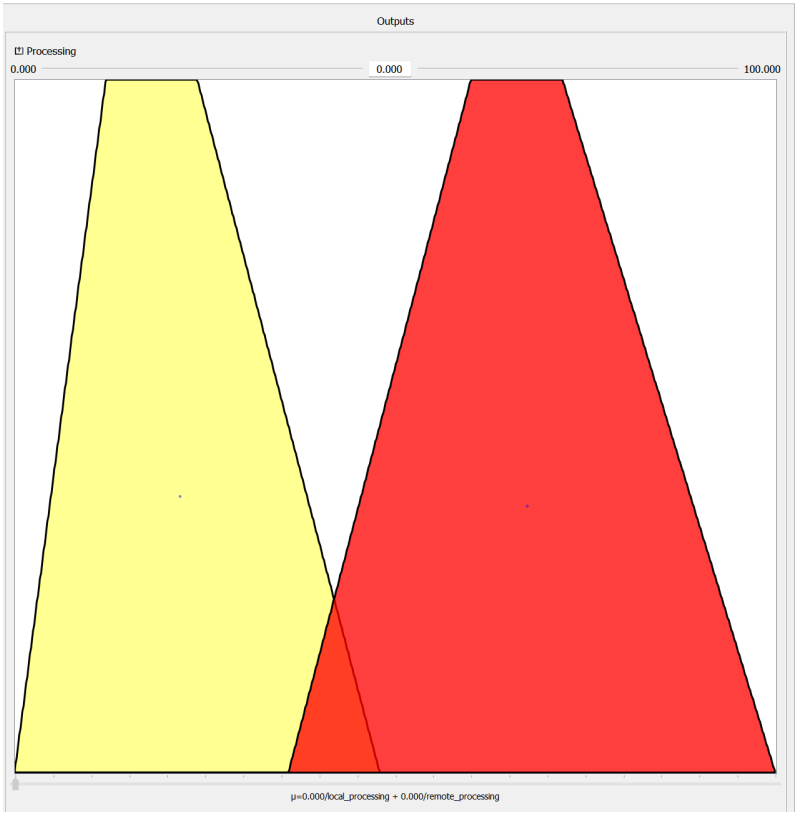

Figure 6: Output variable for the old fuzzy engine.

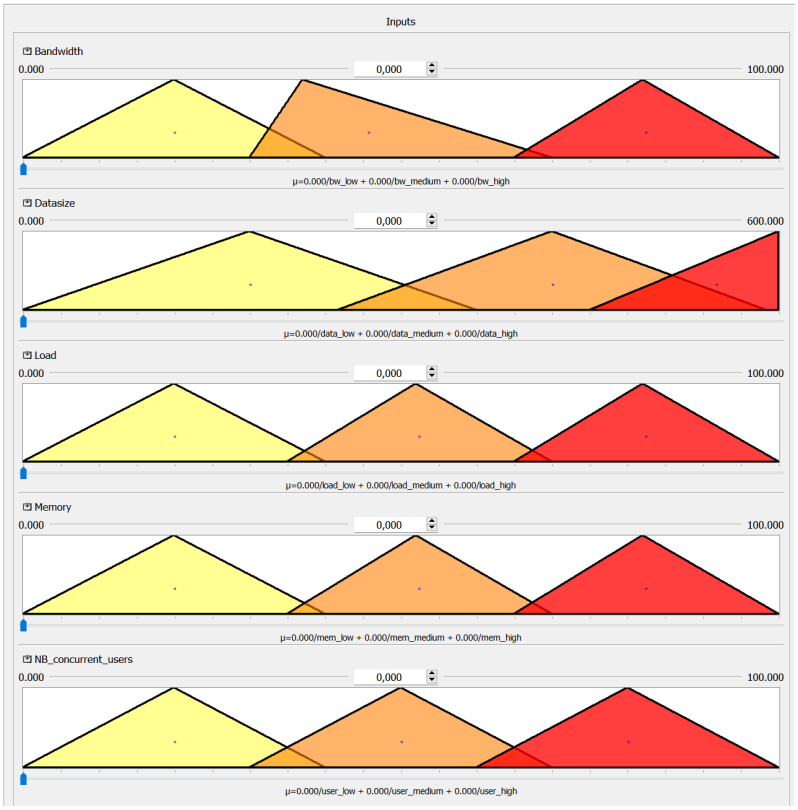

Figure 7: Input variables for the new fuzzy engine.

Figure 8: Output variable for the new fuzzy engine.

8 Future work

The original goal of this project was to create a task scheduler for a fog computing environment, improve the energy efficiency while maintaining the quality of service and considering the use green energy sources. We did not manage to create a task scheduler, but we do have leads on how to proceed.

8.1 Task scheduler

As the name implies, the task scheduler is responsible for assigning tasks to the available resources. The scheduler must take into account the state of the resources, the requirements of the tasks and the constraints of the system. It must also be able to adapt to the changing conditions and make decisions in real-time.

8.1.1 The machine learning approach

The most promising approach is to use machine learning algorithms. Based on the works mentioned in section 2.1.3, we can use Convolutional Neural Networks (CNN). CNNs are a type of neural network that is well suited for image recognition tasks. They are composed of multiple layers of neurons that process the input data in a hierarchical manner. The input data is passed through a series of convolutional layers, pooling layers and fully connected layers. The output is a vector of probabilities that represent the likelihood of the input data belonging to a certain class. The architecture of a CNN is shown in figure 9.

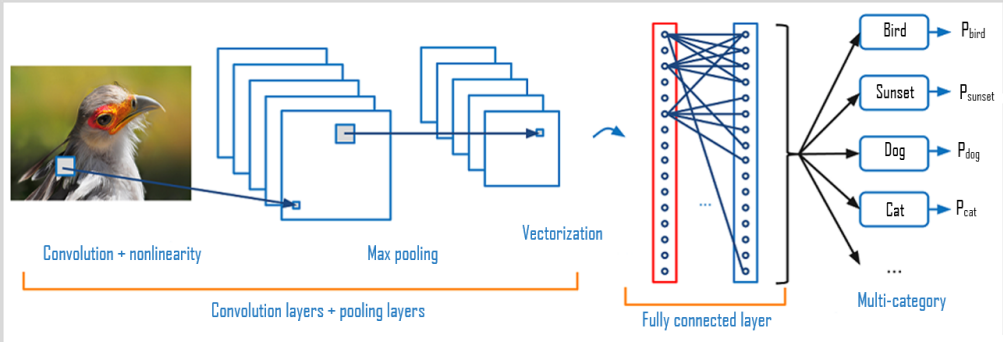

Figure 9: Convolutional Neural Network architecture.[6]

The way we would use CNNs is by training them on pre-existing data. The data would contain the state of the resources, the requirements of the tasks, the constraints of the system and an established schedule like the one shown in figure 10. The CNN would then learn the patterns and relationships between the variables and be able to predict the best schedule for new data. It would also need to take into account the energy sources used to power the resources and the cost of using them.

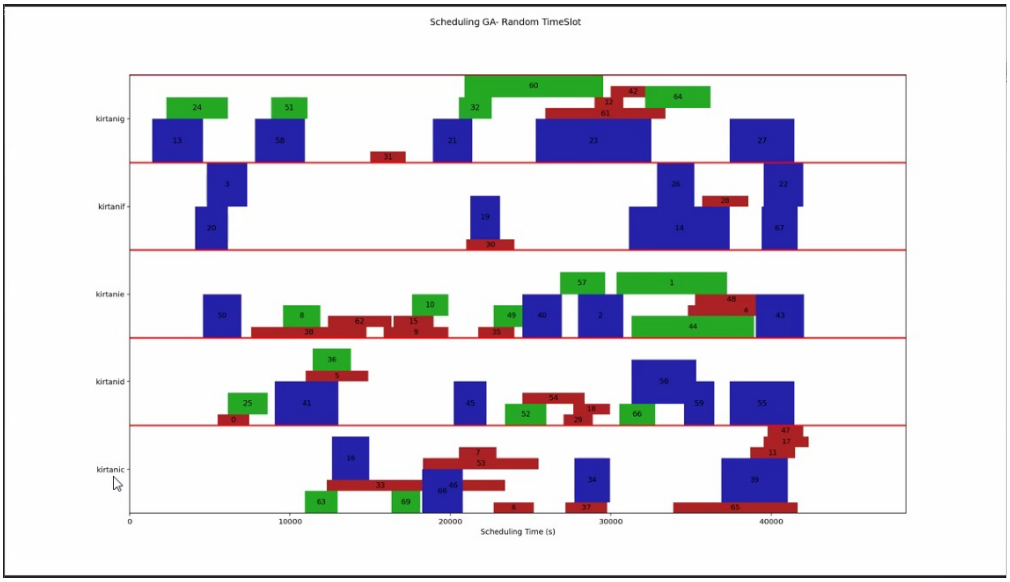

Figure 10: Example of a schedule of tasks. Each color represents the amount of CPU cores used by the task, red is 1 core, green is 2 cores and blue is 4 cores.

After the fuzzy engine offloads the tasks to the cloud or the fog nodes, the CNN would then assign the tasks to the available resources. It will optimize the schedule to minimize the energy consumption, execution time and cost, while maximizing the amount of tasks executed and the green energy used.

9 Conclusion

Conclusion

Bibliography

[1] Mohamed Abdel-Basset et al. "Energy-Aware Metaheuristic Algorithm for Industrial-Internet-of-Things Task Scheduling Problems in Fog Computing Applications". In: *IEEE Internet of Things Journal* 8.16 (2021), pp. 12638–12649. DOI: 10.1109/JIOT.2020.3012617.

[2] Nirase Abubacker, Muhammad Ehsan Rana, and Mafas Raheem. "Fog Computing Applications". In: June 2023, pp. 30–58. ISBN: 9781668444665. DOI: 10.4018/978-1-6684-4466-5.ch003.

[3] Raad S. Alhumaima. "Energy efficiency and latency analysis of fog networks". In: *China Communications* 17.4 (2020), pp. 66–77. DOI: 10.23919/JCC.2020.04.007.

[4] Hala S. Ali et al. "Real-Time Task Scheduling in Fog-Cloud Computing Framework for IoT Applications: A Fuzzy Logic based Approach". In: *2021 International Conference on COMmunication Systems & NETworkS (COMSNETS)*. 2021, pp. 556–564. DOI: 10.1109/COMSNETS51098.2021.9352931.

[5] Kevin Ashton. *That 'Internet of Things' Thing - RFID Journal*. https://web.archive.org/web/20130415194522/http://www.rfidjournal.com/articles/view?4986. June 2009.

[6] Nada Belaidi. *Réseaux convolutifs (CNN) : comment ça marche ?* June 2022. URL: https://blent.ai/blog/a/cnn-comment-ca-marche.

[7] Mohammed Anis Benblidia et al. "Ranking Fog nodes for Tasks Scheduling in Fog-Cloud Environments: A Fuzzy Logic Approach". In: *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. 2019, pp. 1451–1457. DOI: 10.1109/IWCMC.2019.8766437.

[8] Flavio Bonomi et al. "Fog computing and its role in the internet of things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. Helsinki, Finland: Association for Computing Machinery, 2012, pp. 13–16. ISBN: 9781450315197. DOI: 10.1145/2342509.2342513. URL: https://doi.org/10.1145/2342509.2342513.

[9] Claudia Canali and Riccardo Lancellotti. "GASP: Genetic Algorithms for Service Placement in Fog Computing Systems". In: *Algorithms* 12.10 (2019). ISSN: 1999-4893. DOI: 10.3390/a12100201. URL: https://www.mdpi.com/1999-4893/12/10/201.

[10] Dave Evans. *The Internet of Things, How the Next Evolution of the Internet Is Changing Everything*. https://www.cisco.com/c/dam/en_us/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf. Apr. 2011.

[11] Dhanya Hari et al. "Fuzzy-logic-based decision engine for offloading IoT application using fog computing". In: May 2018, pp. 175–194. DOI: 10.4018/978-1-5225-5972-6.ch009.

[12] Zhiming He et al. "Green Fog Planning for Optimal Internet-of-Thing Task Scheduling". In: *IEEE Access* 8 (2020), pp. 1224–1234. DOI: 10.1109/ACCESS.2019.2961952.

[13] Bing Jing and Huimin Xue. "IoT Fog Computing Optimization Method Based on Improved Convolutional Neural Network". In: *IEEE Access* 12 (2024), pp. 2398–2408. DOI: 10.1109/ACCESS.2023.3348133.

[14] Usman Mahmood Malik et al. "Energy-Efficient Fog Computing for 6G-Enabled Massive IoT: Recent Trends and Future Opportunities". In: *IEEE Internet of Things Journal* 9.16 (2022), pp. 14572–14594. DOI: 10.1109/JIOT.2021.3068056.

[15] Javid Misirli and Emiliano Casalicchio. "An Analysis of Methods and Metrics for Task Scheduling in Fog Computing". In: *Future Internet* 16.1 (2024). ISSN: 1999-5903. DOI: 10.3390/fi16010016. URL: https://www.mdpi.com/1999-5903/16/1/16.

[16] Mohammed Al-Musawi, Dunya Alrseetmiwe, and Zahraa Saad. "Fog computing system for internet of things: Survey". In: 16 (Nov. 2023), 1_10.

[17] Juan Rada-Vilela. *The FuzzyLite Libraries for Fuzzy Logic Control*. 2018. URL: https://fuzzylite.com.

[18] Muhammad Ehsan Rana and Nirase Abubacker. "Fog Computing Foundations". In: June 2023, pp. 1–20. ISBN: 9781668444665. DOI: 10.4018/978-1-6684-4466-5.ch001.

[19] Gaith Rjoub and Jamal Bentahar. "Cloud Task Scheduling Based on Swarm Intelligence and Machine Learning". In: *2017 IEEE 5th International Conference on Future Internet of Things and Cloud (FiCloud)*. 2017, pp. 272–279. DOI: 10.1109/FiCloud.2017.52.

[20] CHETAN SHARMA. *Correcting the IoT History*. https://www.chetansharma.com/correcting-the-iot-history/. Mar. 2016.

[21] Carnegie Mellon University. *The "Only" Coke Machine on the Internet*. https://www.cs.cmu.edu/~coke/history_long.txt.

[22] Yan Wang et al. "Service delay and optimization of the energy efficiency of a system in fog-enabled smart cities". In: *Alexandria Engineering Journal* 84 (2023), pp. 112–125. ISSN: 1110-0168. DOI: https://doi.org/10.1016/j.aej.2023.10.034. URL: https://www.sciencedirect.com/science/article/pii/S1110016823009365.

[23] Lijun Wu et al. "An Efficient Binary Convolutional Neural Network With Numerous Skip Connections for Fog Computing". In: *IEEE Internet of Things Journal* 8.14 (2021), pp. 11357–11367. DOI: 10.1109/JIOT.2021.3052105.

[24] R.R. Yager and A. Rybalov. "Full reinforcement operators in aggregation techniques". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 28.6 (1998), pp. 757–769. DOI: 10.1109/3477.735386.

[25] Dexian Yang et al. "Energy saving strategy of cloud data computing based on convolutional neural network and policy gradient algorithm". In: *PLOS ONE* 17.12 (Dec. 2022), pp. 1–18. DOI: 10.1371/journal.pone.0279649. URL: https://doi.org/10.1371/journal.pone.0279649.

[26] Adil Yousif, Mohammed Bakri Bashir, and Awad Ali. "An Evolutionary Algorithm for Task Clustering and Scheduling in IoT Edge Computing". In: *Mathematics* 12.2 (2024). ISSN: 2227-7390. DOI: 10.3390/math12020281. URL: https://www.mdpi.com/2227-7390/12/2/281.

[27] L.A. Zadeh. "Fuzzy sets". In: *Information and Control* 8.3 (1965), pp. 338–353. ISSN: 0019-9958. DOI: https://doi.org/10.1016/S0019-9958(65)90241-X. URL: https://www.sciencedirect.com/science/article/pii/S001999586590241X.

Data Sources

[28] Overheid Andrew. *Understanding Fog Computing vs Edge Computing*. https://www.onlogic.com/eu/blog/fog-computing-vs-edge-computing/. 2021.

[29] Eibe Frank, Mark A. Hall, and Ian H. Witten. *The WEKA Workbench*. Fourth. Morgan Kaufmann, 2016. URL: https://ml.cms.waikato.ac.nz/weka/.

[30] *Fuzzy logic - Wikipedia*. https://en.wikipedia.org/wiki/Fuzzy_logic.

[31] Satyabrata Jena. *Difference Between Edge Computing and Fog Computing*. https://www.geeksforgeeks.org/difference-between-edge-computing-and-fog-computing/. 2022.

[32] Nilima Khanna. *J48 Classification (C4.5 Algorithm) in a Nutshell*. https://medium.com/@nilimakhanna1/j48-classification-c4-5-algorithm-in-a-nutshell-24c50d20658e. Aug. 2021.

[33] Quang Trung Luu. *What Is Fog Computing?* https://www.baeldung.com/cs/fog-computing. 2024.

[34] *Weka (software) - Wikipedia*. https://en.wikipedia.org/wiki/Weka_(software).