AJS

Algerian Journal of Sciences
Volume 2 – Issue 1 – PP. 1 – 14

PS-DDAF: Priority-based Scheduling and Deadline/Distance-based Allocation for Performance Optimization in Fog Computing

Imane Moukhtari[1], Rania Bakiri[1], Sarra Cherbal[2*]
[1]Department of Computer Science, Faculty of Sciences, Univ Ferhat Abbas Sétif 1, Sétif, Algeria [2]LRSD Laboratory, Department of Computer Science, Faculty of Sciences, Univ Ferhat Abbas Sétif 1, Sétif, Algeria, sarra cherbal@univ-setif.dz

Abstract

Fog computing is an advanced approach enabling real-time processing and storage to meet the computational demands of Internet of Things (IoT) devices. However, its resource-constrained nature presents challenges in managing limited resources efficiently. Resource management in fog computing is complex due to its distributed computing architecture, involving numerous diverse and decentralized devices functioning as fog nodes. These nodes, though limited in resources, must handle high demand from IoT devices while meeting specific requirements, such as real-time application latency. In this paper, we propose PS-DDAF (Priority-based Scheduling with Deadline-Driven Allocation in Fog), a novel scheduling framework that integrates three key strategies: a task prioritization mechanism, a deadline-aware resource allocation scheme, and a nearest fog node selection method. Unlike existing solutions that address these aspects in isolation, PS-DDAF combines them into a unified two-phase process that improves responsiveness and resource utilization. Extensive simulations demonstrate that PS-DDAF significantly reduces latency, energy consumption, and cost compared to state-of-the-art methods, making it a scalable and effective solution for real-time fog environments.

Keywords: Fog Computing, Resource Management, Scheduling, Allocation, Placement, Tasks, IFogsim.

Article History

Received: 29-11-2024
Accepted: 13-04-2025
Published: 30-09-2025

* Corresponding author
Email (sarra_cherbal@univ-setif.dz)

1 Introduction

The adoption of Internet of Things (IoT) devices is rapidly increasing across various domains, encompassing everything from small-scale sensors, smart phones, and tablets to household appliances and larger equipment. This proliferation of IoT devices results in a massive influx of data that necessitates efficient processing and storage mechanisms [1, 2, 3]. Cloud services were traditionally responsible for handling computing tasks. The concept behind the cloud is to create reusable computing infrastructure, which effectively distributes computing costs [4]. This approach has substantial benefits, as it significantly reduces the initial investment expenses and lowers the operational costs associated with managing computing infrastructure [5]. As a result of the rapid proliferation of IoT devices, the cloud alone could not adequately cope with the increasing demands. Consequently, a new paradigm known as Fog Computing (FC) has emerged to address this challenge.

FC represents the extension of cloud services towards the network edge, positioning itself closer to the data source. This proximity enables FC to carry out real-time data processing, effectively reducing latency and meeting the end users' requirements [6][7].
FC has successfully addressed the limitations of Cloud computing. Nevertheless, it encountered challenges due to being a resource-constrained paradigm, prompting intensive research efforts over the past few years to optimize resource management. These endeavors aim to achieve better resource utilization and enhance the overall quality of service [8][9].
Despite its advantages, fog computing continues to face several critical challenges. These include efficiently scheduling tasks under dynamic and unpredictable workloads, meeting stringent application deadlines, minimizing energy consumption, and controlling costs. Many existing scheduling mechanisms fall short in addressing the real-time demands and resource heterogeneity inherent in fog environments. Overcoming these limitations requires an intelligent strategy capable of prioritizing time-sensitive tasks, while simultaneously optimizing resource allocation to ensure high system efficiency and responsiveness.

1.1 Paper contributions

Our work aims to improve certain aspects of resource management, particularly for time-sensitive applications, which is a key purpose of FC. To this end, we propose a Shortest Job First (SJF)/priority-based algorithm that ensures prioritized tasks are processed first, regardless of their Million Instructions Per Second (MIPS) requirement, with SJF-based execution for tasks of the same priority. Additionally, we propose an allocation algorithm that evaluates the three nearest Fog Nodes (FNs) and a module placement algorithm based on MIPS requirements, deadlines, and the MIPS availability of the FNs. Our objective is to minimize response time while simultaneously improving the quality of service in terms of network usage, cost, and energy efficiency.

1.2 Paper structure

The subsequent sections of the paper are structured as follows. Section 2 reviews relevant related work and provides detailed discussions on each of them. Section 3 presents our contribution, including a detailed explanation of our proposed algorithms, highlighting their key strengths. Section 4 presents our simulation and results, displaying comparison study based on performance metrics. Finally, Section 5 provides the conclusion of the paper.

2 Related work

Many studies have been conducted in the FC field, and interest in enhancing resource management is

increasing, as evidenced by the numerous proposed solutions and methods for better utilization of this resource-constrained paradigm. In this section, we will discuss some related works and highlight their pros and cons.

2.1 Delay and Deadline Sensitive Scheduling

• **A job scheduling algorithm for delay and performance optimization:** In [10], the authors present a job scheduling algorithm designed for FC in smart healthcare applications, where some tasks are critical while others are delay-tolerant. The algorithm prioritizes critical jobs by processing them first, sorting all incoming jobs by MIPS in ascending order to ensure that the shortest tasks are executed first, which minimizes delay for critical jobs.

This approach effectively reduces response time for latency-sensitive applications by prioritizing shorter jobs. However, it is less effective when critical jobs have high processing demands, as longer critical jobs are deprioritized despite their importance.

• **Deadline-aware and energy-efficient task scheduling:** In [11], the authors address task scheduling by proposing a priority-aware semi-greedy (PSG) algorithm aimed at meeting task deadlines, reducing node energy consumption, and minimizing deadline violation time. They calculate response times for each task on available fog nodes (FNs) and categorize FNs into those that meet deadlines and those that do not. The PSG algorithm prioritizes the best FN from the first list and, if unavailable, selects the best FN from the second list to minimize deadline violations.

The proposed algorithm serves well in terms of the energy consumption as it minimizes the FNs and the system's overall consumption. However, when real time and sensitive time applications are not executed in time with respect to their deadlines, they do not benefit from the minimization of deadline violation time.

• **Cost-aware genetic-based (CAG) task scheduling algorithm:** In [12], the authors propose a CAG task scheduling algorithm using a central MasterFogNode. This node estimates task completion times based on available resources and task requirements. The algorithm employs a genetic algorithm with chromosome encoding to represent potential solutions. A fitness function evaluates solution suitability, guiding the algorithm to reduce costs and increase success rates by selecting the best chromosomes for reproduction in subsequent generations.

The proposed algorithm effectively handles latency-sensitive applications by rejecting tasks that cannot be completed on time. It manages many tasks, minimizes cost, and achieves a high success rate. However, it does not account for network congestion or energy consumption caused by communications between the MasterFogNode, fog devices, IoT, and cloud. Additionally, it requires more resources compared to other scheduling algorithms.

• **List-based scheduling algorithm:** Madhura et al. [13] proposed a list-based task scheduling algorithm for directed acyclic graph (DAG) tasks in a FC environment. The algorithm has three phases: 1) Level Sorting, which sorts tasks within each DAG level to enable parallel execution; 2) Task Prioritization, where tasks with more successor tasks are prioritized; and 3) Processor Selection, which assigns tasks to processors to minimize scheduling length and computation costs while optimizing resource usage.

The study in [13] evaluated the algorithm's performance using both random and real-world graphs. It efficiently schedules tasks to appropriate fog processing nodes, considering computation costs and execution times. However, the algorithm has not been extended to dynamic networks, as noted by the authors.

2.2 Resource and Energy Optimization

• **Optimal Node selection:** Alsmadi et al. [14] proposed an Optimal Node Selection algorithm, designed to improve response time and lower energy consumption in a cloud-fog environment. Operating

at the fog layer, the algorithm first checks if fog nodes (FNs) have sufficient resources for user tasks; if not, the task is sent to the cloud. Using a weighted round-robin (WRR) approach, it schedules tasks based on optimal processing capability and available energy.

This approach improves resource usage and load balancing by leveraging the Spanning Tree Protocol (STP) for data collection and routing, aiming for optimal node selection based on node capacity and energy. However, it does not address delay-sensitive applications or the additional network usage caused by frequent data exchange, which increases energy consumption, delay, and throughput. Furthermore, the algorithm may select an optimal node far from the data source, reducing the efficiency of FC.

- **A scheduling algorithm with Bag-of-Tasks Jobs:** The proposed system [15] optimizes re- source utilization and reduces cloud costs by 7% and energy consumption through dynamic task scheduling. Devices are organized into systems and sub-systems with load-balancing based on utilization thresholds. Jobs are routed to the least loaded systems and sub-systems with shortest queues, activating additional resources when necessary. Although the model enhances efficiency, it slightly increases response time, limiting its suitability for time-sensitive applications.

- **Resource Aware Placement of IoT Application Modules in Fog-Cloud Computing Paradigm:** In [16], the authors propose a Module-Mapping algorithm for Fog-Cloud placement, which assigns application modules to eligible network nodes. The algorithm sorts network nodes and modules by capacity and requirements, respectively, and creates Key-Value pairs. It first places modules on eligible fog nodes and, if no suitable node is found, moves the placement to the cloud.

Although the algorithm ensures optimal module placement on the best node, it does not consider network usage or the time required to complete the process.

3 Enhanced scheme: PS-DDAF

Resource management is the central focus of most researchers in the FC field. Many solutions have been proposed to improve its different aspects. In the previous section, we analyzed recent related works and identified their limitations. Based on this analysis, we propose an enhanced scheme designed to overcome these shortcomings while maintaining a high level of quality of service.
This section presents our contribution and proposed algorithms with detailed explanations. PS-DDAF (Priority-based Scheduling and Deadline/Distance-based Allocation Framework) combines priority-based task scheduling with resource allocation strategies. We will use the terms tasks, jobs, and tuples inter- changeably, and modules refer to the computational components of FC.

3.1 PS-DDAF Approach

Based on the shortcomings identified in the related works we reviewed, we propose an upgraded contribution. We focus on time-sensitive applications, considering that this is a key purpose of FC, and aim to improve resource utilization and user experience.
We present an enhanced algorithm that prioritizes critical applications and serves tasks based on their required MIPS, ensuring the minimization of service delivery latency. Additionally, we propose a placement algorithm that maps modules to the FNs based on their MIPS requirements and the availability of the FNs, while considering the module's deadline.
The proposed allocation method considers only the three nearest fog nodes (FNs) to identify a suit- able placement solution. The FNs perform a MIPS comparison between their available resources and the module's requirements, deciding whether to accept and add the module or refuse it, in which case it is transmitted to the cloud. The algorithm aims to reduce delay time, network usage, energy consumption, and cost.

Algorithm 1 Job scheduling

```
Tuples list (SimEvent e1) Tuple we need to execute /* L1,L2,L3...empty List type:
SimEvent*/
/* Ti,Tr type :      Tuple */
ti← (Tuple) e.getData();
if new Tuple arrives Ti  then
    if (L1 = 0 and L2 = 0 and L3 = 0)or(Ti.priority == 3 and L3 = 0)or( Ti.priority == 2 and
L3 = L2 = 0 ) then
        Allocate VM to Ti
        return Ti
    end
    else
        switch priority (Ti) do
            case 3 do
                add Ti to L3 sort (L3) in ascending order of MIPS break
            end
            case 2 do
                add Ti to L2 sort (L2) ascending order of MIPS break
            end
            case 1 do
                add Ti to L1 sort (L1) ascending order of MIPS break
            end
        end
    end
end
if L3 ≠ 0 then
    //poll method get the list head then remove it.
Tr← L3.poll()
    else if L2 ≠ 0 then
        Tr← L2.poll()
        else
            Tr← L1.poll()
        end
    end
end
Allocate VM to Tr
return Tr
```

3.2 Job scheduling algorithm: Shortest job first with priority sorting

Algorithm 1 is a Shortest job first (SJF) and priority-based scheduling algorithm that takes a list of tuples as input. The algorithm checks if the waiting list is empty, then the virtual machine (VM) is allocated to the tuple, and it is executed directly. If the list is not empty, the tuple is added to the waiting list, and the ”priority” method takes it as input.

The algorithm takes an unsorted waiting list of tuples as input and sorts them by priority, with tuples prioritized according to whether they contain critical (time-sensitive), less critical, or latency-tolerant data. The tuple with priority 3 is placed at the head of the list and executed first, followed by the tuple with priority 2, and so on. Tuples with the same priority are further sorted based on their

MIPS, ensuring that the shortest tuples (SJF) are executed first. As a result, the new waiting list is sorted by both priority and MIPS, with the first tuple processed being the one with priority 3 and the smallest MIPS.

The priority-based solution we suggested enhances the work of [10] by granting priority to critical jobs regardless of their MIPS length. However, it serves the shortest job first in cases of the same priority. It also improves the only existing scheduling algorithm in iFogSim, which is FCFS (First Come First Serve), by giving equal priority to all tasks.

The priority-based solution we suggested enhances the work of [10] by granting priority to critical jobs regardless of their MIPS length. However, it serves the shortest job first in cases of the same priority. It also improves the only existing scheduling algorithm in iFogSim, which is FCFS (First Come First Serve), by giving equal priority to all tasks.

Algorithm 2 Allocation and Module Placement

```
Do
List (module, deadline, MIPS)
sort (Device.children.tuples) in ascending order of deadline
if currentdeviceMIPS > RequiredMIPS then
    Add this module to this device
    else
        Search the nearest Right fog device ()
        if RightdeviceMIPS > RequiredMIPS then
            Add this module to this device
        end else
                Search the nearest Left fog device ()
                if LeftdeviceMIPS > RequiredMIPS then
                    Add this module to this device
                end else
                        Send (List) to cloud
                        end
                end
        end
end
While (list of devices ≠ 0)
```

3.3 Allocation and placement

Algorithm 2 is a module placement algorithm that places the modules on their corresponding Fog devices. Each FN identifies its connected IoT devices, which have specific deadlines and capacity requirements. A simple method is used to generate random numbers as the deadline values, with Max and Min values to limit the range. The same approach is applied to the capacity (MIPS). The obtained values serve as the module's attributes, meaning each module will have a (deadline, capacity) pair as its attributes. The values for the deadline are much smaller than the values for the capacity.

After the FN identifies the end device's requirements, the allocation method performs a comparison between the available MIPS of the FN and the MIPS required by the IoT device. If the FN's MIPS is greater, indicating available resources, the module is accepted and added to the FN. Otherwise, the IoT device searches for a suitable FN both to the right and left, so each IoT device searches a total of three FNs. The same comparison is performed by the other nodes, and if none of them accepts the module, it is finally transferred to the cloud. In this way, the modules are allocated to the best-fitted FN with low latency.

This algorithm overcomes the weaknesses in the proposed works in [14], [11], and [16] by checking only the three nearest available FNs, rather than all of them. This approach minimizes delay time, which is crucial for time-sensitive applications, while also reducing network usage.

Algorithm 3 Sorting Method

```
List SimEvent L sorted list /* l← L....Listtype : SimEvent* /
/* sorted...empty List type :      SimEvent*/
/* s type :      SimEvent*/
/* min,n type :      Tuple*/
min← (Tuple) l.get(0).getData();
s← l.get(0)
while (!l.isEmpty()) do
    for (SimEventsimevent : l) do
        n(Tuple)simevent .getData();
        if (n.getCloudletlength() < min.getCloudletlenght()) then
            s← simevent
            min ← n
        end
    end
    l.remove(s)
    sorted.add(s)
end
return sorted
```

3.4 Sorting Method

Algorithm 3 is used to prioritize tuples based on their computational requirements. The method iterates through the list of tasks (SimEvent objects), compares their codelet length, and sorts them in ascending order of processing needs. The result is a list of tasks ordered from the least to the most demanding, allowing for efficient task execution in subsequent algorithms.
Algorithm 1 calls the sorting method from Algorithm 3 to prioritize tuples by MIPS. However, Algorithm 2 sorts modules by deadline without invoking Algorithm 3, as it directly compares node resources.

4 Simulation and Results

In real life, implementing fog-based systems for experimental purposes can be costly due to the required components such as FNs, cloud data centers, communication gateways, and IoT devices. Therefore, simulating and validating fog scenarios using toolkits offers valuable benefits. Simulation allows researchers to study and analyze FC environments virtually, providing a cost-effective alternative to physical implementation.

In this section, we introduce the toolkit used in our work, followed by the implementation of both the related works we have reviewed and our enhanced contribution. Finally, we present the detailed results obtained from these implementations.

4.1 System architecture: Smart parking system

Our proposed framework, PS-DDAF, is adaptable and can be applied to a wide variety of systems, not limited to smart parking applications. For this work, we have chosen smart parking as the test scenario for our simulation, demonstrating the flexibility and applicability of our approach in real-world settings. Finding a parking spot is challenging due to the imbalance between the number of car owners and the available parking spaces, resulting in time-consuming and fuel-inefficient

efforts to reserve a spot. Implementing a smart parking system provides an effective solution to this issue.

The smart parking model presented in [17] is a fog-based system that consists of the cloud, FNs, cameras as sensors, and Light Emitting Diode (LED) display screens as actuators. The parking area is divided into sections, with each section covered by smart cameras that periodically capture images of the parking slots and send them to the connected FN. An image processing algorithm identifies the vacant spots and updates the information on the LED display screens. The processed data in the FNs is then transmitted to the cloud for permanent storage.

The proposed model aims to simplify and improve the process of locating vacant parking spots, ultimately enhancing the convenience for urban dwellers. The system consists of two modules: the picture- capture module and the slot-detector module. The picture-capture module is integrated into a smart camera that captures images of the parking lane at regular intervals (every 5 seconds). These images are transmitted to the slot-detector module, which analyzes them to identify vacant parking spots. Once a vacant spot is detected, the information is relayed to an LED display, indicating the availability of the parking space.
The following figure 1 illustrates the smart parking model.

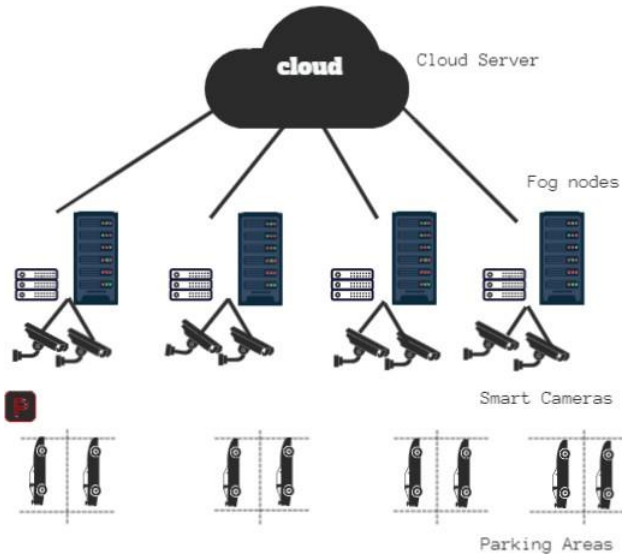


Figure 1: Fog-based smart parking model.

4.2 Architecture Model using IFogsim

Several simulation toolkits are currently available for modeling and simulating FC environments, including FogNetSim++, Edgecloudsim, and IFogsim. Among these toolkits, IFogsim [18][17] has gained significant attention from researchers and has been widely used to model various FC scenarios in recent studies [10], [14], [16], [12], [13]. The IFogSim simulation toolkit is built upon the foundational framework of CloudSim, a widely adopted simulator for modeling cloud computing environments. By extending the basic CloudSim classes, IFogSim enables the simulation of customized FC environments with a substantial number of FNs and IoT devices, such as sensors and actuators.

Our system architecture, as shown in Figure 2, is implemented using the IFogSim simulation toolkit with a graphical user interface (GUI). This GUI allows for the creation of various physical entities, each with specific capabilities and configurations. These entities include sensors, gateways, and cloud virtual machines, along with the establishment of links to describe their connections and interactions within the simulated environment.

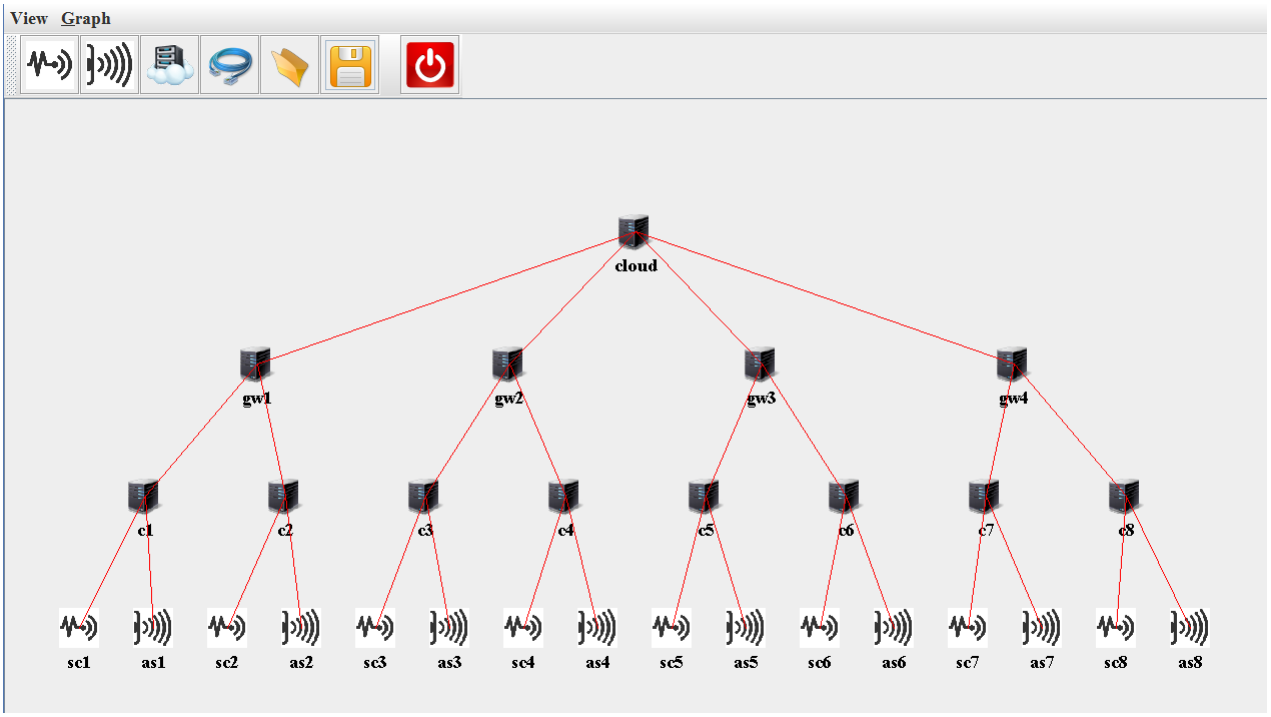


Figure 2: Parking system architecture using IFogsim

4.3 Simulation Parameters

This section will present the different configuration parameters that we used in our simulation. Table 1 presents the CPU and data length of variant tuple types. Table 2 displays the RAM, MIPS, size and Bandwidth (Bw) of the modules in our system. Table 3 presents the configurations of FNs and Table 4 presents fog devices configuration.

Tuple Type	CPU Length (MI)	Data Length (Bytes)
IoT sensor	100	200
Raw Data	6000	600
Store Data	1000	300
Result Data .	100	50
Response Data	100	50

Table 1: Tuple CPU and Data Length

Module	RAM(MB)	MIPS	Size (Bytes)	Bw (Mb per second)
Picture-Capture	10	1000	1000	1000
Slot-Detector	50	1500	4000	800
User-Interface	10	50	12000	100

Table 2: Modules requirement

	MIPS	RAM (MB)	UpBw (Mb/s)	Down Bw (Mb/s)	level	rate per MIPS	busy power (W)	idle power
Cloud	**44800**	**40000**	**100**	**10000**	**0**	**0.01**	**16*103**	**16* 83.25**
Camera	**500**	**1000**	**10000**	**10000**	**2**	**0**	**87.53**	**82.44**
Gateway	**2800**	**4000**	**1000**	**10000**	**1**	**0.0**	**107.4**	**83.43**

Table 3: Configuration of Fog nodes

	Sys	OS	Cost (Monetary units)	Memory Cost	Storage Cost
Each Fog Device	**x86**	**Linux**	**30**	**0.05**	**0.001**

Table 4: Fog Device configurations

Configurations:

- Latency between cloud-gateway=100.
- Latency between gateway-camera=4.
- Latency between cloud-camera=104.
- Latency between gateway-LED=1.
- Number of end devices(camera)=10.
- Number of fog nodes(gateways)=10.
- Number of tuples in the system = (100,500,1000,2000,3000,4000,5000)
- Camera transmission time = each 5s.

4.4 Comparison Graphs

In this section, we will present the different results obtained from implementing our proposed algorithms, the contributions of the recently reviewed related works, and FCFS, as it is the only scheduling algorithm available in IFogsim. The comparison will be displayed in the following graphs with:

- Cloud: is FCFS of IFogsim executed in cloud .
- Edgeward: is FCFS of IFogsim executed in fog nodes.
- SJF(R1): We implemented the proposed SJF algorithm from the related work on job scheduling for delay and performance optimization [10] in the smart parking system using iFogSim.
- Allocation(R2): We implemented the allocation proposal from the work of [14] alongside our SJF/priority scheduling algorithm in the smart parking system using iFogSim.

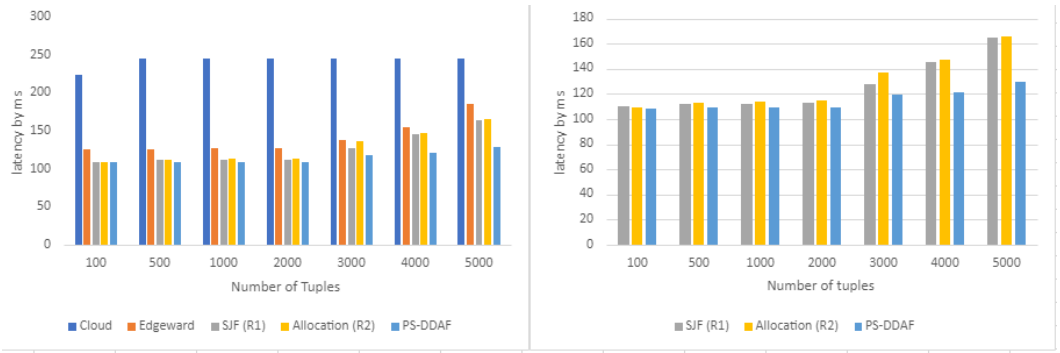


Figure 3: Average loop delay [latency] in (ms) for cloud, edgeward, SJF(R1), Allocation(R2) and our contribution for various numbers of Tuples.

4.4.1 Latency

Figure 3 shows the latency of cloud, edgeward, SJF, Allocation, and our contribution. The number of tuples is shown along the x-axis, while the latency in ms is shown along the y-axis. The comparison of our contribution, related works (SJF and Allocation), and edgeward shows that when the number of tuples is small, the difference is negligible. However, as the number of tuples increases from 3000 to 5000, it becomes clear that our contribution has the lowest latency, demonstrating the efficiency of our allocation algorithm in selecting the FNs. On the other hand, the cloud results are much higher than those of the edgeward, related works, and our contribution. Overall, the results show that our proposed work outperforms the related work, FCFS algorithm, and cloud in terms of latency.

In term of Latency, the improvement ratios of PS-DDAF compared to these approaches are as follows:

47, 37% to Cloud, 30, 19% to Edgeward, 21, 15% to SJF(R1) and 21, 78% to Allocation(R2).

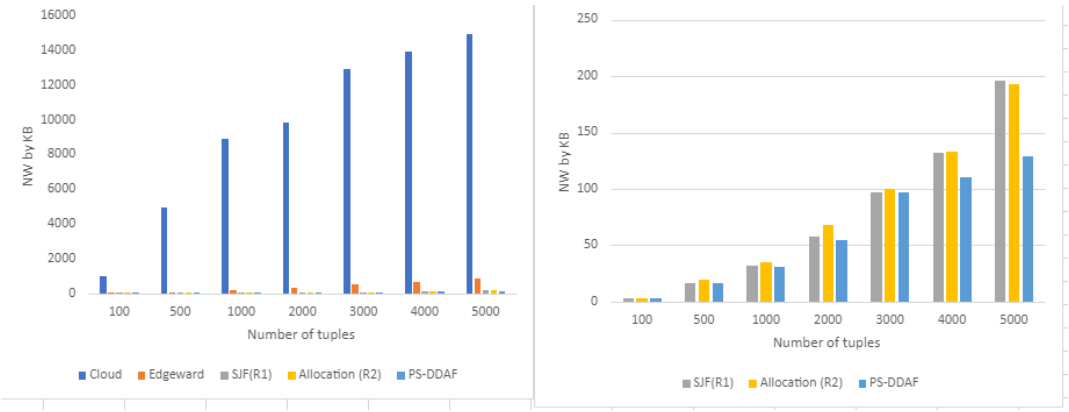


Figure 4: Network usage in [kB] for different numbers of Tuples.

4.4.2 Network usage

In this context, network usage is measured in kilobytes (kB) and represents the cumulative data exchanged among system components throughout the simulation, as reported by IFogSim. This includes communication from sensors to fog nodes, from fog nodes to the cloud, and between fog devices during allocation and module placement decisions.

The figure 4 shows the network usage of cloud, edgeward, SJF, Allocation, and our contribution. The number of tuples is shown along the x-axis, while the network usage (in kB) is shown along the y-axis. The graph indicates that the network usage of the related works, our work, and the edgeward is negligible compared to the cloud, where we clearly observe a contrast between their values, with the cloud showing significantly higher usage. When comparing our contribution to the other related

works, it is evident that our approach results in much lower network usage, demonstrating that our work effectively reduces network usage compared to both the cloud and other proposed fog algorithms.

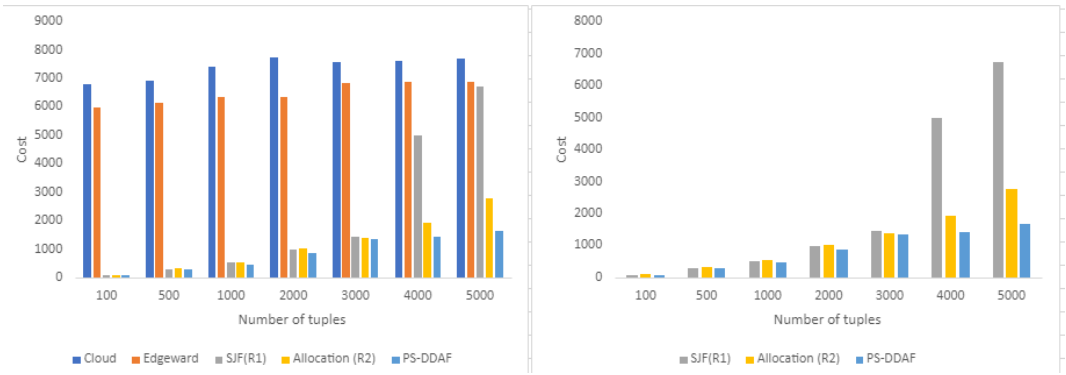


Figure 5: The energy consumed [in J] as the number of tuples varies.

4.4.3 Energy

The figure 5 shows the energy consumption of cloud, edgeward, SJF, Allocation, and our contribution. The number of tuples is shown along the x-axis, while energy consumption is shown along the y-axis. The results displayed in the graph indicate a noticeable divergence between the contributions and cloud/edgeward, with our proposed solution showing lower energy consumption compared to the other related works, and much lower than the edgeward and cloud. Therefore, we can conclude that our contribution improves energy efficiency.

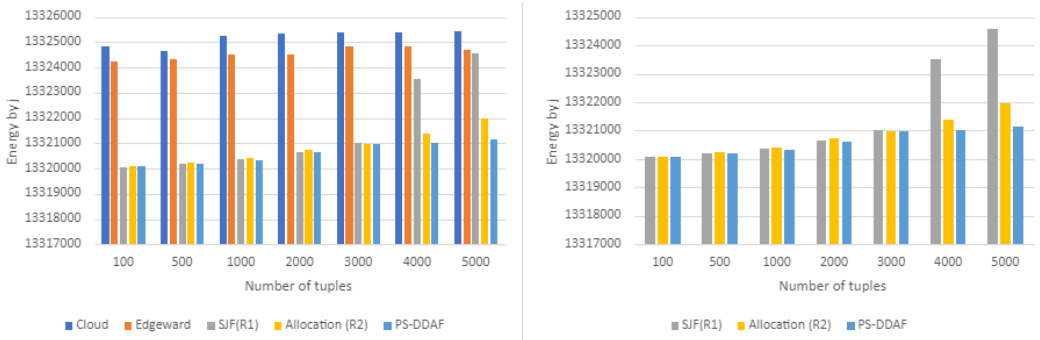


Figure 6: The cost as a function of the number of tuples.

4.4.4 Cost

In this context, cost represents the payment required from users for utilizing computing resources provided by fog and cloud servers. It is calculated based on the duration and intensity of resource usage (CPU MIPS), the utilization level of hosts, and a defined rate per MIPS. This metric reflects the overall financial expense of executing tasks in the system.

Figure 6 shows the cost of Cloud, Edgeward, SJF, Allocation, and our contribution. The number of tuples is shown along the x-axis, while the cost is shown along the y-axis. Compared to the Cloud, the Edgeward values are lower, but both are higher than the values of our work and the related works. The variance among them shows that our contribution has a significantly lower cost than SJF and noticeably lower than Allocation. Overall, the results indicate that our contribution effectively reduces the cost compared to both the Cloud and fog algorithms.

5 Conclusion

The purpose of this study was to enhance the efficiency of fog's limited resources while maintaining QoS. Throughout this paper, we have reviewed and discussed some recent related works and to overcome its shortcomings we have proposed an enhanced solution. We proposed a deadline based placement algorithm, that places modules based on its deadline and MIPS requirement along with the FNs resource

availability, in addition to the allocation method that searches for the nearest optimal node. We also proposed a SJF/priority scheduling algorithm that serves the tasks based on their priority and executes the ones with same priority using SJF. Our contribution was applied on the smart parking model that we selected noting that it could be applied on any giving system and the simulation was done using IFogsim.

We have implemented our contribution and some of the reviewed related works on the smart parking system using IFogsim to get a clear and fair comparison. We have illustrated the comparison using graphs of the obtained results from our contribution, the related work contributions and FCFS algorithm the only scheduling algorithm that exists in IFogsim. Our comparison was based on the following metrics: Latency, Network usage, Energy and Cost.

The findings indicate that our proposed solution performs better in terms of latency, network usage, and cost, while also improving energy consumption.
For future research, we will focus on further minimizing latency while maintaining our enhanced results.

References

[1] Behrouz Pourghebleh, Negin Hekmati, Zahra Davoudnia, and Mehrdad Sadeghi. A roadmap towards energy-efficient data fusion methods in the internet of things. *Concurrency and Computation: Practice and Experience*, 34(15):e6959, 2022.

[2] AR Al-Ali, Ragini Gupta, Imran Zualkernan, and Sajal K Das. Role of iot technologies in big data management systems: A review and smart grid case study. *Pervasive and Mobile Computing*, page 101905, 2024.

[3] Mohammed H Alsharif, Anabi Hilary Kelechi, Abu Jahid, Raju Kannadasan, Manish Kumar Singla, Jyoti Gupta, and Zong Woo Geem. A comprehensive survey of energy-efficient computing to enable sustainable massive iot networks. *Alexandria Engineering Journal*, 91:12–29, 2024.

[4] Mostafa Raeisi-Varzaneh, Omar Dakkak, Yousef Fazea, and Mohammed Golam Kaosar. Advanced cost-aware max–min workflow tasks allocation and scheduling in cloud computing systems. *Cluster computing*, pages 1–13, 2024.

[5] Sururah A Bello, Lukumon O Oyedele, Olugbenga O Akinade, Muhammad Bilal, Juan Manuel Davila Delgado, Lukman A Akanbi, Anuoluwapo O Ajayi, and Hakeem A Owolabi. Cloud computing in construction industry: Use cases, benefits and challenges. *Automation in Construction*, 122:103441, 2021.

[6] Yongheng Zhang, Feng Zhang, Si Tong, and Amin Rezaeipanah. A dynamic planning model for deploying service functions chain in fog-cloud computing. *Journal of King Saud University-Computer and Information Sciences*, 34(10):7948–7960, 2022.

[7] Farshad Firouzi, Bahar Farahani, and Alexander Marinšek. The convergence and interplay of edge, fog, and cloud in the ai-driven internet of things (iot). *Information Systems*, 107:101840, 2022.

[8] Upendra Verma and Diwakar Bhardwaj. Fog computing paradigm for internet of things: Architec- tures, issues, challenges, and applications. *Advances in Distributed Computing and Machine Learning: Proceedings of ICADCML 2022*, pages 1–11, 2022.

[9] Samodha Pallewatta, Vassilis Kostakos, and Rajkumar Buyya. Placement of microservices-based iot applications in fog computing: A taxonomy and future directions. *ACM Computing Surveys*, 2023.

[10] Bushra Jamil, Mohammad Shojafar, Israr Ahmed, Atta Ullah, Kashif Munir, and Humaira Ijaz. A job scheduling algorithm for delay and performance optimization in fog computing. *Concurrency and Computation: Practice and Experience*, 32(7):e5581, 2020.

[11] Sadoon Azizi, Mohammad Shojafar, Jemal Abawajy, and Rajkumar Buyya. Deadline-aware and energy-efficient iot task scheduling in fog computing systems: A semi-greedy approach. *Journal of Network and Computer Applications*, page 103333, 2022.

[12] Tina Samizadeh Nikoui, Ali Balador, Amir Masoud Rahmani, and Zeinab Bakhshi. Cost-aware task scheduling in fog-cloud environment. In *2020 CSI/CPSSI International Symposium on Real-Time and Embedded Systems and Technologies (RTEST)*, pages 1–8. IEEE, 2020.

[13] R Madhura, B Lydia Elizabeth, and V Rhymend Uthariaraj. An improved list-based task scheduling algorithm for fog computing environment. *Computing*, 103(7):1353–1389, 2021.

[14] Ahmad Mohammad Alsmadi, Roba Mahmoud Ali Aloglah, Ahmad Al Smadi, Muneerah Alshabanah, Daniah Alrajhi, Hanouf Alkhaldi, Mutasem K Alsmadi, et al. Fog computing scheduling algorithm for smart city. *International Journal of Electrical & Computer Engineering (2088-8708)*, 11(3), 2021.

[15] Dimitrios Tychalas and Helen Karatza. A scheduling algorithm for a fog computing system with bag- of-tasks jobs: Simulation and performance evaluation. *Simulation Modelling Practice and Theory*, 98:101982, 2020.

[16] Mohit Taneja and Alan Davy. Resource aware placement of iot application modules in fog-cloud com- puting paradigm. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pages 1222–1228. IEEE, 2017.

[17] Kamran Sattar Awaisi, Assad Abbas, Samee U Khan, Redowan Mahmud, and Rajkumar Buyya. Simulating fog computing applications using ifogsim toolkit. In *Mobile Edge Computing*, pages 565–
590. Springer, 2021.

[18] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2016.