Journal of Internet Services and Applications, 2024, 15:1, doi: 10.5753/jisa.2024.4026

Resource Allocation Based on Task Priority and Resource Consumption in Edge Computing

Guilherme Alves Araújo [**Universidade Federal do Ceará** | ***guialves@alu.ufc.br***]
Sandy Ferreira da Costa Bezerra [**Universidade Federal do Ceará** | ***sandycosta@alu.ufc.br***]
Atslands Rego da Rocha [**Universidade Federal do Ceará** | ***atslands@ufc.br***]

Departamento de Engenharia em Teleinformática, Universidade Federal do Ceará (UFC), Campus do Pici, Bloco 725, 60455-970, Fortaleza, CE, Brasil.

Received: 12 January 2024 • **Accepted:** 29 May 2024 • **Published:** 16 September 2024

Abstract The computational power of Internet of Things (IoT) devices is usually low, which makes it necessary to process data and extract relevant information on devices with higher processing capacity. Edge Computing emerged as a complementary solution to cloud computing, providing devices at the network edge with computational resources to handle the data processing and analysis that constrained IoT devices eventually cannot perform. This solution allows data processing closer to the IoT devices, reducing latency for IoT applications. However, the resource constraints of edge nodes, which have lower computational power than the cloud nodes, make resource allocation and processing massive requests challenging. This study proposes an edge resource allocation mechanism based on task priority and machine learning. The proposed approach efficiently allocates resources for IoT requests based on their task priorities while monitoring the resource consumption of edge nodes. This study evaluates the performance of different classification algorithms by using well-known metrics for classifying models. The most efficient classifier achieved an accuracy of 92% and a precision of 90%. The results indicate good performance when using this classifier in the evaluated approach. The proposed mechanism demonstrated that resource management can be done more efficiently with significantly lower resource utilization when compared to an allocation method based only on distance. The study tested different scenarios regarding the number of requests, edge nodes, and a proposed failure mechanism to schedule failed node tasks to functional nodes. This failure control mechanism is a significant contribution of the proposal. Therefore, the proposed method in this study can become a valuable tool for efficient resource management with reduced computational cost and efficient resource allocation.

Keywords: Edge Computing, Resource Management, Resource Allocation, Classification Models, Machine Learning, Task Priority.

1 Introduction

The Internet of Things (IoT) paradigm has been widely adopted in various practical applications, such as smart cities [Rejeb *et al.*, 2022]. This paradigm is enabled through the interconnection and interoperability of physical and virtual entities, enabling the creation of intelligent services and informed decision-making for monitoring, control, and management purposes [Tran-Dang and Kim, 2018]. Combining edge and cloud computing provides a system offering continuous services with varying QoS requirements to end-users.

Edge Computing emerges as an innovative paradigm that aims to address the demands of IoT applications with massive access and critical latency, bringing processing capacity closer to the end-users. This paradigm has been widely applied in IoT environments, representing a decentralized infrastructure composed of processing nodes strategically distributed between end devices and the cloud. This approach allows a more agile and efficient process, transforming the edge into an extension of the cloud [Martinez *et al.*, 2021].

The essence of Edge Computing architecture consists of multiple interconnected layers. Generally, IoT devices are at the bottom layer, responsible for collecting, pre-processing, and forwarding data to the edge nodes. The middle layer comprises the edge nodes with higher processing capacity than the IoT devices. These nodes perform data processing, decision-making, and action-taking functions. Finally, the top layer, represented by the cloud, is responsible for storing permanent information and processing data that the edge nodes do not have enough computing power to perform [Arena and Pau, 2020].

Unlike cloud nodes, computing resources (such as processing, storage, and energy) at edge nodes are limited and dynamic, resulting in constantly changing workloads and applications competing for these restricted resources. Given this situation, effectively managing restricted computing resources at endpoints becomes a considerable challenge to maximize their use optimally, as highlighted by Sharif *et al.* [2022].

Resource management has become a significant focus in Edge Computing in recent years. This area is crucial because it ensures that the resources present in the network are allocated efficiently at the network's edge and nodes can perform their activities efficiently. Proper management of resources is crucial to prevent some nodes from consuming more resources than necessary while others may have insufficient resources. This situation could lead to performance and operational issues.

Resource allocation involves carefully selecting an edge node, which can provide the resources needed to perform a given task or request from an end device [Liu *et al.*, 2020]. However, this process can be challenging due to limited resources. Furthermore, due to the sharing of resources between IoT devices, unpredictable events, unavailability of resources on network nodes, and extended response times are possible [Dlamini and Ventura, 2019]. Efficient resource allocation is a crucial starting point for this solution, aimed at avoiding excessive consumption of node resources and ensuring more effective management [Wang *et al.*, 2022].

However, as IoT applications have different needs (such as sensitivity or tolerance to delays), it is interesting that the efficient allocation of resources considers their requirements. A potential approach is prioritizing application requests and allocating edge resources to execute tasks based on their relevance and urgency. This will be particularly useful for practical applications such as digital health, industrial systems, and intelligent manufacturing.

In digital health systems, prioritizing real-time requests from connected medical devices is important. This helps to ensure that critical data, such as vital signs, is processed and sent with high priority to improve emergency medical decision-making. In the context of Industry 4.0, for industrial systems, prioritizing sensor and device requests in innovative manufacturing environments can ensure that critical demands, such as production monitoring and safety parameters, are treated with priority to avoid downtime or failures in the production process.

Researchers and practitioners have extensively explored machine learning techniques to solve this resource management problem and automate increasingly different solutions to managing resources at the edge, as surveyed in Hussain *et al.* [2020]. Using machine learning in resource management provides a more intelligent and adaptive approach, improves operational efficiency, and allows computer systems to adapt more effectively to new demands. The benefits include automatic optimization, anomaly detection, and autonomous decision-making.

In this context, this work presents a mechanism for allocating resources in edge computing environments. The proposed mechanism is based on continuously monitoring the resources available on edge nodes and uses a task priority classifier for task/request scheduling. This approach enables efficient management of these resources and simultaneously selects the most appropriate edge node to handle the requirements demanded by the applications, considering the resources needed to run the requested services. This mechanism aims to optimize the edge computing environment's performance allowing tasks to be processed quickly and effectively, according to their specific needs and efficiently using node resources. This work provides important contributions:

- A classification mechanism based on machine learning that prioritizes IoT device tasks according to their relevance and urgency of execution.
- A set of rules and weights for prioritizing tasks that consider various factors, such as the type of request, latency, type of connection, request time, distance, and available resources, providing more appropriate and personalized treatment for each task.
- A mechanism for allocating the resources of edge nodes, taking into account different criteria such as node capacity and distance, which aims to guarantee appropriate use of these resources, and also considering the priority of tasks and monitoring the use of edge node resources.
- A mechanism for controlling node failures.

This paper extends our previous work [Araújo *et al.*, 2023], providing the differential aspects:

- A failure control mechanism to bypass potential failures on edge nodes.
- Adjustment of the weight computation formula of the classification model, which now also considers the elapsed time of the request.
- The resource allocation mechanism considers a score to assign the most appropriate node to receive the request. To determine the resource allocation, the mechanism now considers the distance from the node and the percentage of the edge node's resource usage in real time.
- Experiments comparing different simulation scenarios to validate the proposal.
- Experiments with more complex simulation scenarios: greater variety of request variations and edge nodes.
- The classifier uses a data set with more features to define priority and a more significant amount of data, thus benefiting the training and validation of the Machine Learning Model.
- Performance evaluation including other edge node resources (RAM).

The remainder of the paper is organized as follows. Section 2 discusses related work and the differences between them and our proposal. In Section 3, we describe the proposed resource management mechanism for the edge environment, from receiving the request by the classifier to deciding where to allocate these requests. Section 4 presents the proposed failure control mechanism and how it works. Section 5 introduces the methodology and proposal implementation. Section 6 describes the simulation experiments conducted and their design. In Section 7, we describe the results of the experiments to evaluate the proposal. Finally, Section 8 presents some conclusions and outlines future research directions.

2 Related Work

Many efforts from the academic community are being directed towards resource management in Edge Computing. This is because the inefficient allocation of resources available at the network's edge significantly impacts performance improvements. As a result, various algorithmic solutions and methodologies have been proposed, and considerable progress has already been made in this area.

In the work Tran-Dang and Kim [2021], the authors present TPRA (*Task Priority-based Resource Allocation*), a resource allocation algorithm for edge computing based on the priority of tasks. This study divides tasks into two categories: standard and priority. Resource providers can reject standard tasks, assuming they might encounter delays if left unattended. Resource providers must execute priority-categorized tasks successfully and cannot reject them. The system queues standard tasks and calculates the delay duration until standard tasks transition into a priority status. Once they reach this status, they require immediate attention without the option of rejection.

The study Bhushan and Mat [2021] proposes a priority queue-based edge computing architecture in which edge nodes are dynamically allocated based on system load. Task allocation is performed considering priority, which is defined based on service provisioning and divides tasks into two classes: High priority (delay sensitive) and low priority (delay tolerant). The work also assumes a parameter, which represents the average proportion of the number of high-priority tasks relative to the total number of tasks, to calculate the average delay time for each class.

The work Madej *et al.* [2020] presents different scaling techniques. The first is a *FCFS* strategy, in which tasks are scheduled in the order they are received. Furthermore, the paper proposes three additional strategies: customer-oriented, priority-oriented, and a hybrid approach that considers customers and task priorities. The customer-oriented strategy treats all customers as having the same priority without considering individual task priorities. The priority-oriented approach considers the tasks' priorities independently of the customers. Finally, the hybrid strategy considers customers and task priorities, ensuring an equitable and efficient allocation of edge resources. These scheduling techniques proposed in the paper aim to deal with the issue of limited resource availability at the edge, ensuring fairness between clients and considering task priorities.

To achieve effective resource allocation and optimal use of resources, the work Sharif *et al.* [2023] presents an adaptive resource allocation mechanism for Edge Computing. This mechanism aims to dynamically allocate available resources considering the nature of the requests received to obtain optimal utilization. In the scheme proposed by the authors, the mechanism adapts to the demands and priorities of the requests received. After identifying a request, priority is set based on the delay sensitivity of the task, which is checked during the identification process. The available resources are then allocated according to the priorities of the requests to meet the established restrictions.

The article Bui *et al.* [2021] introduces an enhancement to the "Score Based Match-Making" algorithm aiming to optimize task distribution on edge nodes and service quality in resource allocation. It proposes an algorithm utilizing a score based on six factors (node distance, network condition, latency, memory, CPU capacity, and node task execution history) to address challenges related to resource allocation priority. Additionally, the algorithm incorporates a preemption factor to handle emergencies. The preemption factor allows temporarily interrupting a running task if a higher-priority task enters the queue; the interrupted task is later resumed. This approach is implemented through an orchestration platform across various scenarios. The proposed algorithm is compared with two other allocation types: random and a "naive" approach similar to "first come, first serve." The best node for task execution is chosen by comparing the scores of available nodes.

In Yin *et al.* [2020], the optimization method focuses on finding the minimum average delay point for task completion to optimize and allocate system resources. The authors use a lower bound based on a quadratic function to determine the minimum amount of resources required to process each task. Additionally, the model implements a scheduling function that considers three criteria: delay time, transfer time, and resource consumption. This scheduling function calculates the necessary processing time and the minimum resources required for each task type.

Table 1. Related works.

Study	Failure Control	Criterion for Task Prioritization
Tran-Dang and Kim [2021]	No	Delay Time
Bhushan and Mat [2021]	No	Delay Sensitivity Waiting Time
Madej *et al.* [2020]	No	Distance Memory Processor
Sharif *et al.* [2023]	No	Time Constraint
Bui *et al.* [2021]	No	Distance Latency Network Condition Memory Processor Connection History
Yin *et al.* [2020]	No	Delay Time Transfer Time Minimum Resource
Our work	**Yes**	**Service Type** **Resources** **Latency** **Connection Type** **Time** **Distance**

In summary, as shown in Table 1, research proposes to use criteria to prioritize tasks and balance the use of available resources. These approaches consider the tasks and resources of edge nodes as essential prioritization criteria. However, our proposal stands out from others in that it takes into account not only the tasks/requests but also the limitations of resources available at the edge nodes. It also considers important factors like the distance from the node, the latency, the type of connection, and the time elapsed since the request of the task. Furthermore, failure control is an essential contribution of our approach, ensuring that all requests are fulfilled regardless of edge node failures. Our work aims to manage resources on edge nodes, classify their priorities, and determine the best node for allocation. This approach allows for a balanced use of edge resources to meet the needs of requesting services.

3 Resource Management Based on Task Priorities and Edge Node Resource Consumption

The proposed resource management mechanism uses machine learning to classify and prioritize end device (IoT) requests in edge computing environments.

Figure 1 shows the architecture of our proposed mechanism. IoT applications send their requests to a classifier node. This node receives the requests and classifies them into priority levels using machine learning. After classification, tasks are performed based on a defined task priority for the chosen edge node. The resource consumption of edge nodes is monitored during execution.

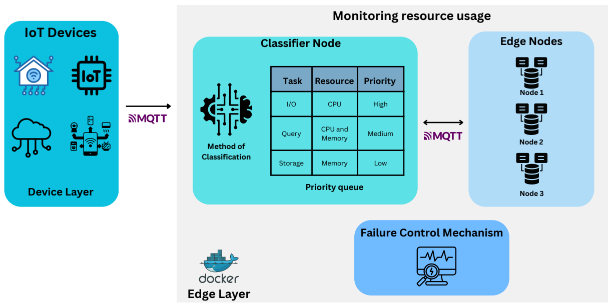


Figure 1. Architecture of the Proposed Mechanism.

Figure 1 shows four essential elements (IoT Devices, Classifier Node, Failure Control Mechanism, and Edge Nodes) of our proposal, divided into two layers: device and edge. The Devices layer is where the requests come from, the so-called end devices closest to the users, such as IoT devices, sensors, and actuators. The Edge layer contains three elements (Classifier Node, Failure Control Mechanism, and Edge Nodes). In the classifier node, incoming requests are classified based on rules and weights established later. A priority queue is then generated and allocated to the edge node. Edge nodes process the requests and provide the resources for the entire application. Finally, there is the failure control element, which stores the status of the requests allocated to the edge nodes in case one of these nodes fails, making it possible to get around this situation. It is important to note that the Edge layer continuously monitors the use of edge node resources. The architecture is detailed in the following subsections.

3.1 Requests

Requests coming from IoT devices encompass several tasks, such as activation, storage, and processing. In the context of these requests, we consider the following assumptions:

1. Each request has a specific type of task; therefore, the types of services were added to the data set (Storage, Processing, Query, and I/O);
2. Requests come from devices that may have different types of network connections;
3. Request latency changes depending on the type of network used.

Following these premises, we adopted the principle that each request requires a specific type of service. A request is defined as a tuple $R_i = (TR_i, L_i, RN_i, H_i, D_i, TC_i)$, where:

- TR_i represents the type of request i.
- L_i indicates the latency of request i.
- RN_i refers to the resource needed to fulfill request i.
- H_i is the time at which request i is fired.
- D_i represents the distance of the devices about the request classifier i.
- TC_i represents the connection type of request i.

These variables constitute the inputs of the classification models used. Furthermore, we incorporate weights to balance the priorities of different tasks, ensuring a weighted approach in the analysis and processing of requests (detailed in Section 3.2).

3.2 Classifier

Classification is a machine learning technique that assigns one or more classes to a data set based on their characteristics or variables. This classification can be binary (two classes, 1 or 0) or multiclass (three or more). This technique is used to identify patterns and trends in data, allowing automated decision-making in different contexts [Scikit-Learn, 2023]. This work used multiclass classification (three classes for priority levels).

We implemented three machine learning classifiers: kNN (k-Nearest Neighbors), SVM (Support Vector Machine), and Logistic Regression to compare and select the most suitable model for classifying the mechanism.

A classifier and an allocation model were implemented to classify application requests coming from end devices into different priority levels (classes), thus allowing an allocation of resources based on established priorities. IoT devices send their service requests (data) to the Classifier Node.

The data used for training and validating the classifiers were extracted from *Kaggle*[1]. This data set has valuable characteristics, such as latency, connection type, and network rate, and has been enriched with other information such as weight, distance based on latency to simulate requests from IoT devices with a more significant number of variables. This information includes the type of service, resources, time, latency, network type, rate, and geographic coordinates, which are used to calculate the Euclidean distance to choose the most appropriate nodes for a given request in addition to the failure control mechanism.

This data is sent to the classifier node, where different weights are applied to resources, and priorities are assigned to each type of request based on predefined rules. The priority rules are defined as follows. Initially, request priorities

[1]https://www.kaggle.com/datasets/suraj520/cellular-network-analysis-dataset

were classified into three levels: High (2), Medium (1), and Low (0). To establish request priority rules, two types of services were categorized into critical services (SC) and non-critical services (SNC), as presented below.

- **Critical Service**
  - I/O (Input/Output)
  - Real-time
- **Non-critical Service**
  - Query
  - Storage

Services in the critical class are assigned High priority (2), while services in the non-critical class are classified as either Medium priority (1) or Low priority (0).

According to studies Cheng *et al.* [2015] and Wang *et al.* [2019], the I/O service is considered critical due to its direct interaction with the end user, significantly impacting the overall user experience. Therefore, the need for an I/O service application is an essential factor to be considered, suggesting priority over other resources. Similarly, the criteria for real-time services are also based on the principle used for I/O services, as these services also influence the user experience and require fast responses. On the contrary, query services (which only provide information to the user) and data storage services (where data is stored locally at the edge or later sent to the Cloud) are considered non-critical. Requests that require the use of the Cloud are generally services that require less immediate response or involve more complex processing, considering that the Cloud is generally distant from end devices [Hong and Varghese, 2019].

In addition to this distinction based on the criticality of the tasks, the request time and the weights defined for the resources necessary to meet each request are also considered. This approach of adding weight to resource priorities aims to achieve an adequate balance between the priorities of different types of services and the availability of resources at the edge, allowing efficient system operation, as such resources are limited.

In this sense, services that demand more resources, such as RAM and CPU, have their priorities adjusted (receive higher weights) compared to services that only require CPU. This decision is made considering that priority services requiring multiple resources can cause extensive queues and lead to system unavailability. At the same time, other priority services that use only one computing resource can be served more quickly. Furthermore, the complexity of requests can increase energy consumption, contributing to services' unavailability. Therefore, it is necessary to appropriately assign weights to resources to deal with these different situations. This aspect considers the real-time availability of resources at Edge nodes.

Therefore, the rule for assigning resource weights is as follows and is described in Equation 1:

$$Req_{weight} = (0.3 * Q_R) + (N * T_R) \qquad (1)$$

The Q_R value (representing resource requirements) plays a crucial role in determining the priority weight (Req_{weight}) of service requests. Requests with higher Q_R values (Q_R = 2) can make a larger contribution to Req_{weight}, indicating higher priority for single-resource requests. For requests with multiple resources (Q_R = 1), the contribution to Req_{weight} is still significant but may be lower than single-resource requests. This prioritization scheme balances the efficiency of single-resource requests with the needs of multi-resource requests while also considering service type (N) and resource availability.

The constant value '0.3' can be interpreted as a base weight assigned to all service requests. This suggests that even the most fundamental requests have a certain level of priority in the system. This base weight ensures that all requests receive some level of consideration, preventing them from being entirely overlooked. serves to offset the impact of variables like resource requirements (QR) and service type (N), thereby crafting a more refined prioritization framework. This value was determined through experimentation, resulting in the most favorable outcomes.

The N value in Equation 1 is an addition variable that takes on different values depending on the type of service requested. Non-critical services (SNC) have either 0.5 or 0.7 values, while critical services (SC) have either 0.1 or 0.3 values. The value of N changes based on the time that has elapsed since the service was requested. The higher the resulting value of Equation 1, the greater the priority of the service request.

The value of T_R depends on the type of connection, which can vary between LTE, 3G, 4G, and 5G. The higher the transmission rate, the lower the latency tends to be. Requests with a network type with a lower transmission rate receive a higher value of T_R. These values are set to 0.3 (LTE), 0.2 (3G), and 0.1 (4G and 5G). This approach aims to prioritize requests from slower connections, giving them a more significant weight (in terms of priority) to compensate for latency.

This way, a high weight is assigned to requests with higher priority to compensate for latency. In this way, the different characteristics of services and connections are considered, enabling better performance.

The values of the weights used were determined empirically through simulations. This practical approach has proven satisfactory in balancing elapsed time, resource quantity, service types, and connection types. Through weight assignment, it is possible to adjust the priority of requests so that critical services receive the necessary attention.

It is essential to highlight that the weights may vary according to the context and specific needs of the addressed problem. Conducting simulations and adjusting the weights based on observed results is a valid and commonly adopted approach in practice. This medium allows for adapting the model to the particular characteristics and constraints of the system, aiming to maximize request performance and efficiency.

3.3 Resource Management with Allocation

In the resource allocation mechanism proposed in this work, the priority classification technique for client requests described in the previous section is used. By establishing priorities for requests, the system can direct resources more effectively, ensuring immediate response to the most relevant and urgent tasks. This medium helps to improve network performance, saving available resources.

To determine the most suitable edge node to meet the request, in addition to real-time monitoring of resource use, the distance of the edge nodes about the classifier is considered. In this study, Equation 2 is used to calculate the Euclidean Distance between the edge node and the classifier node, considering a two-dimensional plane:

$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2}$$

The choice of Euclidean distance is based on the fact that when you want to minimize latency, the physical distance between IoT devices and edge processing nodes is crucial. Using Euclidean distance as a metric allows us to allocate resources to edge nodes in a way that optimizes this distance, resulting in lower communication latency and better overall IoT system performance. Furthermore, Euclidean distance is a common and widely accepted metric in the context of IoT data science. It is often used in load balancing strategies, where edge computing resources are distributed to balance the processing load across different geographic regions, taking into account the spatial distribution of IoT devices.

When using a criterion that combines the resource utilization percentage at the edge node and the distance about the classifier node, a score [Costa *et al.*, 2020] is assigned to select the node. The higher the score, the greater the chance of choosing the node. Equation 3 is used to calculate the score:

$$Score_i = (\alpha * Resources_{node}\%) + (1 - \alpha) * Distance_{node}) \tag{3}$$

We conducted empirical analysis with different values for α and found that 0.3 provided the best results in terms of performance and accuracy. The values 0.3 and 0.7 represent, respectively, that 30% of the final score is determined by the node's resources and 70% by the distance from the node to the destination. This intuitive division facilitates the understanding of the equation and its impact on the selection of the ideal node.

In this way, the most suitable node for executing a given priority task is selected to choose the most capable and closest node, aiming to minimize latency. Equation 3 was tested empirically until satisfactory results were obtained, reaching the values in question.

In Figure 2, a flowchart represents the complete process from the request to the task allocation on the edge nodes.

4 Failure Control Mechanism

A crucial and significant point in this work is failure control. Controlling failures in computing environments can include redundancy, continuous monitoring, network resilience, self-healing, and regular maintenance. These procedures aim to guarantee the availability and reliability of services, detect failures early, guarantee connectivity and recovery capacity in the event of interruption, and keep systems updated and functioning correctly.

This work implemented a control and failure management mechanism based on continuous monitoring and redundancy

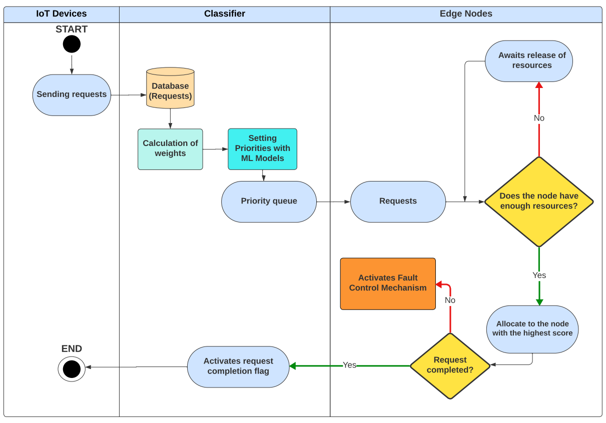

IoT Devices
Classifier
Edge Nodes
START
Sending requests
Database (Requests)
Calculation of weights
Setting Priorities with ML Models
Priority queue
Requests
Awaits release of resources
No
Does the node have enough resources?
Yes
Allocate to the node with the highest score
Activates Fault Control Mechanism
No
Request completed?
Yes
Activates request completion flag
END

Figure 2. Flowchart: Process from the request to the task allocation on the edge nodes.

to process requests at edge nodes efficiently. When setting priorities, a *Thread* is assigned to calculate the score of each edge node in order to determine which one will be chosen for allocation. Once the selection is made, the requests are allocated to the corresponding nodes.

This *Thread* assumes the responsibility of monitoring the requests allocated at a given time and keeping them in a *checkpoint* state, maintaining a copy of this data until they are entirely resolved. This *checkpoint* state is crucial for failure control if any edge node presents problems. Thread monitors edge node states continuously and identifies whether a given node is inactive. Suppose this situation is detected based on the information stored in the *checkpoint* state. In that case, *Thread* can reallocate the requests that were on the failed node, as long as they have not been completed, to other edge nodes that have sufficient resources to complete them. The state of the *checkpoint* is then updated with the new information, allowing the system to adapt and continue processing requests even in the event of a failure.

Figure 3 illustrates how the failure control mechanism works. In scenario I, observing the requests allocated to the edge nodes is possible. At the same time, the *checkpoint* state stores this information to ensure that it is not lost until its complete execution. In scenario II, node 1 fails, and the *Thread* takes control to reallocate the requests initially assigned to this node to other edge nodes in the network. After this relocation, the *checkpoint* state is updated with the new information.

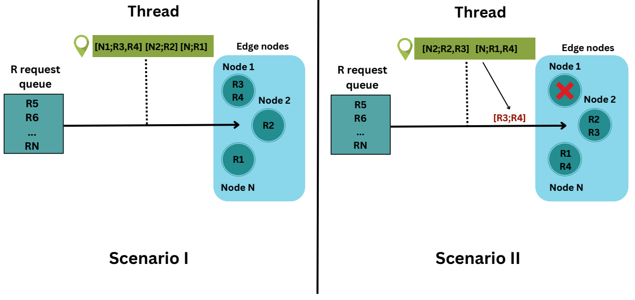


Figure 3. Fault control flow.

These two scenarios illustrate how the fault-checking and management mechanism works in action. The checkpoint state plays a crucial role in controlling allocated requests and enabling recovery in the event of an edge node failure. With this approach, the system can guarantee continuous operation, even in the face of failures, and ensure that all requests are processed correctly. In the event of a node failure, the system can recover by maintaining the states of these nodes for a finite period.

5 Methodology

This section describes the architecture of the resource management mechanism proposed in this section and its components, in addition to the prioritization criteria and the choice of nodes.

5.1 Implementation of the Proposed Mechanism

We use Docker containerization to implement the architecture presented in this work. The utilization of Docker containers played a fundamental role in this study, enabling an edge environment where each edge node operates independently within its container. This approach offers several advantages, such as facilitating the migration of applications between devices with varying hardware configurations. Moreover, Docker containers provide complete isolation of operating systems, enabling multiple edge nodes to run simultaneously on the same machine. This medium ensures that the resources of each node are isolated and utilized independently, even during the execution of tasks on other nodes. Using the 'docker stats' API allows information about the resource usage of each edge node (container). This information is leveraged to assign requests to edge nodes, enabling a more dynamic allocation of resources based on application needs, defined priorities, and available resources.

The following steps were taken to classify priorities. After conducting a series of tests with several classifiers, evaluating their performance metrics, such as accuracy, precision, and possible occurrence of *overfitting*, three models were chosen: kNN, SVM, and Logistic Regression. These models are widely studied and have been consistently successful in several studies, guaranteeing satisfactory results, according to Arowolo *et al*. [2022], Aqib *et al*. [2022] and Rusman *et al*. [2019].

Communication between the edge nodes and the classifier is established using the lightweight Message Queuing Telemetry Transport (MQTT) protocol, designed for device-to-device communication on low-power, bandwidth-constrained networks. MQTT is widely used in IoT environments due to its simplicity, efficiency, and ability to support asynchronous and bidirectional communication between IoT devices [Sasaki and Yokotani, 2019].

MQTT works on a publish-subscribe model [Bayılmış *et al*., 2022]. Devices can post messages on specific topics, while other devices can subscribe to those topics and receive messages specific to that topic. Each edge node is assigned a specific topic in a resource allocation scenario. Requests allocated to an edge node are sent to that particular topic, allowing the edge node to subscribe to it and receive requests directed to it.

We chose to use a distance-based technique as a benchmark for comparison due to the nature of validating the concept of our work. Likewise, we selected a technique highly relevant to the context of IoT devices, where the physical distance between devices and edge processing nodes plays a crucial role in minimizing latency.

It is important to note that our proposal is also distance-based, but we consider other factors such as resource consumption and request time. Our goal is to validate the effectiveness of an approach that incorporates these various factors, aiming to achieve a more robust and efficient strategy.

6 Experiments

We used a physical machine with a Linux operating system equipped with an i7-7560U processor and 8 GB of RAM for the simulations. The machine was connected to a local network via Wi-Fi, and containers were used to implement the edge computing architecture.

We developed a *script* in *Python* in version 3.10.12 to populate the dataset, adding information for simulating requests from IoT devices. The classifier node that received these requests is responsible for assigning and classifying the priorities of each request. These requests contained information about the type of services, required resources, latency, distance from nodes, elapsed time of the request, type of connection, and network transmission rate.

We followed the flow described below in the experiments, varying the number of task requests to the edge nodes:

1. The request/task data is sent to the classifier node.
2. This data serves as input for the classifier to calculate the priorities for each request.
3. Monitoring of the resources of the edge nodes occurs and is sent to the manager node that contains the classifier. This way, the priority table is adjusted according to available resources.
4. The allocation of requests is made based on the priority queue, the resources these tasks require, and the resources available on the edge nodes at that time, in addition to the Euclidean distance of the nodes about the classifier node.
5. A run-time check occurs on the state of the containers so that if there is a failure (state *offline*), the requests can be relocated to other nodes so as not to compromise the execution of the tasks.

To run the experiments, the simulation starts based on all the configurations defined in the Docker Compose file, such as containers, the number of edge nodes, and the type of communication between them, among other configurations.

The experiments carried out allow evaluate the presented classification models (kNN, SVM, and Logistic Regression) and select the one that obtains the best results as the priority classification model. Furthermore, they allow evaluation of resource management efficiency using the proposed mechanism, allocating requests to edge devices based on established task priority rules and evaluating resource consumption by these nodes.

6.1 Experimental Design

Six simulated experiments (A, B, C, D, E, and F) were performed using different scenarios. Experiment A evaluates the classification models (kNN, SVM, and Logistic Regression) presented for the priority classifier node based on the evaluation metrics: Confusion Matrix, Accuracy, F1-score, Precision, and Recall. The model that obtained the best classification results was selected.

The initial configuration of nodes with 256 megabits of memory, aims for a more faithful relationship with real IoT devices. The use of different percentages of available CPU during execution is a crucial point for realism. It is significant to recognize that computers often run other processes in parallel, impacting the availability of resources for each application. The heterogeneity of nodes in terms of CPU capacity is a fundamental aspect of simulating edge devices. IoT environments often feature devices with varied computational resources, and considering this heterogeneity in testing is essential for evaluating the real performance of distributed solutions.

In Experiment B, requests were allocated to edge devices based on predefined rules, using a classifier to determine priorities. Three edge nodes processed requests ranging from 500 to 1000, varying the workload. Next, we evaluated the CPU consumption by the edge nodes, analyzed the behavior of the resource management mechanism compared to an approach where tasks are assigned by selecting the nearest node to the classifier (based on distance), and monitored the resource consumption by edge nodes in both environments.

In Experiment C, six edge nodes were used, with the number of requests received varying between 500 and 1000. A comparison was carried out in an environment in which the proposed architecture was adopted and in another environment that did not incorporate the mechanism proposed in this study. The requests were allocated exclusively based on the latency criterion, selecting the node closest to the classifier.

Experiments D and E followed the same scenarios as experiments B (3 nodes and edges and requests varying between 500 and 1000) and C (6 nodes and edges and requests varying between 500 and 1000), respectively. The objective of these experiments is to evaluate memory consumption with different numbers of edge nodes in each experiment.

In Experiment F, a failure is forced to simulate the shutdown of an edge node, aiming to demonstrate the process of action of the proposed failure mechanism in this situation. Therefore, this experiment evaluates how the system behaves in the event of a node failure, analyzing resource consumption at the edge nodes. This assessment allows understanding how the system reacts and whether it can adapt and redistribute tasks efficiently, ensuring service continuity even with the temporary unavailability of one or more edge nodes.

Initially, we opted for this scenario as an initial proof of concept, aiming to investigate the effects of specific node manipulation. The results obtained were satisfactory in achieving the initial objectives of the study. However, we recognize the importance of expanding these tests to more complex and realistic scenarios in the future, in order to further validate our findings and assess the scalability of the system under more diverse conditions.

In the experiments carried out, strategies were implemented to balance the use of resources. When a resource monitored by an edge node reaches 85% utilization, the edge node stops receiving new requests until the resource is free. Additionally, a check of the condition of the edge nodes is performed.

We summarize the main objectives of the experiments carried out:

1. Evaluate the implemented classification models.
2. Compare the proposal with the traditional approach of choosing the nearest node.
3. Evaluate the proposal about the adaptive use of resources, considering different quantities of requests.
4. Verify the behavior of the proposal in case of failure at any edge node.

Each experiment addressed these objectives in different settings to comprehensively assess the proposal's performance and effectiveness. Table 2 summarizes each experiment's objectives, scenarios, and metrics.

7 Results and discussions

In this section, we present the results of our study, which focuses on analyzing various classification models such as kNN, SVM, and Logistic Regression, determining the most suitable algorithm for allocating requests to edge nodes. We evaluate the results using validation metrics for classification algorithms. Furthermore, we implement and analyze the model that provides the best CPU and RAM consumption results for varying numbers of edge nodes and requests. The study also compares the proposed allocation method with a distance allocation approach, where requests are sent to the nearest edge nodes. We highlight the differences between the two methods.

7.1 Experiment A

The results of the experiments confirm the effectiveness of the implemented classifier in determining the priority and allocation of resources to edge nodes. The attributes obtained from the R_i requests, such as latitude, longitude, network, transmission rate, resources, priority, and weights, play a crucial role in the classification model's outcomes. Initially, the relationship between these variables is studied, and a correlation graph of the variables is displayed in Figure 4. This graph shows the relationship between different variables in a data set, identifying possible linear relationships and evaluating the strength and direction of these relationships. Correlation charts help identify relationships between variables and provide insights into their dependence or independence within the data set.

Based on Figure 4, we have identified some correlations between variables. The most significant relationships are found in network, rate, and latency cases. A better network connection has a positive impact on throughput and reduces

Table 2. Summary of Conducted Simulations.

Experiment	Scenario	Objectives	Evaluation
A	—	Evaluate classification models	Validation metrics of the models
B	I and II - 3 nodes: 500 requests III and IV - 3 nodes: 1000 requests (I and III use the mechanism)	Compare the mechanism proposed with allocation by distance	Resource consumption of nodes (CPU)
C	I and II - 6 nodes: 500 requests III and IV - 6 nodes: 1000 requests (I and III use the mechanism)	Compare the mechanism proposed with allocation by distance	Resource consumption of nodes (CPU)
D	I and II - 3 nodes: 500 requests III and IV - 3 nodes: 1000 requests (I and III use the mechanism)	Compare the mechanism proposed with allocation by distance	Resource consumption of nodes (Memory)
E	I and II - 6 nodes: 500 requests III and IV - 6 nodes: 1000 requests (I and III use the mechanism)	Compare the mechanism proposed with allocation by distance	Resource consumption of nodes (Memory)
F	6 nodes: 1000 requests	Evaluate the failure mechanism	Reallocation of requests

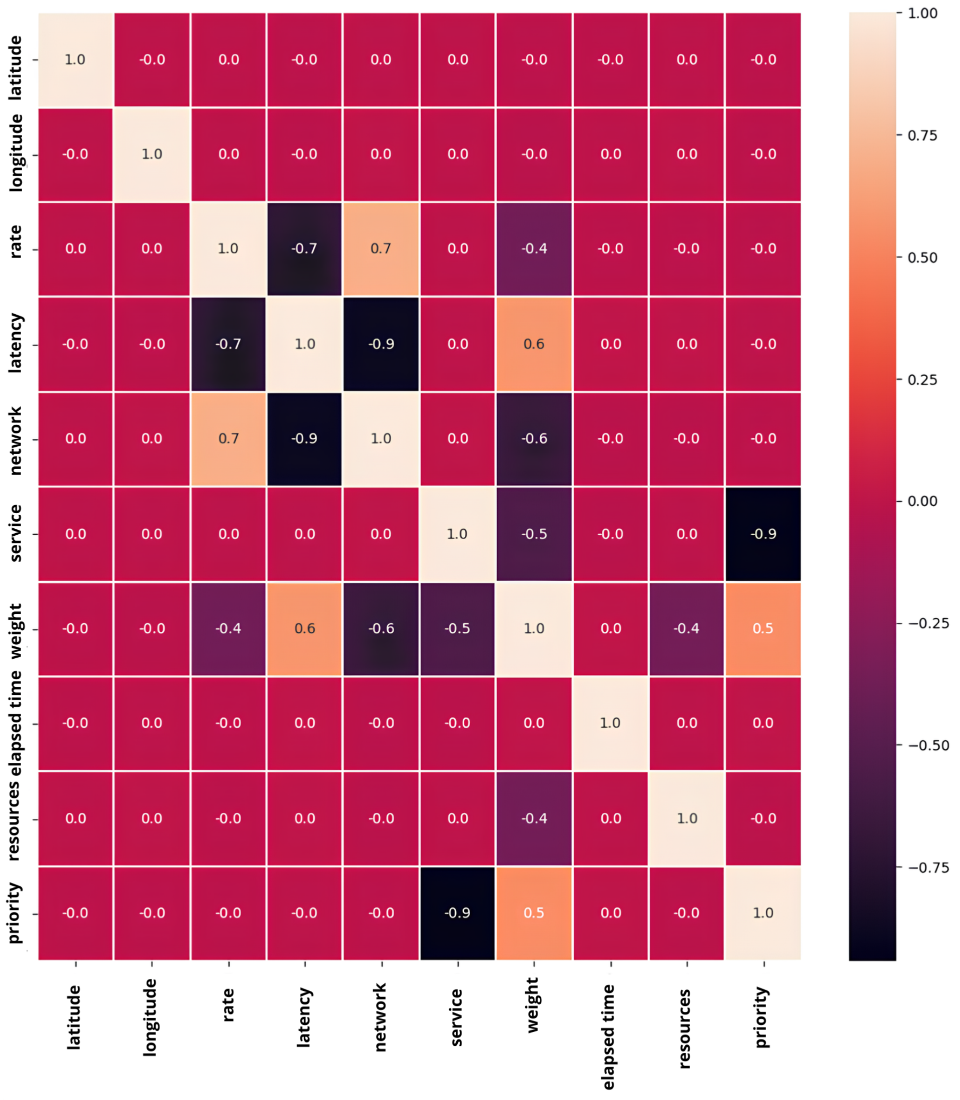


Figure 4. Correlation of dataset variables

latency. In addition, there is a strong correlation between the service and priority variables. This suggests that any changes made to the service would also affect the priority.

The chart is useful in identifying outliers, which are points that significantly differ from the primary trend in the data. These points may represent exceptional cases that require more detailed analysis. However, it is crucial to consider that these conclusions are drawn based on a visual analysis of the correlation graph. A correlation coefficient, such as the Pearson coefficient, can be calculated to obtain a more accurate and statistically based analysis. This quantifies the intensity of the relationships between variables and determines the statistical significance of these correlations.

In this experiment, we inserted the R_i requests from the devices into the classifier. The dataset we used had specific and above-mentioned characteristics. We divided the dataset into training, testing, and validation sets, allocating 70% of the data for training and 30% for testing. We used a set of 16829 requests for validation, corresponding to the dataset's total size. The classifier was trained to correctly classify the priority of each type of service based on factors such as latency, type of network, location, and necessary resources. We applied the previously established rules to the classifier to achieve this. Finally, we present the results for each implemented classification model.

7.1.1 Confusion Matrices

We generated confusion matrices for each model to validate the results and for comparison purposes, as depicted in Figure 5.

Below, we provide a comprehensive analysis of each model's matrix and the information that can be derived from these results.

7.1.2 SVM - Support Vector Machine

The SVM, Figure 5(a), presented inferior results to the other two classification models (Logistic regression and kNN).

Some critical information was revealed during the analysis of the SVM classification model's performance. The confusion matrix allowed for a clear visualization of hit and error distributions for each class. It was observed that the model struggled to classify classes with a priority set to "Medium" and could not accurately classify this category. However, the classes with "High" and "Low" priorities were more accurately classified, although there were still a significant number of errors for the "Low" class.

The analysis of evaluation metrics also brought relevant *insights*. The accuracy of approximately 73% showed that the model was correct in approximately half of the predictions but made errors in approximately 44%. This medium indicates that the model is not achieving a desirable level of accuracy in its predictions.

The values obtained for precision indicate that of the times the model made an optimistic prediction, it was correct in around 63% of cases, a relatively low value. In contrast, a recall of 53% indicates that the model is ineffective. Furthermore, the average value of the F1-score reaching 56% showed that the model managed to find a moderate balance between precision and recall, being able to identify positive examples and limit false positives correctly, but still far from satisfactory results for the problem in question.

The priority classification model using SVM has some strengths but presents significant challenges. Although the model balanced precision and recall, its predictions must be more accurate as its performance is not uniform across all classes. Precision suggests that the model is making positive decisions with some confidence, but recall indicates missing a significant proportion of positive examples.

The SVM model proved less effective than other models, such as kNN and Logistic Regression. Inaccurate classification lowered evaluation metrics values and negatively affected the overall model performance.

7.1.3 Logistic Regression

Logistic Regression, Figure 5(b), outperformed SVM, with an accuracy of approximately 89%.

The results indicate that the classification system performs well, with acceptable levels of accuracy and reliability. Upon reviewing Figure 5, it is clear that the Logistic Regression model outperforms the SVM method in terms of accuracy. The Logistic Regression model more accurately categorized priorities, leading to a more balanced distribution between correct and incorrect classifications for each category in the confusion matrix.

Based on the confusion matrix analysis, it is evident that the Logistic Regression model had relatively fewer correct predictions in the High and Low-priority categories. However, it performed exceptionally well in the Medium priority classification, surpassing the performance of the SVM model.

By examining the confusion matrix and taking into account the values of True Positives (VP), True Negatives (VN), False Positives (FP), and False Negatives (FN), we can calculate the accuracy rate, which reached 89%. This suggests that the model performs well in classifying categories correctly and minimizing errors.

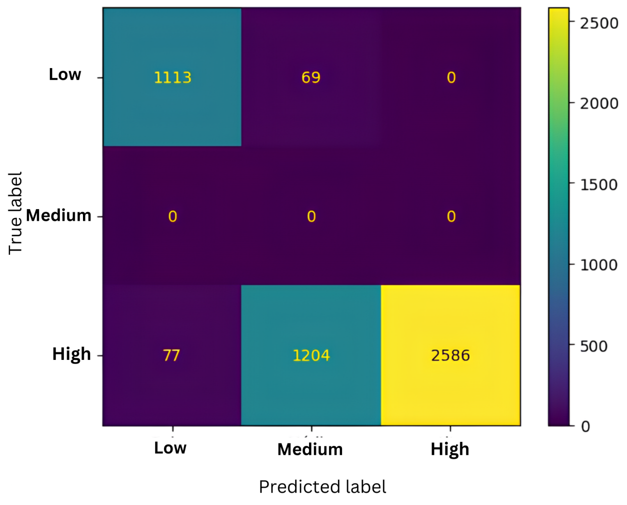
As for the accuracy of the model, which shows the number of correct optimistic predictions, it recorded around 84%. This format indicates that the model issues positive decisions with a reasonable degree of confidence and makes errors only in a few cases. With an average F1 score of 78%, the model achieves a satisfactory balance between precision and recall.

The recall, which was 84%, indicates that the model can identify the majority of positive examples and minimizes the cases in which it misses examples that it should have identified. The model operates consistently with frequent successes and controlled errors.

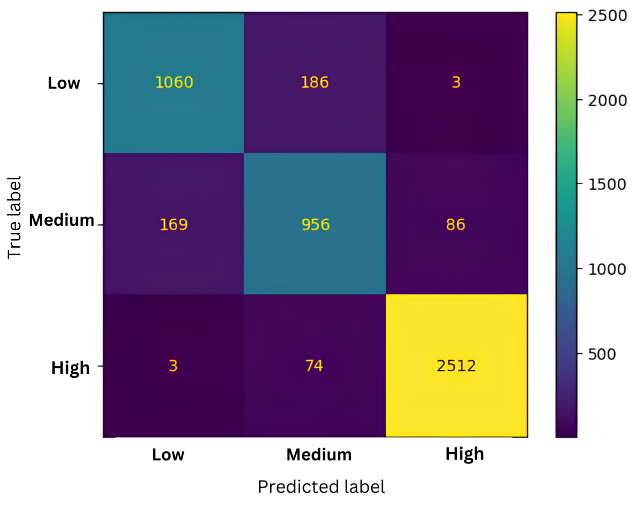
The priority classification model using logistic regression performs well and has promising results. It could be the primary classifier for the entire architecture. However, the kNN model performs even better, with an accuracy exceeding 90%. You can see the results below.

7.1.4 kNN - K-Nearest Neighbors

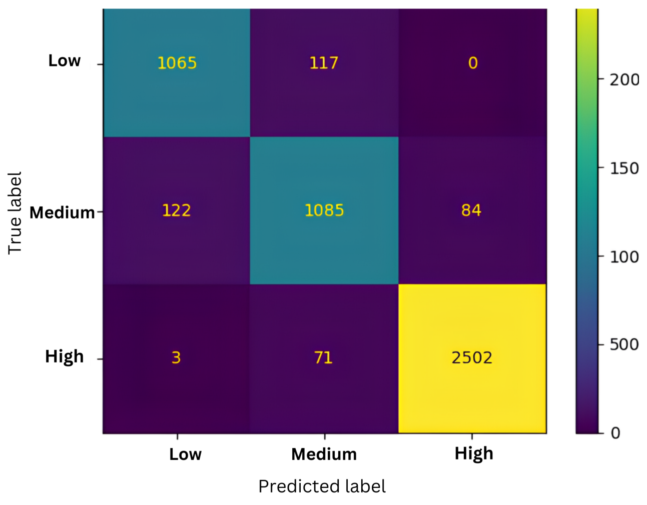
The kNN, Figure 5(c), presented the best results compared to the other two classification models (SVM and Logistic Re-



(a) Confusion matrix for SVM model.



(b) Confusion matrix for RL model.



(c) Confusion matrix for k-NN model.

Figure 5. Confusion matrix for each model.

gression), with an accuracy of approximately 92%.

The confusion matrix allows us to visualize the number of correct and incorrect predictions a model makes for each class. According to the results, the classification models have performed very well in this scenario. The kNN model has achieved an accuracy of 92%, indicating that it can accurately predict and classify priorities with high precision.

An accuracy of 90% indicates that the model is making positive decisions with considerable confidence and minimizing the errors obtained; combined with this, an average recall of 89% means that the model could correctly identify around 89% of the positive instances present in the dataset. These positive instances refer to samples or instances within the data set that belong to the class considered ”positive” for the classification problem. In this case, they are the classes that the request belongs to Low, Medium, or High.

The model achieves an average F1 score of 87%, indicating a well-balanced performance between precision and recall. This score reinforces a consistent and well-balanced performance of the model.

Therefore, the classification model that proved to be most efficient was kNN, despite being only slightly superior to Logistic Regression. An interesting factor that can influence this and change the performance of a model is its hyperparameters. Classification models can be ”tuned.” based on hyperparameters to improve results, it was not the object of study in the present work.

kNN and Logistic Regression are machine learning algorithms with unique properties and advantages. Depending on the problem at hand, one may be more suitable than the other. Below, we highlight some advantages of kNN compared to logistic regression, which can contribute to obtaining more favorable results, although it does not necessarily guarantee superior results:

- kNN is a non-parametric method, which means it does not make any specific assumptions about the data distribution. It relies on its nearest neighbors to make decisions, giving it more flexibility when dealing with input data. Logistic regression, on the other hand, assumes a specific distribution of data, usually a binomial or multinomial distribution.
- kNN can handle non-linear relationships between features and output. It can capture intricate decisions that logistic regression, being a linear model, struggles to model effectively. kNN makes no assumptions about the functional form of the data and can adapt to more complex patterns.
- kNN may be more robust to *outliers* than logistic regression. Since kNN relies on nearest neighbors, the impact of outliers on the outcome may be minimized. On the other hand, logistic regression can be influenced by individual data points and can be sensitive to outliers.
- kNN can better handle imbalanced datasets where target classes have different proportions. Because classification relies on nearest neighbors, kNN can adapt to varying class ratios. Logistic regression can encounter issues with a significant class imbalance since it can favor the majority class.

Table 3 compares all priority classification models used in our experiment. It presents average Accuracy, Precision, Recall, and F1-score values and errors.

The performance of these classification models may vary depending on the data and the nature of the problem. In some cases, logistic regression may perform better in terms of accuracy. In other cases, KNN may be better suited to capturing complex patterns in data. It is essential to choose the correct evaluation metrics for each type of problem (classification or regression).

7.2 Experiment B - (CPU)

After selecting KNN as the most suitable model, we will use it to manage resources. This includes regulating resource usage based on each request’s priorities, denoted as R_i. Furthermore, we will continue to assess resource consumption in the edge environment.

In Experiment B, this analysis aims to evaluate the impact of the proposed mechanism on CPU utilization compared to a traditional approach. In the traditional approach, we select the closest edge node to perform a task without considering other factors that affect the use of limited edge resources.

7.2.1 Scenario I and II

We conducted two different scenarios for comparison. In Scenario I of Experiment B, three edge nodes (*fog*) were created, and 500 requests were processed using the mechanism proposed in this work. In Scenario II of Experiment B, the traditional approach, which only considered the edge node closest to the task for processing, was used.

Figure 6 shows the CPU consumption during the execution of the experiment with three edge nodes and 500 requests. It compares the approach proposed in this work with allocation only by distance. In the approach proposed in this work (a), CPU usage remains relatively balanced, not exceeding 80% of node resource utilization. The threshold established for this Scenario is 85%; therefore, if a monitored node reaches or exceeds 85% of its resource usage, it does not receive new requests temporarily.

Without approach (b), requests are directed to the nearest edge node, resulting in irregular consumption with several peaks in usage, maintaining an average consumption between 85% and 90% of resources.

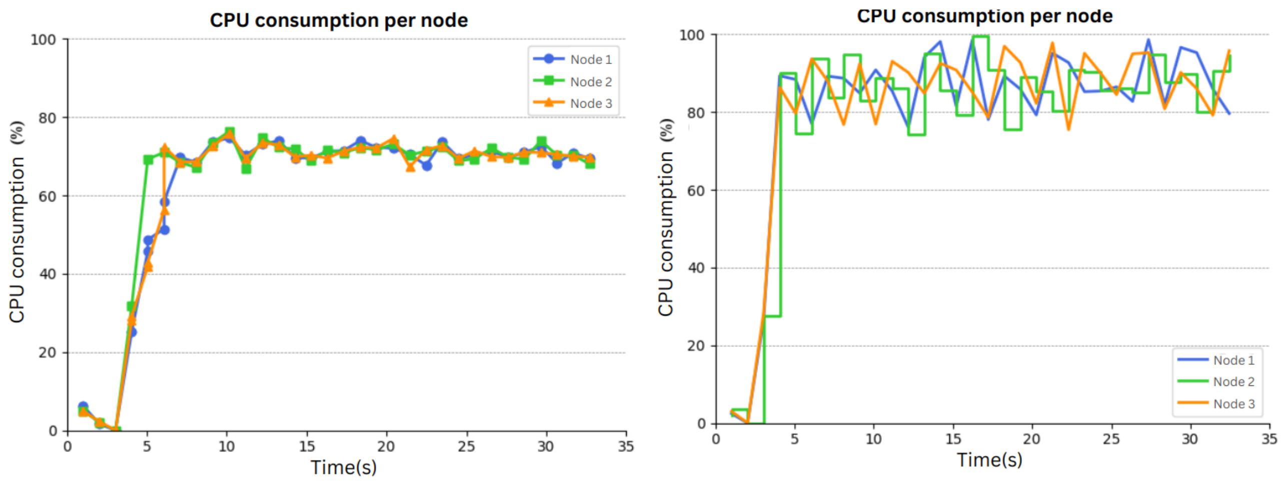
We established a threshold of 85% as a barrier for receiving new requests in all scenarios of the proposed architecture, aiming to avoid overloading the nodes and possible failures. However, if a node reaches, for example, 79% resource utilization, new demand may lead it to exceed this established limit. This measure aims to prevent overloads on nodes to avoid failures. Allocation by distance showed peaks of 100% resource utilization, suggesting overload and potential node failure.

7.2.2 Scenario III and IV

In scenarios III and IV of Experiment B, we doubled the number of requests (1000 requests) to reevaluate the performance of the proposed mechanism. Figure 7 displays the CPU con-

Table 3. Evaluation metrics of the models

Classification Models	Accuracy	Precision	F1-score	Recall
SVM	0.73	0.63	0.56	0.53
Logistic Regression	0.89	0.84	0.78	0.84
kNN	**0.92**	**0.90**	**0.87**	**0.89**



(a) Proposed mechanism **(b)** Allocation by distance

Figure 6. Comparison of CPU consumption for 500 requests.

sumption for this scenario; the scenario involves using three edge nodes handling 1000 requests.

While analyzing the graphics, we observed that the mechanism can increase the volume of requirements. The mechanism operates with permanent but balanced CPU management to ensure that the usage of the system does not exceed the limit of 86% to 87%. Ideally, we should aim to maintain the usage within the 85% limit to meet most of the requirements.

On the other hand, in the scenario where allocation is based on the distance of requests, there is a notable escalation in CPU utilization peaks with the increasing number of requests. These peaks may occur at both higher and lower values, signifying an imbalance in the utilization of resources.

During this experiment, we monitored CPU consumption and all data, allowing us to observe the distribution and use of processing resources in each case. The results obtained are useful for analyzing the proposed mechanism's efficiency by comparison with a traditional approach, especially when using the CPU on our node's edge.

7.3 Experiment C - (CPU)

In Experiment C, we increased the number of fog nodes to 6 and maintained a range of 500 to 1000 requests.

7.3.1 Scenario I and II

In Scenario I and II, with 500 requests as shown in Figure 8, the use of the proposed mechanism resulted in a maximum CPU utilization of around 41% of the total capacity and an average of around 39%, demonstrating its effectiveness in the equitable distribution of tasks among all nodes.

On the other hand, without the mechanism, in Scenario II, with 500 requests, CPU utilization peaks close to 58%, and an average of around 41% can be observed.

These results show that the proposed mechanism has attractive advantages in managing CPU resources and keeping the load distributed among edge nodes. This results in more efficient use of available resources, reducing the risk of overload and enabling more stable performance of the entire system.

7.3.2 Scenario III and IV

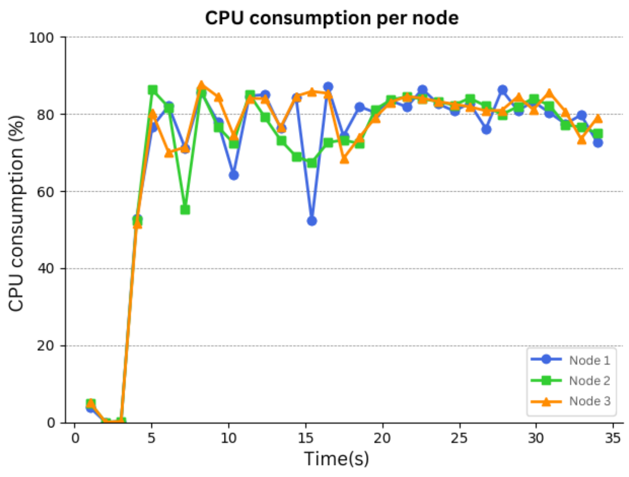
In scenarios III and IV, the number of requests increased to 1000.

On the other hand, when we used allocation by distance (Figure 9 (b)), we observed the highest spikes in CPU utilization, which exceeded 60% usage. It is crucial to note that consumption at the nodes increased exponentially between t=5s and t=7s. However, on average, the consumption remained around 55%, which did not compromise the behavior of the nodes.

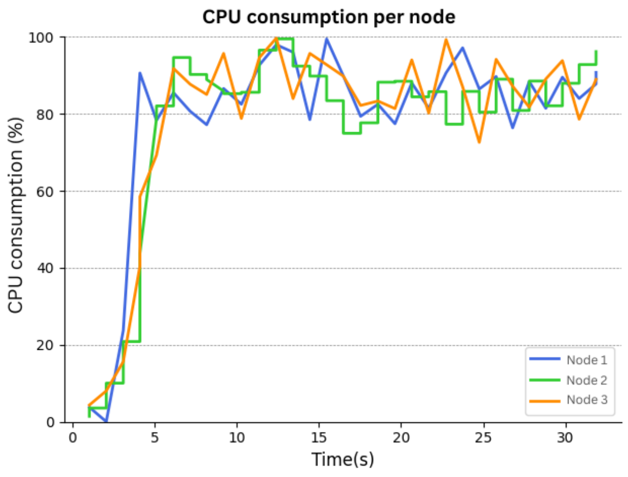
We have noticed that the usage of resources never exceeds 80% of available capacity at any given time. This is due to the increase in the number of edge nodes, which means that the more nodes are available, the more resources are there to distribute the load.

However, it is essential to consider that increasing the number of edge nodes results in higher computational costs. This leads to a trade-off in design decisions that must consider available resources, associated costs, and infrastructure management capacity.

We applied the Student's t-statistical test in Experiments B and C and found sufficient statistical evidence to reject the null hypothesis. The null hypothesis (H_0) states that there is

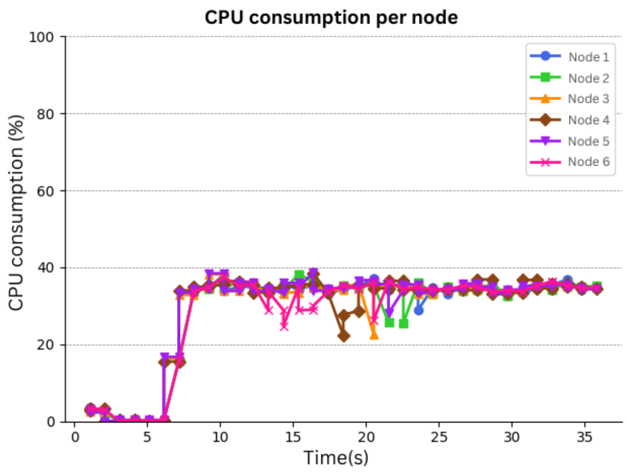


(a) Proposed mechanism

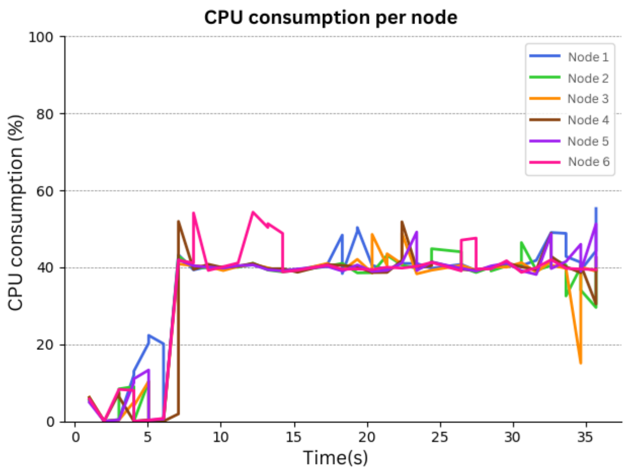


(b) Allocation by distance

Figure 7. CPU consumption 1000 requests.

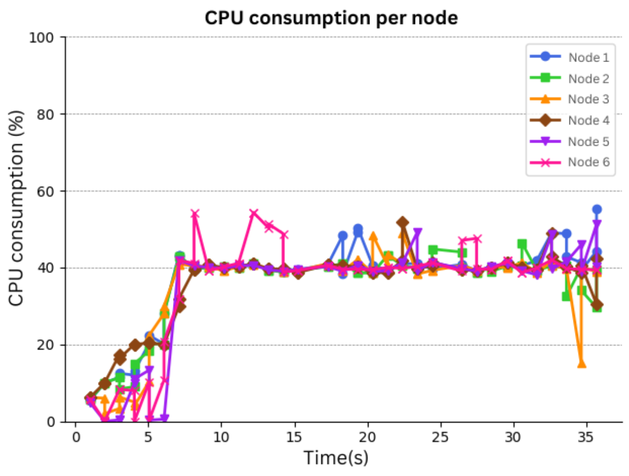


(a) Proposed mechanism

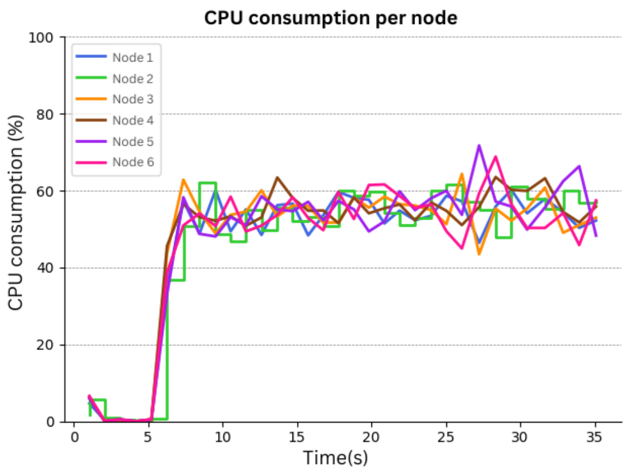


(b) Allocation by distance

Figure 8. CPU consumption 500 requests.



(a) Proposed mechanism



(b) Allocation by distance

Figure 9. CPU consumption 1000 requests.

no significant difference between the means of the compared groups.

Therefore, this result concludes that there is a significant difference between the groups. This indicates that the group means are distinct to a degree that cannot be attributed to chance. Thus, it implies a natural effect from using the two compared approaches.

7.4 Experiment D - (Memory)

During the execution of an experiment, a detailed analysis of RAM consumption was carried out, considering different quantities of requests. Two different environments were compared again: one environment with the implementation of the proposed mechanism and another environment without this mechanism. In the latter environment, the system sends requests to the nearest fog node without considering the management mechanism.

An assessment of RAM consumption is necessary to understand how the proposed mechanism affects system memory resource usage compared to the traditional request dispatch approach. This analysis is valuable for determining the engine's effectiveness in managing memory usage and ensuring adequate resource allocation, especially in high-demand scenarios and under different request loads.

Based on the results obtained, it is possible to better understand the system's behavior regarding memory consumption, evaluate the effectiveness of the proposed mechanism, and make informed decisions to optimize the performance and stability of the edge environment.

7.4.1 Scenario I and II

In Scenario I, the figure (Figure 10 (a)) shows the use of the mechanism proposed in this work with three edge nodes and 500 requests. It is evident from the analysis that memory usage remains low, not exceeding 20%, and with an average utilization of around 16% of total capacity. This indicates that the allocation mechanism efficiently controls the use of resources and performs well.

In contrast, in Scenario II, without using the proposed allocation mechanism, the figure (Figure 10, (b)) still shows three edge nodes and 500 requests. However, the RAM memory consumption has spikes of up to 40% on edge node 1. This is because it is the closest node in the experiment and results in a greater allocation of requests to it. Despite this, due to the low number of requests, the use of memory resources was not affected much. Nevertheless, the proposed architecture and allocation mechanism have proved to be more efficient and controlled regarding memory usage.

7.4.2 Scenario III and IV

In Figure 11, (a), Scenario III, we increased the number of requests to 1000. This time, the memory consumption was greater. We observed peaks of more than 60% memory usage with the proposed mechanism and an average of around 38% to 40%. However, despite the increase, memory consumption is still lower and more regulated than the results obtained without using the mechanism, as shown in (Figure 11, (b)), Scenario IV. In this case, average usage exceeds 40%, and peaks even exceed the 80% barrier.

Again, it is essential to note that, as with CPU resources, if a node becomes overloaded with memory due to uncontrolled request allocation, it can stop working and harm the system. Thus, the results indicate that the proposed mechanism effectively manages memory consumption, keeping it at safe levels and avoiding overload of any particular node.

7.5 Experiment E - (Memory)

Experiment E was carried out in a similar way to Experiment D, except that it involved twice the number of edge nodes, i.e., a total of 6 nodes.

7.5.1 Experiment E - Scenario I and II

In Scenario I (Figure 12,(a)), the resource consumption was significantly reduced due to the low number of requests and a relatively high number of edge nodes, with usage peaks reaching a maximum of 30%. In contrast, in Scenario II (shown in Figure 12, (b)), where the proposed mechanism was not used, the consumption was slightly higher, with peaks reaching a maximum of 42%.

Despite the limited number of requests, the system demonstrated balanced functioning in this scenario. An average reduction of approximately 8% in memory consumption was observed when compared to the absence of the proposed architecture. Although this may seem like a modest improvement, any gain can be considered a positive indicator in environments with limited resources.

7.5.2 Experiment E - Scenario III and IV

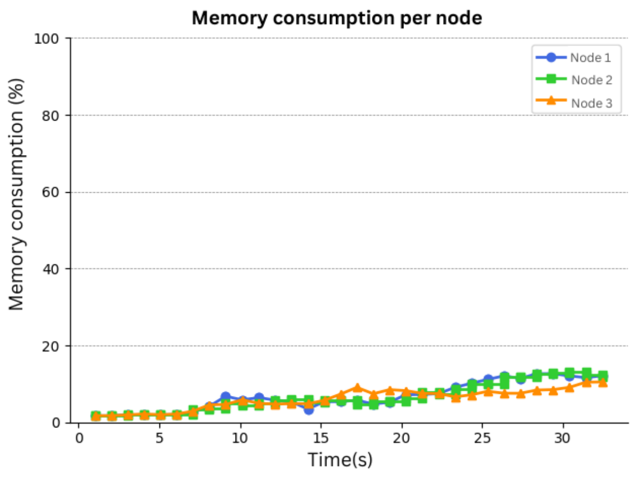
In stages III and IV of Experiment E, when the number of requests increased to 1000, it was possible to observe a similar pattern in stages I and II. The proposed mechanism resulted in a slight increase in RAM consumption, as illustrated in Figure 13, (a), with usage peaks reaching a maximum of 41%.

Meanwhile, in Scenario IV, where the proposed mechanism was not used (as illustrated in Figure 13, (b)), memory consumption increased slightly, slightly exceeding 60%. On average, memory consumption with the minimal engine ranged between 20% and 40%, while distance-based allocation resulted in consumption between 30% and 60%.

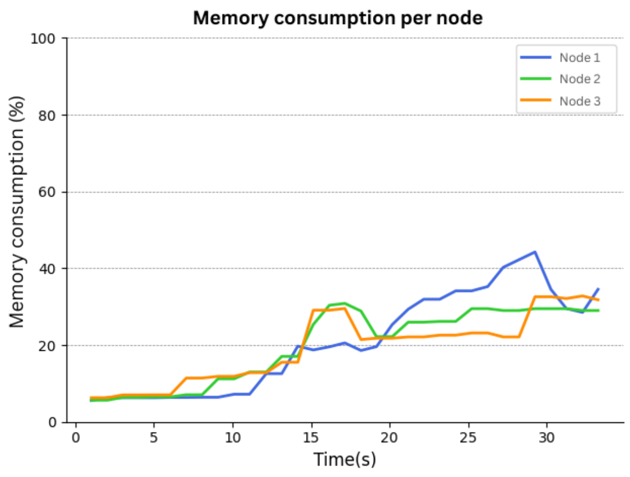
Despite the increase in memory consumption as the number of requests increases, it is notable that the proposed mechanism maintains its role of controlling the use of this resource in an effective and balanced way. In both scenarios, memory consumption does not reach a critical level that could compromise the stable functioning of the system.

Thus, in scenarios III and IV, it is clear that the proposed mechanism continues to provide benefits in terms of management and optimization of memory consumption, ensuring the proper functioning and performance of the system even in situations with more demanding requirements.

It is important to note that the advantage of the proposed mechanism becomes more apparent as the request load increases and the number of edge nodes becomes smaller. In

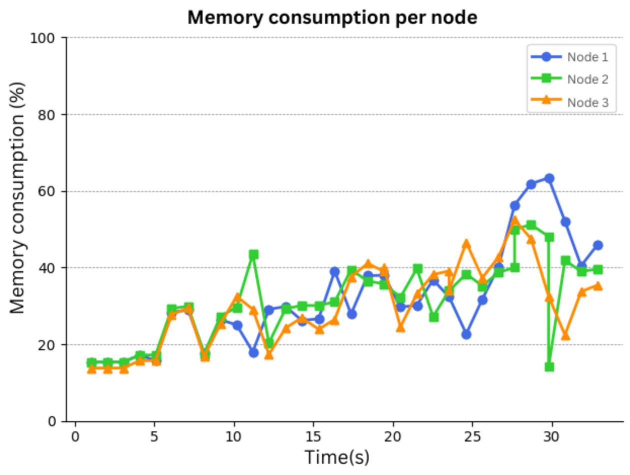


(a) Proposed mechanism

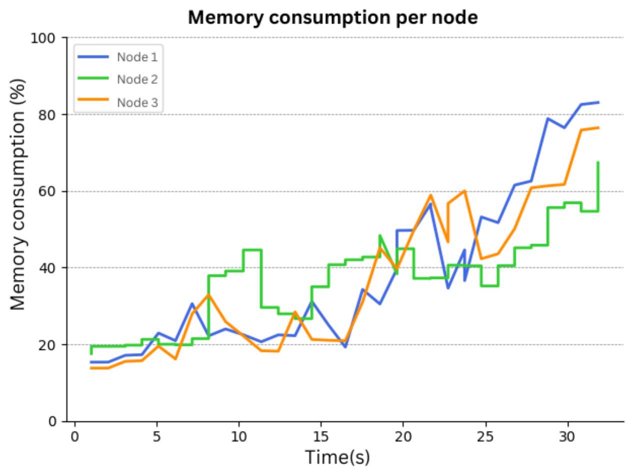


(b) Allocation by distance

Figure 10. Memory consumption 500 requests.

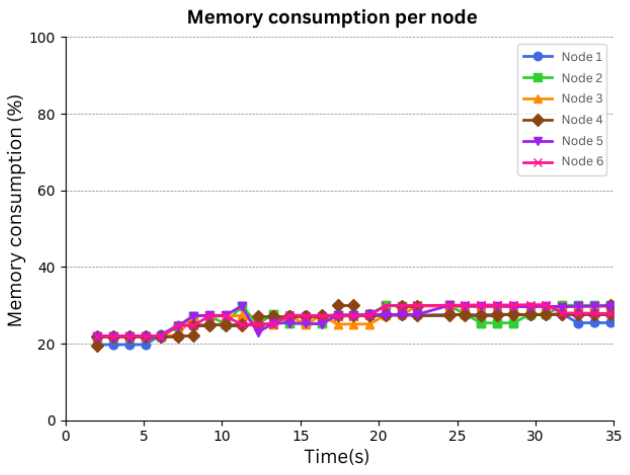


(a) Proposed mechanism

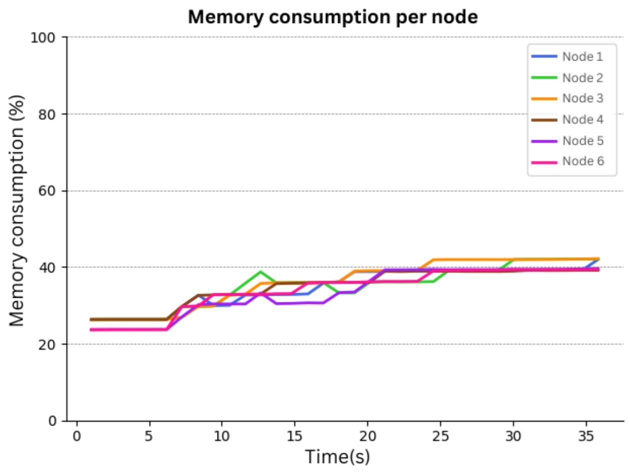


(b) Allocation by distance

Figure 11. Memory consumption 1000 requests.

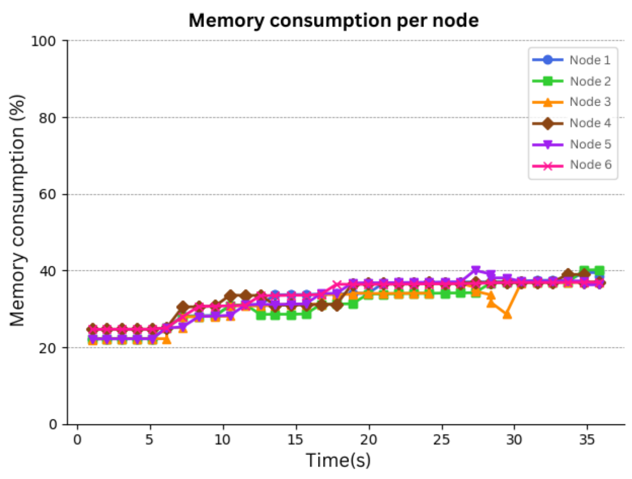


(a) Proposed mechanism

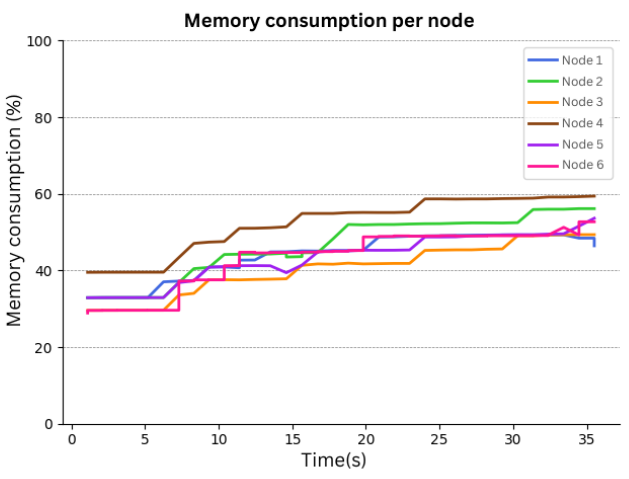


(b) Allocation by distance

Figure 12. Memory consumption 500 requests.



(a) Proposed mechanism



(b) Allocation by distance

Figure 13. Memory consumption 1000 requests.

scenarios with higher demand, the resource allocation mechanism proposed in this work is essential to guarantee a more balanced distribution of resources and avoid overloading specific nodes.

Proper management of memory consumption is crucial to ensure a stable and reliable operation of an edge computing system. Overloaded resources can lead to failures and unavailability of services. The approach proposed in this work proved beneficial in managing memory consumption and contributed to a more balanced infrastructure performance in this environment.

Similarly to the analysis of CPU consumption, we applied the Student's t-test statistical methodology to Experiments D and E, obtaining similar conclusions. In both cases, we could reject the null hypothesis.

Therefore, the findings converge to the existence of statistical evidence supporting a substantial difference. This medium applies to whether or not to adopt the approach presented in this study.

7.6 Experiment F- Failure Mechanism

Experiment F (Figure 14) used scenario III of Experiment E with six edge nodes and 1000 requests. In this experiment, we randomly selected two fog nodes (nodes 3 and 5) to simulate a failure.

First, we selected Node 3 to shut down abruptly, simulating a shutdown due to a lack of resources or a device failure. We observed a drop in Node 3's consumption until it reached 0, showing that, due to the ”failure”, no resources are being consumed by this node. In addition, we observed that after the shutdown (at t = 7.0s), the other nodes started to consume more resources because the requests that were on Node 3 were reallocated to other nodes, which have enough resources to handle these requests. For example, Node 4 gradually increased its consumption.

This situation arises because all requests allocated to edge node 3 are in a state of pending preservation, awaiting resolution. A *Thread* works like a *checkpoint* because, in this case, as a node crashed (Node 3), all requests being processed were lost. However, this *checkpoint* mechanism reallocates these requests to other available nodes that can process them. This process occurs due to continuous monitoring of the resources used.

After the requests are reallocated, the consumption of the nodes that receive them tends to increase until everything stabilizes again temporarily. This process represents balancing the use of nodes previously connected to the disconnected node. This ability to redistribute requests confirms the system's ability to recover from failures once the network failure has been identified and other nodes have absorbed the lost requests. This process is repeated once more at t = 13s when Node 5 shuts down. We can see that there is a gradual drop in consumption by Node 5 and that there is an increase in consumption by the other nodes, proving that the fault mechanism works.

It is essential to highlight that we randomly select the disconnected nodes since it is impossible to predict which node will fail. This format showcases the system's adaptive capability to handle failures by reassigning tasks to the available nodes, thereby ensuring an uninterrupted service.

The failure mechanism under analysis operates in the same way regardless of the monitored resource. This indicates that the results obtained from CPU consumption analysis can be applied to the consumption of other resources., such as RAM. By focusing on CPU consumption, we could validate the functioning of the failure mechanism itself by observing how nodes behave during its execution. This approach allowed us to accurately identify the impacts of the mechanism on the overall system performance. As this is an initial concept validation, it is evaluated only with the use of the CPU resource.

7.6.1 Discussions

By comparing the results obtained in the two environments (with the proposed mechanism and allocation by distance), it is possible to determine whether the allocation with the proposed mechanism effectively improves CPU and Memory utilization and, subsequently, the overall system performance. This analysis is essential to verify and evaluate the

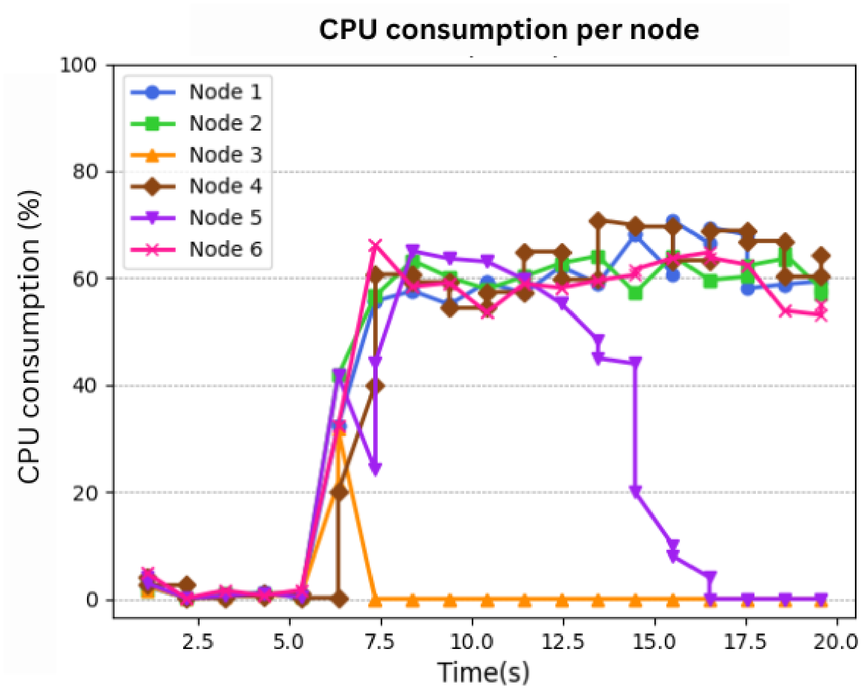


Figure 14. Behavior of nodes with the failure mechanism with 1000 requests.

effectiveness of the resource allocation mechanism proposed in this work. It provides essential information about its usefulness in edge computing environments, from priority definition to request execution. Understanding these results can contribute to making informed decisions about resource management in edge systems to achieve a better balance between performance, computational cost, and resource consumption.

We compared the architecture presented in this work with other priority definition methods covered in the work Bui *et al*. [2021], and it is observed that the authors use different criteria to define priorities. They employ a more complex strategy, considering six factors to classify resource requests, which requires a more elaborate analysis of the requests and, consequently, can increase the total processing time. The results of this work demonstrate that the authors could define their request priorities; however, we noticed excessive consumption of resources, and they highlighted a weakness in the approach we identified in the priority distribution. Although the resource utilization is lower compared to other priority-setting algorithms used by them, the utilization reaches close to 100% and reduces only after a certain period.

In the architecture proposed in this work, in addition to considering several factors when defining priority, as in Bui *et al*. [2021], there was concern about controlling the use of resources on edge nodes. It would be efficient to have a good priority classification for requests with adequate control over the use of resources to resolve these requests, as this could lead to node unavailability. Resource utilization is an essential aspect of efficient resource management. In the results obtained in this work, we observed that in no case tested, there was an excessive overload in the use of edge resources, even with fewer available nodes and increased requests.

Another essential factor to be compared is that the authors in Bui *et al*. [2021] establish using a preemption factor in the algorithm to handle emergencies. This medium can be a valid strategy but can cause network bottlenecks if not appropriately managed. This circumstance arises because resources have already been allocated, and halting this task would lead to the forfeiture of those resources. If a task is running, it is because it has been classified as a priority over the others. Therefore, it is essential to carefully consider the impact of preemption on network operations.

One of the main criteria differentiating the proposed work from others is its failure control mechanism, which plays a crucial role due to the constant exposure of edge nodes to possible failures, such as connection errors, unstable operability, and unavailability. This characteristic is essential, as edge nodes play a fundamental role in processing and forwarding data and requests. This mechanism is a way to control and manage resources at the edge.

8 Conclusion

Resource management is an essential technique in edge computing systems due to the limitations of available resources. This work presents a mechanism to allocate resources efficiently in edge computing environments, prioritizing tasks based on their urgency, necessary resources, and real-time availability on the computing nodes at the edge.

To achieve this objective, we developed a priority classifier for requests based on machine learning techniques. The classifier is trained on historical data and relevant task characteristics such as criticality, latency, connection type, location, elapsed time, and required resource load. Using these data, the priority classification model and runtime resource monitoring provide a priority distribution for requests from IoT devices, enabling efficient resource allocation and reducing unnecessary consumption.

Our evaluations show that the KNN classifier is the most efficient, with an accuracy of 92%, surpassing the SVM and Logistic Regression models. We obtained a precision of 0.90. Therefore, KNN was used in the request allocation and resource control experiments.

In the resource allocation mechanism, the distribution of available resources on edge nodes was optimized, considering the priority assigned to tasks and the quantity and availability of necessary resources. This mechanism results in an efficient allocation of resources, reducing their consumption. For instance, CPU usage has become more stable, varying on average between 75% to 80% for three edge nodes and 40% for six edge nodes compared to the random allocation approach, avoiding consumption peaks that could lead to service unavailability.

The mechanism also demonstrated efficiency in reducing memory consumption and keeping it stable, with an average consumption below the consumption of the approach without the proposed mechanism, especially in scenarios with more requests and fewer edge nodes. These results indicate that the proposed mechanism can adequately handle a more significant load of requests, ensuring a more efficient allocation of memory resources and avoiding overloads on specific nodes.

Furthermore, the proposed failure control mechanism proved efficient in detecting edge node errors and reallocating tasks to guarantee the fulfillment of all requests. This way, it is ensured that the service continues to function even in failure situations.

8.1 Future work

There are some possibilities for improving the proposal presented:

- We plan to perform simulations with a larger number of nodes, in search of more comprehensive results and a more realistic representation of the scenarios.
- Compare our proposal with other approaches that explicitly deal with faults in addition to a comparison with state-of-the-art solutions. This will allow us to assess the ability of our proposal to maintain system performance and reliability in scenarios with faults in network nodes.
- Mobility: Choose a node independent of the node's position, aiming for mobility, since dealing with the geographical position of nodes can be a complex task. Alternatively, Received Signal Strength Indication (RSSI) can be introduced, which is a type of metric that analyzes the quality and power of the connection signal received by a given device. We can use this method to determine the requests from the nearest end devices.
- Adjusting classifier hyperparameters: To enhance the generalization capacity of priority classifiers, it is advisable to fine-tune their hyperparameters. This process aids the model in avoiding issues like overfitting or underfitting, enabling it to discern pertinent patterns in the data and generalize effectively to diverse real-life situations. Additionally, cross-validation should be investigated to validate the selected models.
- To enrich the task prioritization analysis, we will apply other techniques, such as Batch Scheduling, Round Robin Scheduling, and Preemptive Scheduling, and then compare them with the currently employed approach.

Acknowledgements

This work was supported by the Ceará Foundation for Scientific and Technological Development (FUNCAP) through the process MLC-0191-00164.01.00/22.

Competing interests

The authors declare that they have the following competing interests.

Availability of data and materials

The datasets generated and/or analyzed during the current study are available.

References

Aqib, M., Kumar, D., and Tripathi, S. (2022). Classification of edge applications using decision tree, k-nn, svm classifier. In *2022 IEEE Students Conference on Engineering and Systems (SCES)*, pages 01–06. DOI: 10.1109/SCES55490.2022.9887690.

Araújo, G., Bezerra, S., and Rocha, A. (2023). Um classificador de prioridade de requisições para alocação de recursos na computação em borda. In *Anais do XV Simpósio Brasileiro de Computação Ubíqua e Pervasiva*, pages 131–140, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbcup.2023.230787.

Arena, F. and Pau, G. (2020). When edge computing meets iot systems: Analysis of case studies. *China Communications*, 17(10):50–63. DOI: 10.23919/JCC.2020.10.004.

Arowolo, M., Ogundokun, R., Misra, S., Jonathan, O., and Kadri, A. (2022). *K-Nearest Neighbour Algorithm for Classification of IoT-Based Edge Computing Device*, pages 161–179. DOI: 10.1007/978-3-030-80821-1_8.

Bayılmış, C., Ebleme, M. A., Ünal Çavuşoğlu, Küçük, K., and Sevin, A. (2022). A survey on communication protocols and performance evaluations for internet of things. *Digital Communications and Networks*, 8(6):1094–1104. DOI: 10.1016/j.dcan.2022.03.013.

Bhushan, S. and Mat, M. (2021). Priority-queue based dynamic scaling for efficient resource allocation in fog computing. In *2021 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pages 1–6. DOI: 10.1109/SOLI54607.2021.9672442.

Bui, T. B., Sakr, A., Castrillón, J., and Schuster, R. (2021). Six-factors score-based match-making based on priority and preemption for resource allocation in edge computing. In *2021 IEEE International Conference on Edge Computing (EDGE)*, pages 44–50. DOI: 10.1109/EDGE53862.2021.00016.

Cheng, Z., Li, P., Wang, J., and Guo, S. (2015). Just-in-time code offloading for wearable computing. *IEEE Transactions on Emerging Topics in Computing*, 3(1):74–83. DOI: 10.1109/TETC.2014.2387688.

Costa, A., Rocha, A., Delicato, F., and Souza, J. (2020). Balanceamento de carga na borda da rede usando blockchain das coisas. In *Anais do XII Simpósio Brasileiro de Computação Ubíqua e Pervasiva*, pages 91–100, Porto Alegre, RS, Brasil. SBC. DOI: 10.5753/sbcup.2020.11215.

Dlamini, S. and Ventura, N. (2019). Resource management in fog computing: Review. In *2019 International Conference on Advances in Big Data, Computing and Data Communication Systems (icABCD)*, pages 1–7. DOI: 10.1109/ICABCD.2019.8851016.

Hong, C.-H. and Varghese, B. (2019). Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms. *ACM Comput. Surv.*, 52(5). DOI: 10.1145/3326066.

Hussain, F., Hassan, S. A., Hussain, R., and Hossain, E. (2020). Machine learning for resource management in cellular and iot networks: Potentials, current solutions, and open challenges. *IEEE Communications Surveys Tutorials*, 22(2):1251–1275. DOI: 10.1109/COMST.2020.2964534.

Liu, X., Yu, J., Wang, J., and Gao, Y. (2020). Resource allocation with edge computing in iot networks via machine learning. *IEEE internet of things journal*, 7(4):3415 – 3426. DOI: 10.1109/JIOT.2020.2970110.

Madej, A., Wang, N., Athanasopoulos, N., Ranjan, R., and Varghese, B. (2020). Priority-based fair scheduling in edge computing. In *2020 IEEE 4th International Conference on Fog and Edge Computing (ICFEC)*, pages 39–48. DOI: 10.1109/ICFEC50348.2020.00012.

Martinez, I., Hafid, A. S., and Jarray, A. (2021). Design, resource management, and evaluation of fog computing systems: A survey. *IEEE Internet of Things Journal*, 8(4):2494–2516. DOI: 10.1109/JIOT.2020.3022699.

Rejeb, A., Rejeb, K., Simske, S., Treiblmaier, H., and Zailani, S. (2022). The big picture on the internet of things and the smart city: a review of what we know and what we need to know. *Internet of Things*, 19:100565. DOI: 10.1016/j.iot.2022.100565.

Rusman, J., Tahir, Z., and Salam, A. E. U. (2019). Fog computing concept implementation in work error detection system of the industrial machine using support vector machine (svm). In *2019 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, pages 160–164. DOI: 10.1109/ISRITI48646.2019.9034597.

Sasaki, Y. and Yokotani, T. (2019). Performance evaluation of mqtt as a communication protocol for iot and prototyping. 4:21–29. Available at: `https://core.ac.uk/download/pdf/228834689.pdf`.

Scikit-Learn (2023). scikit-learn, machine learning in python. Available at: `https://scikit-learn.org/stable/`.

Sharif, Z., Jung, L. T., and Ayaz, M. (2022). Priority-based resource allocation scheme for mobile edge computing. In *2022 2nd International Conference on Computing and Information Technology (ICCIT)*, pages 138–143. DOI: 10.1109/ICCIT52419.2022.9711641.

Sharif, Z., Jung, L. T., Razzak, I., and Alazab, M. (2023). Adaptive and priority-based resource allocation for efficient resources utilization in mobile-edge computing. *IEEE Internet of Things Journal*, 10(4):3079–3093. DOI: 10.1109/JIOT.2021.3111838.

Tran-Dang, H. and Kim, D.-S. (2018). An information framework for internet of things services in physical internet. *IEEE Access*, 6:43967–43977. DOI: 10.1109/ACCESS.2018.2864310.

Tran-Dang, H. and Kim, D.-S. (2021). Task priority-based resource allocation algorithm for task offloading in fog-enabled iot systems. In *2021 International Conference on Information Networking (ICOIN)*, pages 674–679. DOI: 10.1109/ICOIN50884.2021.9333992.

Wang, K., Tan, Y., Shao, Z., Ci, S., and Yang, Y. (2019). Learning-based task offloading for delay-sensitive applications in dynamic fog networks. *IEEE Transactions on Vehicular Technology*, 68(11):11399–11403. DOI: 10.1109/TVT.2019.2943647.

Wang, Z., Lv, T., and Chang, Z. (2022). Computation offloading and resource allocation based on distributed deep learning and software defined mobile edge computing. *Computer Networks*, 205:108732. DOI: 10.1016/j.comnet.2021.108732.

Yin, C., Li, T., Qu, X., and Yuan, S. (2020). An optimization method for resource allocation in fog computing. In *2020 International Conferences on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData) and IEEE Congress on Cybermatics (Cybermatics)*, pages 821–828. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData-Cybermatics50389.2020.00139.