IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 27 September 2024, accepted 10 October 2024, date of publication 14 October 2024, date of current version 28 October 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3480360

RESEARCH ARTICLE

FUSION: A Fuzzy-Based Multi-Objective Task Management for Fog Networks

ARYA MOTAMEDHASHEMI[1], BARDIA SAFAEI[1], AMIR MAHDI HOSSEINI MONAZZAH[2], JÖRG HENKEL[3], (Fellow, IEEE), AND ALIREZA EJLALI[1]

[1]Department of Computer Engineering, Sharif University of Technology, Tehran 11365-11155, Iran
[2]School of Computer Engineering, Iran University of Science and Technology, Tehran 16846-13114, Iran
[3]Chair for Embedded Systems (CES), Department of Computer Science, Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany

Corresponding author: Bardia Safaei (bardiasafaei@sharif.edu)

ABSTRACT While the Guarantee Ratio (GR) is critically important in delay-sensitive fog applications, the existing deadline-aware task assignment strategies prioritize the balance of utilization over this criterion. Therefore, this paper introduces FUSION: a fuzzy-based task management policy, which provides a high GR with the least possible makespan. FUSION considers the effect of propagation, uplink/downlink delays, and also the bandwidth between the layers on the tasks' completion time during offloading. It benefits from a fuzzy offloader, along with a VM-ranking strategy based on a fuzzy quantified proposition. Hence, it uses two simple and efficient fuzzy ranking approaches, i.e., Decomposition and OWA. By employing fuzzy-based models, FUSION can handle uncertainty in rapidly changing fog environments with time-varying task sets with minimal computation complexity against existing meta-heuristic algorithms. FUSION considers tasks' size with respect to VM's processing capacity (MIPS), arrival rate, length, deadline, processing time, and execution time. In addition to VMs' load, and busy time, FUSION considers laxity as one of its VM-ranking objectives. FUSION also conducts load-balancing, but only when it can improve the rankings to not affect the GR. Based on the iFogSim simulations, FUSION provides a higher GR in 63% of the scenarios compared to state-of-the-art. Furthermore, evaluations of the offloader algorithm indicate that FUSION provides higher GR in more than 55% of the scenarios.

INDEX TERMS Internet of Things, fog computing, network, task assignment, offloading, scheduling, fuzzy logic, multi-objective, guarantee ratio, makespan.

I. INTRODUCTION

Offloading the computing-intensive tasks from IoT devices to the cloud can significantly improve the computing efficiency of resource-constrained IoT devices [1]. Nevertheless, cloud computing is coping with substantial challenges, including long End-to-End (E2E) delays, traffic congestion, big data process, and communication cost [2]. Meanwhile, E2E delay, and communication costs in particular are caused due to the long physical distance between Data Centers (DCs) and End-Users (EU), located at the edge of the network [2]. These issues degrade the Quality of Service (QoS), which is not well-suited for time-sensitive services, e.g., real-time applications. Fog computing is a virtualized platform [3], which was first introduced by Cisco to extend cloud computing to the edge of a network by providing computing, storage, and networking between DC and EU [4]. In fog computing infrastructures, many geo-distributed devices (fog nodes), e.g., EU devices, routers, switches, and access points, provide cloud-like computing services in proximity to the edge devices to reduce the overall latency.

The generic architecture of fog computing could compromise several fog node layers between the cloud and end users [5]. Among different architectures, the three-layer architecture is widely used and is depicted in Fig. 1 [6]. According to this model, the computation and storage capability, bandwidth, and latency are increased in a bottom-up manner [4]. Various applications can utilize cloud/fog

The associate editor coordinating the review of this manuscript and approving it for publication was Ye Liu.

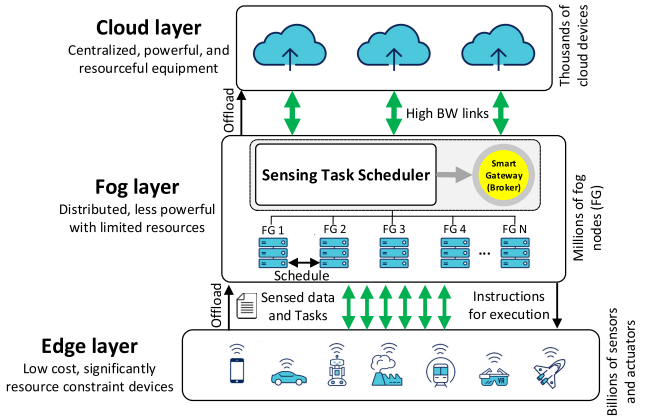


FIGURE 1. The three-tier fog computing architecture.

infrastructure, e.g., smart cities [7], Intelligent Transportation Systems (ITS) and Vehicular Ad-hoc Networks (VANET) [8], smart health-care systems (fall detection [9], ECG monitoring [10], real-time epileptic seizure detection [11] and oxygen level control [12]), and multi-player online video games [13].

Among the various applications that use the cloud/fog infrastructure, those that are latency-sensitive can benefit more from the fog concept due to the closer proximity (e.g., healthcare systems, surveillance in smart cities, VANET, and video games). Most latency-sensitive applications require real-time computations, which necessitate the execution of tasks to be completed before a specified deadline. One of the key factors contributing to the task completion time is the employed task management strategy including offloading, scheduling, and task migration. Meanwhile, since IoT devices are resource-constraint, to finish executing tasks before their deadlines, nodes may need to offload their tasks to a more powerful computation resource at the fog layer. Not to forget that fog devices may also need to offload their assigned tasks to a more powerful device in the cloud layer.

Generally speaking, tasks are members of a bigger set known as a job. In real-world applications, e.g., google clusters contain hundreds of thousands of jobs (applications), Where every job consists of one to thousands of tasks [14], [15]. Post to the offloading, when a job is received at a fog device or a cloud data center, it is decomposed into a set of independent tasks, which will be then mapped into Virtual Machines (VMs) for execution based on a task scheduling policy. The offloading and scheduling mechanisms have a decisive impact on guaranteeing the timing constraints of the tasks (deadlines), their completion time, makespan of the jobs, guarantee ratio, and many other factors in the fog framework. Therefore, the main goal of any assignment mechanism in the field of real-time fog applications must be to provide a high guarantee ratio with the least makespan possible. The guarantee ratio refers to the ratio of completed tasks to the total number of arrived tasks. Having a higher guarantee ratio is especially important in safety and mission-critical applications, where executing even one more task could be a game changer.

Offloading and scheduling mechanisms are usually managed by employing multi-objective fitness functions to consider several factors in the process of task assignment. Several studies have tried to develop deadline-aware task assignment mechanisms for multi-tiered fog infrastructures [16], [17], [18]. While many of these studies do not consider load-balancing, those few who do, prioritize load-balancing over the guarantee ratio, which is a threat to meeting the timing constraints of real-time tasks. While a high and evenly distributed utilization can be valuable, it should not be directly included in the fitness function of real-time fog applications as it can prioritize load-balancing over the guarantee ratio. Establishing a trade-off between load-balancing and other critical performance metrics in real-time fog applications, i.e., latency involves several strategies, that could be taken into consideration in the developed fitness functions. These include adaptive load distribution, priority-based task handling, resource allocation, performance monitoring, predictive analysis (employing predictive analytics to forecast potential bottlenecks and adjust load distribution proactively to prevent performance degradation), and Service Level Agreements (SLAs). SLAs specify the minimum performance metrics that must be met, and design load-balancing strategies to adhere to these agreements.

On the other hand, while executing tasks in more powerful computational resources may decrease the execution time of the offloaded tasks, offloading imposes transmission and propagation delays to the tasks' completion time. However, existing studies neglect the imposed communication overheads between different layers of the fog architecture. Finally, it should not be forgotten that the offloading and scheduling processes have their own processing expenses. Therefore, they must not take a considerable amount of resources in the IoT and fog devices. Accordingly, while previous studies do not pay attention to this matter, we need a lightweight, simple, and easy-to-implement decision-making algorithm.

To address all of the mentioned challenges, this paper proposes **FUSION**: a novel fuzzy-based multi-objective task management policy for multi-tiered fog networks. The main contributions of FUSION are as follows:

- Employs a fuzzy offloader, which decides to offload the tasks before scheduling, based on a set of fuzzy rules. At every layer, FUSION decides whether to keep the task for local execution or offload it to the higher layer. By employing fuzzy-based models, FUSION can handle uncertainty in rapidly changing fog environments with time-varying task sets [19]. Generally, in cases, where it is difficult to develop an exact mathematical model in a rapidly changing, dynamic, and uncertain environment, fuzzy logic is the most employed technique due to its lower computation complexity [20], [21], [22]. The use of fuzzy logic in both the offloader and the ranking algorithms of FUSION makes it easy to implement and

provides lower time complexity than all of the existing meta-heuristic algorithms.

- In case of local execution, the FUSION assignment algorithm ranks the VMs and assigns tasks to the top-ranked VM. The ranking of VMs is based on a fuzzy quantified proposition, which indicates that almost all of the objectives are satisfied. To this end, it uses two simple and efficient fuzzy ranking approaches, i.e., Decomposition (DECOMP) and Ordered Weighted Averaging (OWA).
- Considering various task specifications including the task size concerning the VM processing capacity (MIPS of the VM), arrival rate, length, deadline, processing time, and execution time.
- In addition to VM load, and busy time, FUSION considers laxity as one of its VM ranking objectives. Laxity ensures that FUSION meets the real-time requirements. Laxity refers to the amount of time between the end time of a task and its deadline. The higher the laxity of a task, the more time we save for other tasks to meet their deadlines.
- It conducts load-balancing, but only when it can improve the rankings to not affect the guarantee ratio.
- Considers the imposed communication overheads to the completion time of tasks during the task offloading from IoT/edge devices to fog and cloud layers. To this end, it takes into account the propagation delay, uplink, and downlink communication delays, and also the effect of available bandwidth in the transmitting path.
- Provides a high guarantee ratio with the least possible makespan for real-time fog applications.
- Finally, by indirectly considering throughput through the execution time and meeting deadlines, FUSION ensures that a high throughput alone is not prioritized.

FUSION has been implemented and evaluated on a 3-layer architecture considering a comprehensive set of scenarios in the iFogSim [23]. In this architecture, the cloud layer resides at the top, two levels of fog devices in the middle (to evaluate the offloader), and a number of IoT/edge devices that offload their tasks. According to our evaluations, using an offloader improves the makespan and guarantee ratio in more than 55% of the considered scenarios. Also, FUSION improved the guarantee ratio and execution time in 63% and 89% of the scenarios, respectively. Furthermore, in scenarios, where FUSION shows the highest amount of guarantee ratio, it provided the least makespan against the state-of-the-art.

The organization of this paper is structured as follows: Section II covers the related studies. Section III explains the system model, the scheduling problem, the mathematical formulation of the problem, fuzzy offloading, and fuzzy ranking. In Section IV, FUSION is described in detail. Section V explains the system setup and discusses the evaluation results. Finally, section VI concludes the paper and discusses future studies.

II. RELATED STUDIES

The existing studies in the context of task scheduling can be categorized into two families: 1) Heuristic algorithms and 2) Meta-heuristic algorithms. Heuristics are often problem-dependent, while Meta-heuristics are problem-independent techniques that can be applied to a broad range of problems. In other words, a heuristic exploits problem-dependent information to find a good enough solution to a specific problem, while meta-heuristics are general algorithmic ideas that can be applied to a broad range of problems. First, several important heuristic algorithms will be introduced. Accordingly, the authors in [24] have implemented the Min-Min algorithm. Since this algorithm does not support load-balancing, they have come up with two solutions. One solution is that after the algorithm is finished, the task with the shortest length on the VM with the most load will be selected; then the completion time of that task on other VMs will be calculated. If the completion time on another VM leads to a better makespan, the task will be assigned to that VM, and the timetable of the VMs will be updated. This process continues until no other operation is feasible. The second solution considers task priorities, where tasks labeled with a VIP tag will be scheduled on the VIP resources by the Min-Min algorithm, and the non-VIP tasks will be scheduled on the non-VIP resources based on the first solution.

The Max-Min algorithm is implemented in [25], where at every iteration, the longest task is selected and will be scheduled on a VM, which offers the least completion time of that task. This algorithm also suffers from load-balancing. In [26], a shortest-load-first algorithm is proposed that considers load-balancing. The task with the shortest length is selected and will be assigned to the VM with the lowest load. If two or more tasks have the same length (also shortest), or two or more VMs have the same load, the selection policy would be FCFS. The authors in [27] have proposed an improved version of the Weighted Round-Robin (WRR), whose goal is to perform load-balancing for reducing the makespan. In [28], authors have proposed a fast heuristic algorithm based on the genetic algorithm to offload codes for maximizing the number of tasks that should be executed on wearable devices with guaranteed delay requirements. In another related study, authors have tried to jointly optimize the computing and communication resources in the fog node by formulating a delay-sensitive data offloading problem that mainly considers transmission delay and local task execution. To do so, they have obtained an approximate solution via Quadratically Constraint Quadratic Programming (QCQP) [29].

Meta-heuristic algorithms use optimization techniques to find the optimal or near-optimal solution. In [30], the authors proposed a meta-heuristic algorithm based on the ant colony optimization for cloud environments, which aims to lower the makespan, and balance the load. In a related study, the authors in [17] proposed a meta-heuristic algorithm based on the Ant-colony optimization that considers the deadline of real-time tasks in the cloud and fog environments. The aim of

this algorithm is to maximize the obtained profit by assigning the tasks to VMs. They have also considered load-balancing in their proposed technique. The authors in [31] have used a reinforcement learning approach that considers the deadline of real-time tasks in the edge computing framework. The main goal of this study is to reduce the makespan and manage the energy consumption. Machine learning has been also utilized in [32], where the main objective of the authors is to introduce a reliable backup task assignment strategy for fog environments in terms of both functionality and timing. A cost-aware meta-heuristic task scheduling algorithm based on the genetic algorithm has been introduced in [16], which considers the deadline of real-time tasks on the cloud and fog-based environments. The aim of this algorithm is to schedule as many tasks as possible while minimizing the financial cost.

Two meta-heuristic algorithms are addressed in [33]. In this study, the first algorithm utilizes the ant colony optimization for task offloading to satisfy the quality of service constraints (specifically the response time), while considering load-balancing. The second approach employs an offloading algorithm by using the Particle Swarm Optimization (PSO) technique to minimize response time and maintain load-balancing. In another related study, the authors have come up with a meta-heuristic approach based on the genetic algorithm to minimize makespan, and cost of service [34]. Nevertheless, their newly proposed algorithm does not consider real-time constraints. In [35], an energy-aware meta-heuristic algorithm based on an ant-mating algorithm is proposed. The algorithm considers the tiered IoT network but does not consider real-time constraints. A deadline-aware meta-heuristic algorithm on a tiered IoT structure has been proposed in [18], which utilizes ant colony optimization. The authors in this study have considered the mobility feature and proposed a method for location prediction.

An iterative optimization (TPIO) technique is proposed in [36] to try to optimize capacity and traffic allocation in Mobile Edge Computing (MEC) to satisfy the latency percentage constraint. TPIO has an iterative nature because it uses a tightly coupled two-phase process. The two phases of TPIO capacity and traffic allocation are closely interlinked. Adjustments in one phase directly affect the other, necessitating an iterative process to find an optimal balance. The iterative nature allows the technique to meet two key objectives: providing a minimal-capacity network and ensuring that a certain percentage of traffic meets the latency constraint. Iteration helps in fine-tuning the system to achieve these goals. By iteratively adjusting capacity and traffic, TPIO can more effectively allocate resources where they are most needed, ensuring that the MEC architecture operates efficiently without over-provisioning. The iterative process allows for continuous improvement in performance metrics, such as latency, by allowing for repeated adjustments until the desired performance level is reached. In [37] the authors have proposed an auction method to model the interactions of MEC nodes, and also the likelihood of successful offload from users. They have tried to minimize the cost of offloading to get more users engaged in the offloading process. In another study, authors have proposed a resource allocation mechanism to ensure reliable and low-latency communication for services in space-air-ground networks [38]. Authors in [39] have proposed a scheduling mechanism based on classifying the tasks according to historical scheduling data and creating a certain number of VMs. Then, tasks are matched with concrete VMs dynamically, to improve scheduling performance and achieve load-balancing of resources. In another study, authors have proposed a task scheduling and resource management strategy with minimized task completion time for promoting the user experience [40].

Generally speaking, in critical fog applications, particularly in dynamic and unpredictable environments, e.g., mobile fog networks, several techniques are employed to improve the tasks' guarantee ratio. These include: 1) Task prioritization, 2) Resource management, 3) Implementing fault-tolerant mechanisms, 4) Employing advanced scheduling algorithms, like the Critical Task First Scheduler (CTFS), and 5) load-balancing. Several studies have tried to improve tasks' throughput by proposing latency-aware offloading mechanisms. In [41], the authors have proposed an accelerated gradient offloading strategy to guarantee delay constraints and provide energy-efficient computation for Industrial IoT (IIoT) within a fog computing environment. IIoT has been also the main target of authors in [42], where they have utilized an AI-based Whale Optimization Algorithm (WOA) as part of their proposed QoS-aware offloading mechanism. In another study, authors have claimed that in contrast with the existing techniques, it is impractical to consider a deterministic QoS (delay) guarantee for tasks due to the high dynamics of the mobile wireless environment when offloading to edge servers. Therefore, they proposed a task offloading with a statistical QoS guarantee in mobile edge computing. They introduce a statistical computation model and a statistical transmission model to quantify the correlation between the statistical QoS guarantee and task offloading strategy [43]. Furthermore, energy efficiency problems with performance guarantees in mobile-edge computing are investigated in [44], where an optimization problem for mobile-edge cloud computing is introduced. In another research, a cost-efficient task-offloading method for delay-sensitive applications in fog computing systems is proposed [45]. This technique aims to minimize the overall system cost in terms of energy consumption and makespan, which includes transmission time, processing time, and waiting time. The tradeoff between the response time and the energy utilization has been also the main focus of authors in [46], where they have introduced a Metaheuristic Mountain Gazelle Optimization Algorithm-based task scheduling approach (MMGOA-TSA) in the Next-Generation IoT Fog-Cloud Networks. The same

goal has been pursued in [47], where a multi-objective Harris Hawks Optimization (HHO)-based task scheduling algorithm (MoHHOTS) is developed for cloud-fog computing networks. Finally, a load-balanced offloading mechanism based on PSO with improved response time has been introduced in [48]. While there are industry certifications and regulatory standards concentrating on the reliability, safety, and performance of critical fog applications, they may not explicitly focus on tasks' guarantee ratio. They encompass the broader objectives of ensuring that critical systems are reliable and perform as expected, which includes meeting specific performance benchmarks such as guarantee ratios. A number of safety standards used in industries, include DO-178B for avionics, and ISO 26262 for road vehicles [49].

III. SYSTEM MODEL AND PROBLEM FORMULATION

In our system model, we assume two layers of fog devices between an edge device and the cloud data center. The layer closer to the cloud has more computation capability than the layer closer to the edge device. Each layer has its own offloader. The offloader is aware of the VMs capacities of its layer. Based on the outcome of the offloading, a fog layer may have some (or none) tasks to assign to its VMs. The scheduling algorithm is responsible for the distribution of the fog's tasks among the fog's VMs. For each task, the scheduler ranks the VMs to find a suitable VM that satisfies all the objectives. The problem of finding suitable VM for running a task is a multi-objective decision-making problem as we have multiple objectives to satisfy like guarantee ratio, makespan, and load-balancing. These objectives are typically conflicting in nature. In other words, improvement in one objective may cause deterioration in others. In such problems, it is important to establish the trade-off between all of the considered objectives. Hence, obtaining a single optima solution is very complicated as opposed to the single objective optimization problems [50]. Instead, multi-objective decision-making provides a set of solutions known as Pareto optimal solutions. These solutions have equal importance and are non-dominated by each other but are superior to the rest of the solutions in the search space. The number of Pareto solutions increases with the number of objectives. Therefore, multi-objective decision-making is considered as an NP-hard problem, necessitating the approximation of the Pareto front to reduce the computation and complexity of the applied algorithms. Nevertheless, the wide range of solutions provided to the decision-makers, which enables them to choose a single best solution in a flexible manner has made multi-objective decision-making an appealing approach [51].

Among different approaches for solving multi-objective decision-making problems, fuzzy logic approaches are very effective due to their simplicity both in time complexity and implementation. Accordingly, in FUSION scheduler, we benefit from fuzzy logic-based ranking to rank VMs based on our multiple objectives. In the following, for a given task, FUSION evaluates the rank of each VM. Each VM may have a different evaluation (rank) for an objective. Therefore, the rank of any VM is an aggregation of its rank for every objective. Table 1 represents the definitions of the employed notations in our formulations.

TABLE 1. Definition of Notations in this Study.

Notation	Definition
t^{ij}_{Exec}	Execution time of $Task_i$ on VM_j
$t^{j}_{Current}$	Current elapsed time of VM_j
$t^{ij}_{Completion}$	Completion time of $Task_i$ on VM_j
ΔT_{Job_k}	Makespan of Job_k
$T_{latency}$	Total latency on a path
$T_{propagation}$	Propagation delay on a path
T_{Exec}	Total time of task execution
T^{up}_{Comm}	Total time of uplink communication (Sending)
T^{down}_{Comm}	Total time of downlink communication (Receiving)
$N_{completed}$	Number of tasks finished before their deadlines
N_{total}	Total number of tasks
ΔT_{VM_j}	Total busy time of VM_j
$U_{average}$	Average utilization of VMs
N_{vm}	Number of VMs on a fog node
X_{ij}	Assignment of $Task_i$ on VM_j
Q	Relative Quantifier
μ_A	Membership function of fuzzy predicates
μ_Q	Membership function of the relative quantifier
w_i or w_j	Weights for OWA method
$V(p)$	Value of a proposition in the DECOMP method
τ or F_Q	Value of the proposition in the OWA method
a_i	Arrival time of $Task_i$
e_i	Expected end time of $Task_i$
d_i	Absolute deadline of $Task_i$
l_i	Length of $Task_i$ (in number of instructions)
CPB_i	Cycles Per Byte of $Task_i$
s_i	Size of $Task_i$ (in Bytes)
C_i	Communication time for offloading
P_i	Processing time for offloading
BW_j	The existing bandwidth between IoT node and VM_j
PC_j	Processing capacity of VM_j
GR	Guarantee ratio

A. ARCHITECTURE OF THE SYSTEM

As we mentioned, we consider three-layer architecture having two levels of fog devices between an edge layer and the cloud layer. Except for the cloud layer which does not have any upper layer, each layer has its Offloader. Furthermore, all the layers have their own broker. The broker is responsible for receiving tasks and assigning them to VMs. The offloader is responsible for making a decision before task assignment, to whether keep an incoming task in its layer or offload it to a higher layer. It is worth mentioning that passing tasks between the layers impose communication and propagation delays to their makespan. In fog computing networks, the communication protocol commonly used for conducting task offloading is the Message Queuing Telemetry Transport (MQTT) protocol. MQTT is a lightweight, publish-subscribe messaging protocol that is well-suited for establishing efficient and reliable communications between devices in IoT and fog computing environments [52].

B. SCHEDULING PROBLEM

In real-world applications, e.g., google clusters [14], [15], tracks contain hundreds of thousands of jobs (applications).

Where every job consists of one to thousands of tasks. Due to the computational limitations of IoT devices, they may offload their jobs to the fog or the cloud layer. Each job will be then decomposed into its independent tasks. This set of tasks needs to be assigned to VMs for execution. We assume real-time tasks in this study. Thus, tasks have deadlines (d_i) and we must finish their executions before their deadlines. Therefore, the scheduler must consider the deadline of the tasks. The makespan of a job (ΔT_{Job_k}), is the interval between the instance of time when its first task starts executing to the end of executing its last task. Accordingly, the task with the highest finish time will determine the makespan. The makespan is a critical metric in evaluating the performance and efficiency of task scheduling algorithms in virtualized environments. Makespan is used as a primary indicator of the performance of scheduling algorithms. A shorter makespan typically signifies a more efficient algorithm, as it indicates that tasks are completed in less time. By analyzing the makespan, it's possible to identify bottlenecks in the system. Long makespans may indicate that certain resources are overburdened or that the task distribution is not balanced [53]. It could be also used as an indication for the resource utilization, cost efficiency, and QoS [54]. The ratio of tasks finished before their deadlines to the total number of tasks is known as the Guarantee Ratio (GR). In conclusion, the problem that we are going to solve by FUSION is to assign tasks to VMs in a way that provides a high guarantee ratio with the least makespan possible.

C. PROBLEM FORMULATION

To formulate our problem, first, we need to describe the characteristics of the tasks in the system. Accordingly, we have some jobs that later will be decomposed into several independent indivisible tasks as in the following:

$$Job_k = \{Task_1, Task_2, \ldots, Task_m\}$$
$$Task_i = \{a_i, d_i, s_i, CPB_i, l_i, JobID \in \{1, \ldots, k\}\}$$

where every task has its own set of specifications. Accordingly, a_i represents the arrival time of $Task_i$, d_i represents the absolute deadline of $Task_i$ (the time at which the job should be completed from $t = 0$), s_i is the size of the task in bytes, CPB_i is cycles per byte of the task (the number of instructions needed for execution per each byte), l_i is the number of instructions (length) of the task that has to be executed on the processor of a VM and $JobID$ is the job that owns the task. A job is assumed to be completely executed if its last task (in its task set) is completed. If we consider t^{ij}_{Exec} as the execution time of $Task_i$ on VM_j, and the $t^{ij}_{Completion}$ as the completion time of $Task_i$ on VM_j, the makespan of Job_k (ΔT_{Job_k}) will be calculated via equations (1)-(4).

$$t^{ij}_{Completion} = t^{j}_{Current} + t^{ij}_{Exec} \tag{1}$$
$$t^{ij}_{Exec} = \frac{l_i}{PC_j} \tag{2}$$
$$l_i = CPB_i \times s_i \tag{3}$$
$$\Delta T_{Job_k} = Max\{t^{ij}_{Completion}\}, j \in \{1, \ldots, m\} \tag{4}$$

As it has been mentioned in Table 1, $t^{j}_{Current}$ is the elapsed time from $t = 0$ in VM_j. The total path latency is composed of propagation, and communication delays, and is calculated as in the following.

$$T_{latency} = 2T_{propagation} + T^{up}_{comm} + T^{down}_{comm} \tag{5}$$

where T^{up}_{comm}, and T^{down}_{comm} are calculated based on equations (6), and (7), respectively.

$$T^{up}_{comm} = \frac{s_i}{BW_j} \tag{6}$$
$$T^{down}_{comm} = \frac{Size(Task_i^{Response})}{BW_j} \tag{7}$$

In equation (7), $Size(Task_i^{Response})$ indicates the size of the response corresponding to the execution of $Task_i$ to the IoT device. According to these equations, the uplink and downlink communication delays are dependent on the available bandwidth between the submitting IoT node and the destined fog device VM_j. Considering two fog layers ($FL1, FL2$) between the cloud (CL) and the IoT devices (IoT), according to the following, the propagation delay is calculated based on the offloading destination of $Task_i$.

$$T_{propagation} = \begin{cases} T_{IoT \to FL1} & If\ Task_i\ is\ offloaded\ to\ FL1 \\ T_{IoT \to FL1} + T_{FL1 \to FL2} & If\ Task_i\ is\ offloaded\ to\ FL2 \\ T_{IoT \to FL1} + T_{FL1 \to FL2} + T_{FL2 \to CL} & If\ Task_i\ is\ offloaded\ to\ CL \end{cases} \tag{8}$$

According to (8), if an IoT device sends a task to the first fog layer, the propagation delay is calculated solely for the path between the IoT device and FL1. Subsequently, if the first fog layer decides to offload the received task to the next fog layer (FL2), the propagation delay would be calculated for the path encompassing the IoT device, FL1, and FL2. Finally, if the second fog layer opts to offload the received task to the cloud (CL), the path for calculating the propagation delay would include the IoT device, FL1, FL2, and the cloud. The considered values will be later discussed in Section IV.

In cloud and fog networks, having a low makespan is beneficial due to its positive impacts on the total latency and response time. In real-time applications, where tasks must be finished before their deadlines, we must consider the guarantee ratio to achieve this goal [55]. As it has been discussed earlier, the guarantee ratio (GR) is the ratio of the tasks completed within their deadlines to the total number of tasks [17]. This metric is calculated through equation (9).

$$GR = \frac{N_{completed}}{N_{total}} \tag{9}$$

where $N_{completed}$ represents the number of tasks completely finished before their deadlines, and N_{total} is an indication of the total number of tasks. This metric indicates the ratio of tasks that have completed execution before their respective deadlines, to all of the received tasks. It can be utilized to determine the ratio of tasks that have been finished before their deadlines on a fog or cloud device or for all of the

tasks received from various devices. While the guarantee ratio is a custom metric used in real-time applications, there are more general metrics, which are considered in a wider range of applications. One of the most well-known general metrics is utilization. In queuing theory, the utilization is defined as the ratio of the arrival rate (μ) to the service rate (λ). In the context of fog computing, utilization (U) could be referred to as the portion of time that the VM is busy (in other words, the VM is executing tasks). Therefore, in our study, the arrival rate could be considered as the arrival rate of tasks, that are entered the VM to be executed, while the service rate corresponds to the execution rate of the VMs. Based on this definition, when the number of offloaded tasks to a specific VM exceeds the execution rate of that VM ($U \geq 1$), congestion occurs and the buffer of the hosting fog device starts to fill until it gets fully occupied [56]. Therefore, occurrence of congestion creates an unstable network, leading into task or packet drops, communication overhead and higher delays in the network [57]. Typically, dense network infrastructures are more prone to congestion than sparse networks with limited number of offloaded tasks. In addition to increased delay in the network, the network congestion leads to reduced throughput [58]. Furthermore, congestion also increases the number of retransmissions, which can lead into more energy consumption by the IoT nodes [59], [60]. Based on the principles of queuing theory, in order to avoid congestion and instability in the fog network, we shall keep the utilization under one. As expected, this would also affect the makespan and the total latency. On the other hand, it is undesirable to have idle VMs. Thus, higher (but lower than one) and evenly balanced utilization could provide lower idle times in the deployed VMs (in cases where the VMs capacities are uniform). Having higher utilization values means that the employed algorithm distributes the tasks to the VMs efficiently. The average amount of utilization for the deployed VMs is calculated via equations (10), and (11), respectively.

$$\Delta T_{VM_j} = \sum_{i=1}^{m} t_{exec}^{ij} \tag{10}$$

$$U_{average} = \frac{\sum_{j=1}^{N_{vm}} \Delta T_{VM_j}}{N_{vm}} \tag{11}$$

where ΔT_{VM_j} indicates the total busy time of VM_j, and N_{vm} represents the total number of VMs.

According to the above discussions, the multi-objective offloading/scheduling problem that we are going to solve in this study consists of three objectives: 1) Having the least makespan as possible, 2) Having the highest guarantee ratio, and 3) Having a high and balanced average amount of utilization as possible. Now, let's consider that a $Task_i$ has been submitted by an IoT node. Based on the concept of task assignment, we can consider a random variable X_{ij}, which indicates whether $Task_i$ is assigned to VM_j for execution or not.

$$X_{ij} = \begin{cases} 1 & If\ Task_i\ is\ assigned\ to\ VM_j \\ 0 & Otherwise \end{cases} \tag{12}$$

A real-time task ($Task_i$) must be assigned to only one VM and it must be executed before its deadline. In other words, equations (13) and (14) must be satisfied.

$$\sum_{j} X_{ij} = 1 \tag{13}$$

$$t_{Completion}^{ij} + T_{latency} \leq d_i \tag{14}$$

Thus, the objective function of this study could be determined as follows.

$$F = \{Min(\Delta T_{job}), Max(GR), Max(U_{average})\} \tag{15}$$

D. FUZZY OFFLOADING

Many contemporary fog applications are composed of mobile IoT devices, in which the number of devices, as well as the topology of the network, are highly time variable. On the other hand, different types of tasks with variable resource requirements, lengths, timing constraints (hard, firm, and soft deadlines), and arrival and departure specifications are produced every instance of time. These issues lead to the generation of highly dynamic task sets to be executed by either fog devices or cloud servers. In addition, it is expected that throughput, balance of load among the fog devices, and the finishing time of the tasks will be also time-varying. It has been shown that by employing fuzzy-based models, we can handle uncertainty in rapidly changing environments such as mobile fog applications [19]. Because it allows for the representation of vague or imprecise information [61], [62]. Traditional Boolean logic relies on binary values (true or false), which may not always accurately capture the nuances of real-world situations where information is not always clear-cut. Fuzzy logic, on the other hand, allows for the use of degrees of truth (between 0 and 1), enabling more flexible and nuanced reasoning in uncertain or ambiguous scenarios [61]. Applying fuzzy logic has shown eye-catching improvements when used for solving related network challenges such as congestion mitigation, hand-off control, network, and workload management [63], [64], [65], [66], [67], [68]. Generally speaking, in cases, where it is difficult to develop an exact mathematical model in a rapidly changing, dynamic, and uncertain environment, fuzzy logic is the most employed technique due to its lower computation complexity against other decision-making algorithms [20], [21], [22]. In addition to handling uncertainty, Fuzzy logic can tolerate imperfect data with great precision [69]. In several studies, fuzzy logic has been employed in decision-making to cope with data imprecision [70]. Therefore, based on the specifications of the problem under study, and also the characteristics of fuzzy logic, we were motivated to use this approach in the structure of our FUSION task assignment mechanism. Fuzzy logic is a system for representing the meaning of

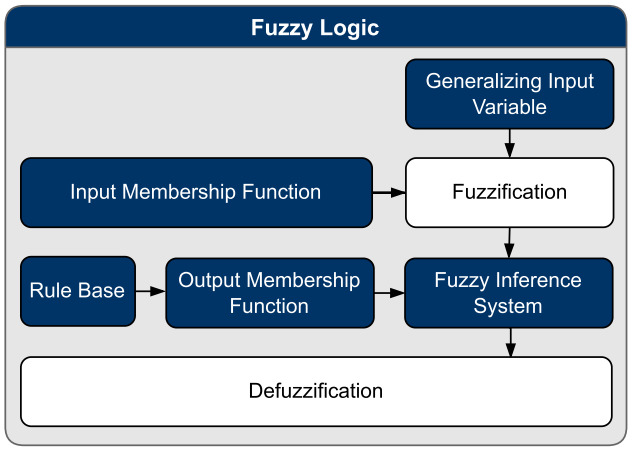


FIGURE 2. Fuzzy logic process diagram.

propositions expressed in a natural language when the meaning is imprecise [71]. According to [71] a proposition in natural language may be viewed as collection of constraints, $C_1, \ldots, C_m$ restricting the value of variables, $X_1, \ldots, X_k$. The constraints and the variables are implicit.

In fuzzy logic, the canonical representation of the proposition p is used to get an explicit meaning of the proposition. The *canonicalform*, $p \rightarrow X$ *is* A, where A is a fuzzy predicate and x is the variable. This canonical form implies that the possibility distribution of X is equal to A, $Poss\{X = u\} = \mu_A(u)$ *where* $u \in U$. Where $\mu_A(u)$ is the membership function of A, the degree of which X is A. In the case of possibility distributions of the form $Poss\{X = u, X = v\} = \mu_A(u) \wedge \mu_B(v)$. The operators $\vee$, and $\cup$ denote the operator *max* and the operators $\wedge$ and $\cap$ denote the operator *min*.

To use fuzzy logic for our problem, we have some input variables, some output variables, set of rules, and an inference method. Each predicate of input and output variables has a membership function. The membership degree of the current values of input variables is called fuzzification. These membership values will then trigger some of the rules which will determine the membership degree of the output variable. With the help of an inference method the membership degree of the output variable will be defuzzified into a value in the domain of the output variable. In short, we determine the output value by the current input values based on some rules. Fig. 2 depicts the process diagram of the fuzzy offloader.

As it can be seen in Fig. 2, in this study we use the same technique for the FUSION offloader. According to this figure, the input variables represent the metric values used to evaluate the performance of our algorithm. During the fuzzification phase, these input values are combined with the associated membership functions of the fuzzy predicates to obtain fuzzy values for the input variables. To calculate the output variable values, a set of rules, membership functions for output fuzzy predicates, and a defuzzification method are employed. The rules yield value(s) for the fuzzy output variable, and the Center of Gravity (COG) method is used to determine the defuzzified value of the output variable. COG defuzzification is a simple, accurate, well-known and long-standing method in the context of fuzzy systems. This technique is a combine-then-defuzzify algorithm that determines the crisp output as the center of gravity of the combined fuzzy set. One of the necessary specifications of a defuzzifier is its continuity; meaning that the slight alterations in the rules should not drastically affect the output of the defuzzifier. It has been shown that the COG defuzzifier is continuous, and as long as the considered fuzzy sets are not empty, it provides a continuous output for continuous membership function [72], [73]. The COG provides relatively higher weights to lower membership values, whereas the other well-known defuzzification technique, e.g., MOM, neglects lower membership values [74]. As it has been mentioned in [75], COG never exhibits RMS error. Generally speaking, COG is an important property since it reflects both the location and the shape of a fuzzy set definition [76]. Therefore, COG has been selected as the defuzzifier technique in this study.

The input variables are: *tasksize*, *deadline*, and Ci/Pi (the communication time over the processing time). The fuzzy predicates of the input variable *tasksize* are: *small*, *medium*, and *large*. These predicates are based on the capacity of the VMs residing on the offloader layer. The fuzzy predicates for input variable *deadline* are: *early*, *moderate*, and *high*, which means how close the deadline of a task is to the current time. The predicates for the input variable Ci/Pi are: *low*, *modearte*, and *high*, which denote whether or not a task has more communication time than its processing time. The fuzzy predicates for the output variable *status* are: *offload*, *probable*, and *local*. As we mentioned, we use COG as the defuzzification method.

Since FUSION has two fog layers and each layer has one offloader module, we will have two different rule bases. The rules in a fuzzy logic system are often expressed in the form of ''if-then'' statements. There are three inputs each having three fuzzy predicates, therefore the total number of rules will be $3^3 = 27$. However, we may not need to have all the rules included. Fuzzy logic systems can work well even if some of the rules are skipped [71], [77]. This is because fuzzy logic is designed to handle uncertainty and imprecision, and can still make good decisions even when some of the rules are not included. Table 2 and Table 3 show the rules for each offloader. It should be mentioned that among the rules, only those that have the least impact on the precision, and quality of FUSION have been excluded from the system.

E. FUZZY RANKING

In this section we represent our problem as a fuzzy quantified proposition. Then in order to rank VMs for FUSION scheduling algorithm, we utilize fuzzy ranking algorithms to rank them and eventually assign the top-ranked VM to the current task.

TABLE 2. Offloader layer 1.

Rule No.	Task Size	Deadline	Ci/Pi	Status
1	small	early	low	offload
2	small	early	moderate	probable
3	small	early	high	local
4	small	moderate		probable
5	small	high		probable
6	medium	early		probable
7	medium	moderate		offload
8	medium	high		offload
9	large	early	low	offload
10	large	early	moderate	probable
11	large	early	high	local
12	large	moderate		offload
13	large	high		offload

TABLE 3. Offloader layer 2.

Rule No.	Task Size	Deadline	Ci/Pi	Status
1	small	early		local
2	small	moderate		probable
3	small	high	low	offload
4	small	high	moderate	local
5	small	high	high	local
6	medium			probable
7	large	early		probable
8	large	moderate		offload
9	large	high		offload

1) *LINGUISTIC QUANTIFIERS*

propositions of the form *Q Es are A* or *Q BEs are A*, begin with a quantifier where Q is the quantifier and E is the expression that its membership is evaluated in the membership function of a fuzzy predicate A. This quantifier can be decreasing, such as at most 5, or non-decreasing, such as all, at least 2. These quantifiers mostly are of two kinds. The first kind are called absolute, such as at least 1, at most 2, etc. The second kind are called proportional or relative, such as almost all, at least half, etc. The first kind can be represented as fuzzy subsets of non-negative numbers, whereas the second kind can be represented as fuzzy subsets of the unit interval because it is proportional and it should be represented as a proportion of a whole set.

2) DECOMPOSITION METHOD

Assume we have a proposition p_1 as "Almost All objectives are satisfied by x". This is of the first form as mentioned above. Q ="Almost All", E ="set of objectives", and A = "satisfied by x". The proposition p_2 as "Almost All important objectives are satisfied by x" is of the second form mentioned above. Q ="Almost All", B ="important objectives", E ="set of objectives", and A = "satisfied by x". As proposed in [78], for the truth value of propositions of the form p_1 we have equation (16).

$$V(p) = \underset{C\subset E}{Max}[V_p(C)] = \underset{C\subset E}{Max}[T_1(Q(C), \underset{E_i\in C}{T}(A(E_i)))] \quad (16)$$

where T_1 and T represents a $t-norm$. The elements of $A(E_i)$ are in descending order, $i < j \Longrightarrow A(E_i) \geq A(E_j)$. Therefore for our fuzzy quantified proposition "Almost All objectives are satisfied by x" where Q is a relative quantifier (Almost All) we have equations (17), (18), and (19).

$$T_1[Q(C), \underset{E_i\in C}{T}(A(E_i))] = min(Q(C), \underset{E_i\in C}{T}(A(E_i))) \quad (17)$$

$$\underset{E_i\in C}{T}[A(E_i)] = min[\underset{E_i\in C}{T}(A(E_i))] = A(E_i) \quad (18)$$

$$V(p) = max_{i=1}^{n}[min(\mu_A(x_i), \mu_Q(\frac{i}{n}))] \quad (19)$$

where $i < j \Longrightarrow \mu_A(x_i) \geq \mu_A(x_j)$. In our case, for each task we rank the VMs, and the task is assigned to the top rank VM.

3) OWA METHOD

According to [79], An OWA operator of dimension n is a mapping $f : R^n \rightarrow R$ that has an associated n vector $W = [w_1, w_2, \ldots, w_n]^T$ where:

1) $w_i \in [0, 1]$.
2) $\sum_{i=1}^{n} w_i = 1$.

Furthermore, $f(a_1, \ldots, a_n) = \sum_{j=1}^{n} w_j.b_j$, where b_j is the jth largest of the a_i. The w_j in the summation for the Ordered Weighted Averaging (OWA) method represents the corresponding weight from the weight vector associated with b_j. A number of properties can be associated with the mentioned operators:

- The OWA aggregation is commutative, that is the aggregation is indifferent to the initial indexing of the argument.
- Monotonicity: if $\hat{a}_i \geq a_i$ for all i, then $f(\hat{a}_1, \ldots, \hat{a}_n) \geq f(a_1, \ldots, a_n)$.
- Idempotency: if $a_i = a$ for all i, then $f(a_1, \ldots, a_n) = a$.

The OWA aggregation is bounded by the Min and Max of the arguments. Thus, for any OWA aggregation f, $Min_i[a_i] \leq f(a_1, \ldots, a_n) \leq Max_i[a_i]$. For the quantifier guided aggregation, If Q_1 and Q_2 are two proportional quantifiers, we say $Q_1 < Q_2$ if $Q_1(r) \leq Q_2(r)$ for all r. An important subclass of these proportional linguistic quantifiers are the regular monotonic quantifiers. These quantifiers satisfy the following conditions:

1) $Q(0) = 0$.
2) $Q(1) = 1$.
3) $Q(r_1) \geq Q(r_2)$ if $r_1 \geq r_2$.

Assume the proposition is of the form $QY's$ are B. The first step in the process of evaluating the truth of these quantified propositions is to associate with the quantifier Q an OWA weighting vector W of dimension n, where n is the cardinality of Y. In particular, the weights are obtained as:

$$w_j = Q(\frac{j}{n}) - Q(\frac{j-1}{n}). \quad (20)$$

It can be seen that w_js satisfy the conditions $w_j \in [0, 1]$ and $\sum_{j=1}^{n} w_j = 1$. After obtaining the weights, the truth value τ,

of the proposition $QY's$ are B, can be obtained as:

$$\tau = F_Q(b_1, \ldots, b_n) \tag{21}$$

where F_Q represents the OWA aggregation of b_i using the weights obtained from Q. Where Q represents the proportional linguistic quantifier *"Almost All"*, as equation (20).

IV. DETAILED DESCRIPTION OF FUSION

FUSION has two stages: Fuzzy Offloading and Fuzzy Ranking. When the task list arrives at a fog device, FUSION Offloader (using Fuzzy Logic), decides which tasks should be offloaded to a higher layer device. Then the FUSION Scheduler (using Fuzzy Ranking), finds the most suitable VM for each task. This means that for each task, the VMs are ranked and the top-ranked VM is assigned to that task.

A. FUSION OFFLOADING

Knowing that the offloader has a knowledge of VM capacities of its layer, we use *Task Size* as one of the fuzzy input variables in FUSION Offloader. Since *Task Size* varies during the run-time and we cannot know it beforehand, we measure their size with respect to the VMs capacities. As Fig. 3.a shows, at the first fog layer, the VM capacities are: $\{2000, 3000, 4000, 5000, 6000\}$ Million Instructions Per Seconds (MIPS) and the fuzzy predicates are determined based on the size of a task with respect to the VM capacities. On the other hand, Fig. 3.b shows the membership function of the task size at the second fog layer. The VM capacities at this layer are: $\{9000, 10000, 11000, 12000, 13000\}$ MIPS.

Since FUSION considers real-time application, we assumed that the tasks have deadlines. Fig. 3.c shows the membership function of the deadlines. FUSION considers the communication delays in its task offloading decisions. Indeed, there may be situations where offloading a task to a more powerful fog leads to a higher response time. To take the correct decision in such situations, FUSION uses another input variable which is $T_{comm}/T_{processing}$. This input determines whether it is beneficial to offload the task to a higher level due to the lower communication time or execute it locally due to the higher communication time with respect to the processing time. Fig. 3.d shows the membership function of fuzzy input variable C_i/P_i.

For the fuzzy output variable, FUSION introduces 3 fuzzy predicates: offload, probable, and local. *Probable* means that the task will be offloaded based on the defuzzified value of the output variable. We use this predicate to show some non-determinism. There are cases that the inference of fuzzified values cannot determine whether to offload or execute locally. For these cases, we use this probabilistic approach. Fig. 3.e shows the membership function of fuzzy output variable *Status*.

Algorithm 1 shows how the FUSION offloading works. For each task from the incoming task list, the input variables are fuzzified (lines 2-4). Then in line 5, the defuzzified value of output variable *status* is calculated. In case of the probable decision, if the value of the fuzzy output variable *Status* is less than the lower bound of the probable interval, the task is marked for offload and is added to the offloaded list (lines 6-7). If *Status* is greater than the upper bound of the probable interval, the task is marked for local execution (lines 19-20). When *Status* is in the probable interval, the task is marked for local execution with the probability of *Status* (lines 8-12). Otherwise, the condition $C_i/P_i < 1$ is checked, and the task is marked for offload and added to the offloaded list, if the condition is satisfied (lines 13-16). The default decision would be local execution if none of the conditions mentioned above hold. Ultimately, the offloaded list is returned to the scheduler. It's worth knowing that this list may be empty when there are no task to offload.

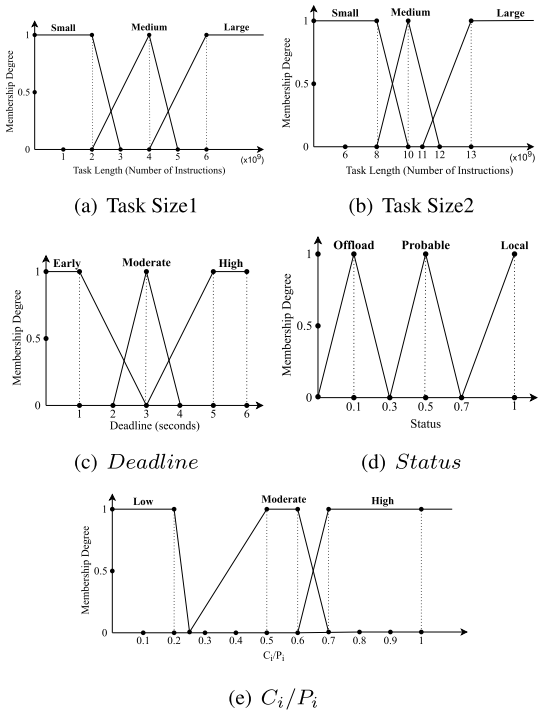


FIGURE 3. Membership functions of input/output variables in offloader.

B. FUSION VM RANKING

To assign VMs to the new arrival tasks in the task list, the FUSION's *Fuzzy-Based Ranking* algorithm ranks VMs and assigns the top-ranked VM to the task. There are three objectives for ranking: laxity, load, and execution time. The laxity ensures that we meet the real-time requirements. By definition, the amount of time between the end time of a task and its deadline represents laxity. This parameter could be also refereed to as slack time [80]. In the context of fog and edge computing, a higher slack time on one server leads to a higher dispatching probability, and then more requests will be dispatched to the server [81]. Accordingly, the higher the laxity of a task, the more time we save for other tasks to meet their deadlines. Since calculating a task's laxity depends on the task's expected end time, and there is no prior knowledge about the expected end times, we need to normalize this

Algorithm 1 Offloader Mechanism

Input *List* $<$ *Task* $>$ IncomingTasks
Output *List* $<$ *Task* $>$ OffloadedList

for each $t \in \mathcal{I}ncomingTasks$ **do**
  $s_i \leftarrow Membership(Size(t))$
  $d_i \leftarrow Membership(Deadline(t))$
  $c_i \leftarrow Membership([C_i/P_i](t))$
  $status \leftarrow Output(s_i, d_i, c_i, COG)$
  if *status* is behind the *PROBABLE* interval **then**
    Add t to OffloadedList
  else if *status* is in the *PROBABLE* interval **then**
    Generate random number R ▷ U(0,1)
    if $R \leq status$ **then**
      Mark t for local execution
    else if $R > status$ **then**
      if $C_i/P_i < 1$ **then**
        Add t to OffloadedList
      else
        Mark t for local execution
      end if
    end if
  else if *status* is out of the *PROBABLE* interval **then**
    Mark t for local execution
  end if
end for
return OffloadedList

Algorithm 2 Scheduler in Fog Node

Input List of Incoming Jobs
Output Map <Tasks,VMs>

Decompose jobs into *TaskList*
TaskList $\leftarrow$ *Offloader*(*TaskList*)
while *Size*(*TaskList*) > 0 **do**
  Remove one task from the list
  Rank VMs (Decomposition or OWA)
  Map the removed task to the top-ranked VM
  Update timetables
end while
return Mapping

Algorithm 3 Scheduler in Cloud Node

Input List of Incoming Jobs
Output Map <Tasks,VMs>

Decompose jobs into *TaskList*
while *Size*(*TaskList*) > 0 **do**
  Remove one task from the list
  Rank VMs (Decomposition or OWA)
  Map the removed task to the top-ranked VM
  Update timetables
end while
return Mapping

objective for the domain of the membership function. This objective (Fig. 4.a) is also used in [17] as profit. Equation (22) shows the normalization formula.

$$Saving = \frac{d_i - e_i}{d_i - a_i} \tag{22}$$

where d_i is the absolute deadline of $task_i$, e_i is the expected end time of the $task_i$, and a_i is the arrival time of the $task_i$. With equation (22), the domain of the membership function is now known. Since *Saving* in equation (22) varies between [0,1] and a higher *Saving* is a better objective, the membership function of this objective would become quite simple as Equation (23).

$$Membership_{Laxity} = \frac{d_i - e_i}{d_i - a_i} \tag{23}$$

Another objective is *load* (Fig. 4.b) which represents the current load of VMs. To calculate the load of VMs, we need their current makespan, which can be calculated by equation (10), and the maximum makespan for normalization. The maximum makespan can be calculated by equation (24).

$$Maxspan = max(\Delta T_{VM_j}) \; ; \; j \in [1, N_{vm}] \tag{24}$$

Finally, the load of a VM can be normalized by equation (25).

$$load = \frac{\Delta T_{VM_j}}{Maxspan} \tag{25}$$

The *load* in equation (25) takes values in [0,1] and a lower load has a higher priority with respect to a higher load.

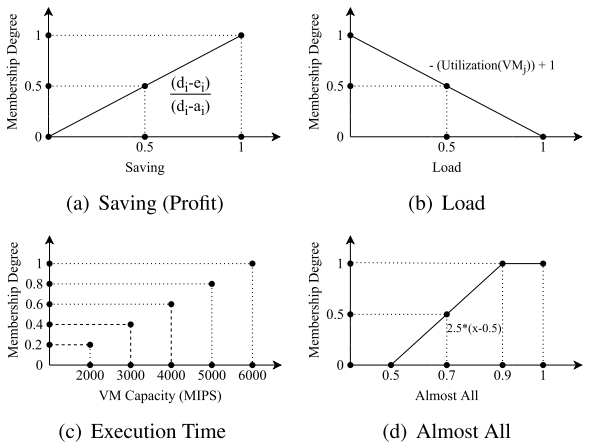


FIGURE 4. Membership functions of objectives and fuzzy proportional quantifier.

Therefore, the membership function of this objective can be calculated by equation (26).

$$Membership_{Load} = -(\frac{\Delta T_{VM_j}}{Maxspan}) + 1 \tag{26}$$

The third objective is the execution time (Fig. 4.c). Considering makespan, a task should have the lowest execution time possible. In Equation (2), we defined the execution time of a task on a VM. For the membership function of execution time, we only need the capacity of the VMs. In each step, the length of the task is fixed, and the ranking algorithm iterates through VMs. The set of the capacities of VMs contains

discrete values. Therefore, the membership function would be a set of tuples. We sort the VM capacities set in ascending order($Capacity(VM_1) \leq \ldots \leq Capacity(VM_k)$), Then tuples are defined in Equation (27).

$$Membership_{Exec} = (Capacity(VM_i), \frac{i}{k}) ; \ i \in [1, k] \quad (27)$$

The membership function of the proportional linguistic quantifier *"Almost All"* (Fig. 4.d) that we used both in the decomposition and OWA approaches, is according to the Equation (28).

$$Q_{AlmostAll}(\frac{i}{n}) = \begin{cases} 0 & (\frac{i}{n}) < 0.5 \\ \frac{5}{2}(\frac{i}{n} - \frac{1}{2}) & 0.5 \leq (\frac{i}{n}) \leq 0.9 \\ 1 & (\frac{i}{n}) > 0.9 \end{cases} \quad (28)$$

Algorithm 2 and Algorithm 3 show that FUSION can be applied in both fog and cloud nodes. Algorithm 2 and Algorithm 3 are almost similar, except that no offloading takes place for the cloud node, because there is no higher level node above the cloud layer. The arriving jobs are decomposed into a list of independent tasks (line 1), the list is then sent to the offloader, and the remaining tasks (if any) will go for scheduling (line 2). For each of the remaining tasks, VMs are ranked using two methods (Decomposition and OWA), and the top-ranked VM is then assigned to the task (lines 3-6). At the end of each iteration, timetables are updated to keep track of the current time (line 7) and finally the mapping is returned to the broker (line 9). In Algorithms 2 and 3, The "Mapping" is the output "Map<Task, VM>". The algorithm takes a list of incoming tasks, and for each task it ranks virtual machines and maps the task to the top-rank virtual machine. In the end, this mapping of tasks to virtual machines is returned so that the broker can use them for execution.

Generally speaking, time synchronization between fog devices is a critical challenge in fog computing networks due to several reasons. Many fog applications require precise coordination between devices, where even small timing discrepancies can lead to significant errors or system malfunctions. Accurate time synchronization is essential for implementing fault-tolerance services and for the correct sequencing of events, which is crucial for maintaining system reliability. Synchronized time enables time-triggered communication, which is necessary for deterministic and predictable data exchange within the network. Furthermore, in distributed systems, data consistency relies heavily on timestamping, and without synchronized clocks, it's challenging to maintain a consistent state across the network. Proper time synchronization can improve the overall efficiency of the network by enabling better scheduling of tasks and reducing the need for time-related error corrections. Accurate time stamps are vital for tracking network usage, identifying latency issues, and detecting security breaches,

Algorithm 4 Updating Timetables

Input $Task_i$, VM_j

1: **if** $t^j_{Current} < Arrival(Task_i)$ **then**
2: $\quad t^j_{Current} \leftarrow Arrival(Task_i)$
3: **end if**
4: **if** $t^{ij}_{completion} + T_{latency} \leq Deadline(Task_i)$ **then**
5: $\quad t^j_{Current} \leftarrow t^{ij}_{completion}$
6: $\quad$ **return**
7: **else**
8: $\quad$ Mark $Task_i$ for offload
9: $\quad$ **return**
10: **end if**

which are all critical for the secure operation of fog networks.

Maintaining synchronization and data consistency when offloading tasks between different layers of the fog computing architecture presents several challenges of its own. For example, due to the distributed structure of fog computing, fog nodes require more communications for synchronization and maintain data consistency. This requires more resources to be taken from computation and given to communications. Thus, the processing power could get lower, leading into more delays and affecting real-time decision-making. Another issue is data freshness, which refers to ensuring that the most recent data is used when tasks are offloaded, which is critical for real-time decision-making. Dealing with the dynamic nature of the network, such as variable latency and bandwidth, can also affect the timely synchronization of data. Fog nodes may have limited computational and storage resources, making it difficult to handle large volumes of data and maintain consistency. In environments like vehicular fog computing, the mobility of nodes can lead to frequent changes in network topology, complicating data synchronization efforts. Furthermore, when the number of devices and the volume of data grows, ensuring consistent data across all nodes becomes increasingly challenging. Concurrency control is another challenge associated with data consistency and synchronization, which refers to managing concurrent access to shared data resources without causing conflicts or inconsistencies. Integrating and synchronizing data from diverse set of heterogeneous devices and platforms with different capabilities and data formats is another challenge in this context. Finally, ensuring that data consistency and synchronization mechanisms do not compromise the security and privacy of the data being handled are also considered as one of the challenges in maintain synchronization and data consistency [82].

Therefore, time synchronization is very important for determining the timing parameters of the tasks, such as arrival time, current time, completion time, etc, for making decisions about scheduling, and offloading. Accordingly, in FUSION, we have used Algorithm 4 to track time. If the arriving task has an arrival time later than the current time, the current

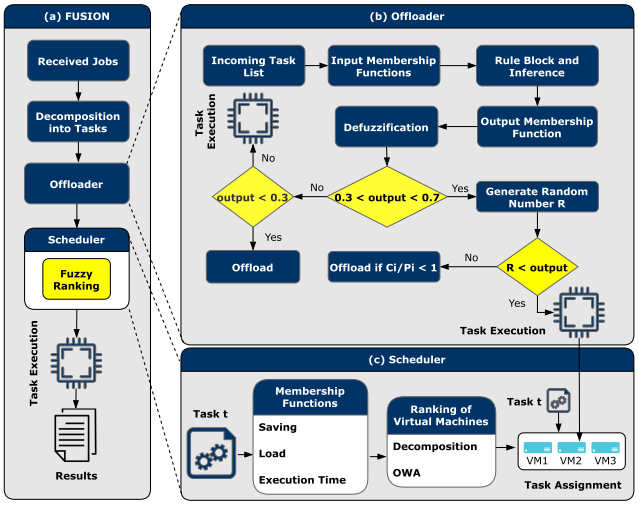


FIGURE 5. Big picture of the FUSION task management.

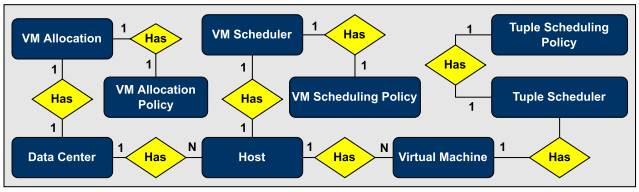


FIGURE 6. Entity Relationship Diagram (ERD) of CloudSim scheduling.

time is then set to the task's arrival time (lines 1-2). If the expected completion time of a task is earlier than its deadline, it is kept for scheduling and the current time is updated to the completion time (lines 4-6). Otherwise, the task is offloaded to a higher level (line 8). Finally, the big picture of the FUSION is depicted in Fig. 5.

As Fig. 5.a shows, the received jobs are decomposed into a list of independent tasks. These tasks are first sent to the offloader. The offloader decides which tasks to send to the upper layer, so that offloading may lead to better execution times considering tasks' deadlines. The remaining tasks are then sent to the scheduler for assignment of suitable VMs. The scheduler outputs a mapping of tasks to VMs which the broker uses to send a task to its VM.

As Fig. 5.b shows, based on each task's characteristics (task size, deadline, and worst-case execution time) as the inputs to fuzzy input membership functions, their degree of membership is calculated. Accordingly, the calculated values trigger rules. The fuzzy output value is then determined by inference and is later defuzzified by the defuzzification method. This defuzzified value is used to make decisions about offloading. If the value is in the interval of $[min_{threshold}, max_{threshold}]$, the task is marked for local execution with a probability. Indeed, a random number R is generated and if $R < Output$ ($Output$ is the defuzzified value of the fuzzy output variable), the task is marked for local execution. Otherwise, it is offloaded only if C_i/P_i (task's utilization) is less than 1. If the output value is less than the $min_{threshold}$, the task is marked for offloading. Finally, if the output value is greater than the $max_{threshold}$, the task is marked for local execution.

As Fig. 5.c shows, considering those tasks which were marked for local execution, a suitable VM will be assigned. Accordingly, for each task, all the VMs are ranked based on three objectives: Saving (or profit), Load, and Execution Time. The ranking is based on decomposition or OWA. Finally, the top-ranked VM is assigned to the current task.

V. SYSTEM SETUP AND EVALUATION RESULTS

To evaluate FUSION, we have used the iFogSim [23] simulation environment, which is based on the CloudSim [83]. iFogSim simulates data centers, data center brokers, VMs, and task scheduling algorithms. Although real-world test beds are highly valuable, iFogSim offers a dependable, economical, and scalable option for deploying fog networks and assessing offloading strategies. Its extensive modeling features and validated outcomes have established it as a respected resource within the research community. Fig. 6, illustrates the entity relationship diagram of the CloudSim scheduling mechanism. The VM allocation policy, VM scheduling policy, and cloudlet scheduling policy are the default policies in CloudSim (space-shared). We also implemented our own Cloudlet task scheduler (space-shared) to consider deadlines for the tasks in order to prohibit their execution if their deadlines are not met. It is also worth mentioning that our proposed technique employs a non-preemptive task execution mechanism. For the FUSION fuzzy offloader, we used jFuzzyLogic Java library developed by [84] and [85], which supports Fuzzy Control Language (FCL). The definition of membership functions of input and output variables, fuzzy predicates, and rule blocks can be implemented via FCL. jFuzzyLogic has implemented several defuzzification methods including the Center of Gravity (COG).

We considered 3 data centers, one for each layer, each of which has to allocate its resources to 5 VMs. There is one Processing Entity (PE) for each of the VMs. There is a data center broker at each layer that receives tasks and maps them to VMs. According to the mentioned architecture in Section III, each of the two fog layers has its fuzzy offloader module. In total, 25 IoT/edge devices have been considered in the simulations that submit their jobs for execution to the fog or the cloud. Each job consists of several tasks, and their considered model has been discussed in detail in Section III-C. We assume that each job of the IoT devices has an equal number of tasks that differ in size and length. For the communication time of a task, its size is the contributor, and for the execution time or processing time, its length is the contributor. To calculate the expected finishing time of a task, we only need $T_{propagation}$, T_{exec}, and T^{up}_{comm}. Since the

TABLE 4. Simulation environment parameters.

Parameter	Value
No. VMs	15
No. tasks	200-1000 (Increments of 100)
No. Data Centers	3
No. Hosts	3
No. PEs of each host	5
No. PEs for each Cloudlet	1
No. IoT Devices	25
PEs of each Host	2
VM capacity	500-5000 (MIPS) mult. of 500
Task deadline intervals	[1, 3], [2, 4], [4, 6] seconds
Length of tasks	[50, 1500] (MI), $CPB \times Size$
Size of tasks	[100, 1000] KB
Cycles Per Byte	[500, 1500]
BW from IoT $\rightarrow$ FL1	100 Mbps
BW from FL1 $\rightarrow$ FL2	120 Mbps
BW from FL2 $\rightarrow$ cloud	200 Mbps
Latency from IoT $\rightarrow$ FL1	0 ms
Latency from FL1 $\rightarrow$ FL2	100 ms
Latency from FL2 $\rightarrow$ cloud	300 ms

returned task size (the response to the submitted task) is small, the T_{comm}^{down} is negligible.

For the sake of simplicity, we consider a single link between each layer with the same throughput as in multiple parallel links. Assuming the same throughput will not affect the arrival time of the incoming tasks. On the other hand, considering a single link reduces the complexity of handling multiple parallel links. Due to the physical proximity, we assumed that the latency between the IoT/edge devices and the first fog node is negligible. Table 4 summarizes the most important simulation environment parameters, which have been considered in this study.

As stated in Table 4, the dataset is composed of tasks, VMs, number of fog nodes, number of IoT devices that send tasks, and the latency for each path. The size, cycles per byte, and deadline of a task are obtained from a uniform distribution mentioned in Table 4. The number of tasks for every scenario has been increased from 200 to 1000 with steps of 100 tasks. It is also assumed that each IoT device sends the same number of tasks. Each cloud/fog node is composed of a data center. Each data center has some hosts and a broker, and each host is responsible for managing VMs. The VM capacity is also obtained from a uniform distribution mentioned in Table 4. The information regarding the resources of data centers and hosts is also indicated in Table 4.

A. PERFORMANCE METRICS

We have evaluated and compared the performance of FUSION with state-of-the-art from six different metrics. These metrics include:

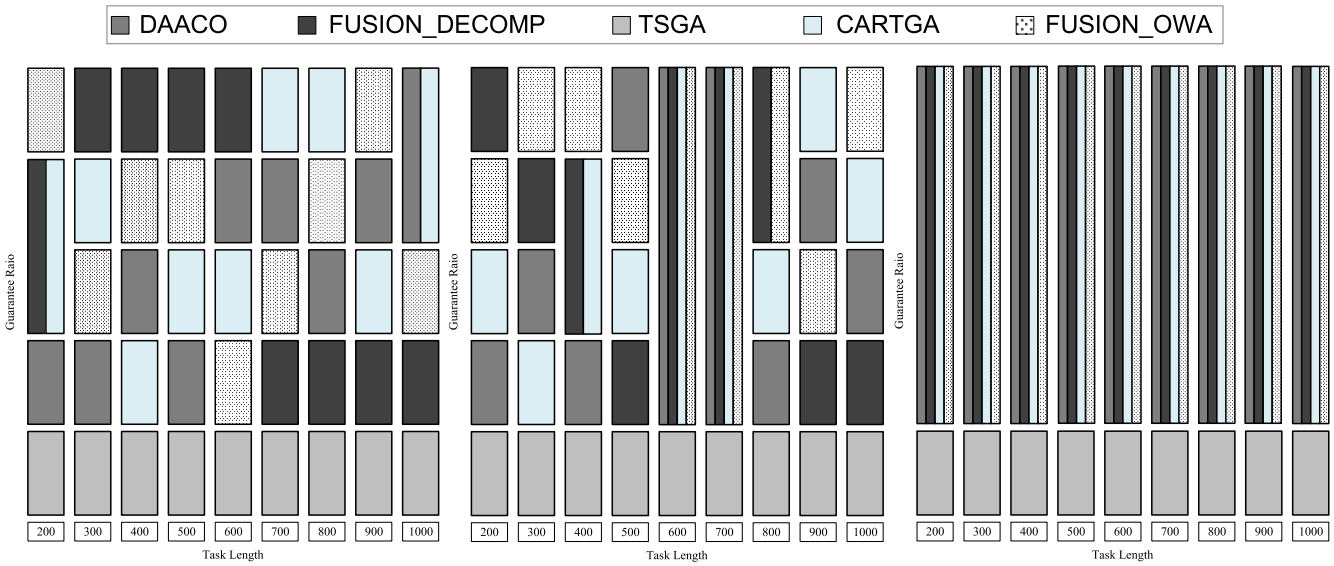
1) ***Job Makespan***: The time interval between the submission of job's first task to the finishing time of its last task (equation (4)).
2) ***Total Makespan***: The task (from any job) that has the highest finishing time determines the total time taken to execute jobs.
3) ***Total Guarantee Ratio***: The total number of tasks that have met their deadline out of all the tasks from all the jobs (equation (9)).
4) ***Average Utilization***: The average ratio of the VMs' busy time periods over the maximum working time. This metric shows how effectively the mapping algorithm could distribute tasks to VMs to improve both makespan and guarantee ratio (equation (11)).
5) ***Algorithm Execution Time***: The execution time of the algorithm's code. This metric is used to measure the time complexity of the implemented algorithms. In other words, it defines the elapsed time between the start of the algorithm and its termination.
6) ***Average Task Type***: The average number of tasks executed at each layer categorized by different deadline intervals and the total number of tasks. This metric is used to measure the performance of the offloader.

It is worthy to mention that the difference between the Job makespan and the total makespan is that in the former a task from the job, which has the highest finishing time determines the makespan, whereas in the latter a task among all the tasks and jobs, which has the highest finishing time determines the makespan. This means that for a job that has been finished last, the total makespan is equal to the job makespan. Therefore, other jobs have less makespan than the job that is finished last. Accordingly, reporting the results of total makespan is more effective when comparing the performance of algorithms. In FUSION we use the total makespan to measure the performance.

FUSION uses load balancing only when it helps the ranking of a task. Utilization alone may not be a suitable metric to compare algorithms, because there might be a powerful resource that behooves the algorithm to execute tasks on it, thus reducing the utilization. FUSION's main focus is to make tasks meet their deadlines in a less makespan, and utilization is used during the ranking process. Therefore, it is not necessary to report the results of the average utilization as a separate chart.

As we mentioned before, in FUSION, VMs can be ranked by employing two techniques (Decomposition and OWA). These two approaches are indicated with DECOMP, and OWA, respectively. Three state-of-the-art studies have been selected for comparison. We selected two deadline-aware algorithms [16] and [17], and a Genetic Algorithm (GA) that doesn't consider deadline [34]. The offloader module has been used for all the other methods. In this study, to simplify the comparison, we refer to the proposed method in [17] as Deadline-Aware scheduling algorithm based on Ant-Colony Optimization (DAACO), the proposed method in [16] as Cost-Aware Real-Time scheduling algorithm based on Genetic Algorithm (CARTGA), and the proposed method in [34] as an evolutionary Task Scheduling algorithm based on Genetic Algorithm (TSGA).

Accordingly, CARTGA uses the genetic algorithm, whose fitness function is the success ratio over the cost of usage [16].



(a) Deadline: 1-3 (s) (b) Deadline: 2-4 (s) (c) Deadline: 4-6 (s)

FIGURE 7. Evaluation and comparison of FUSION's guarantee ratio with state-of-the-art meta-heuristic algorithms.

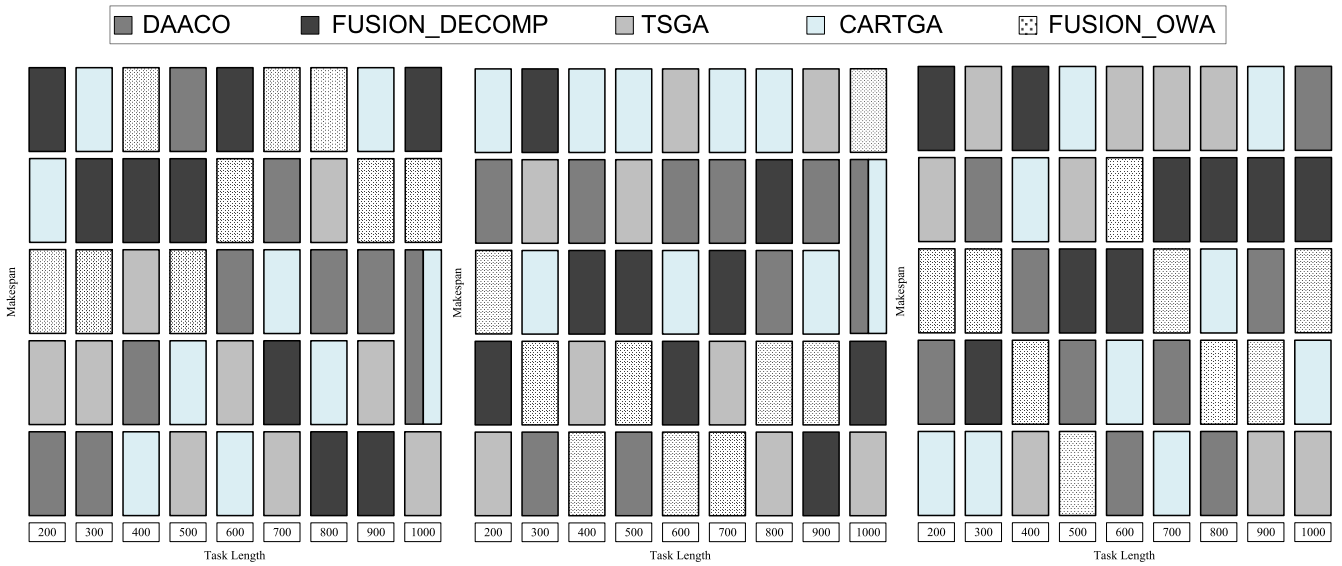
The cost of usage is obtained based on the amount of time of using memory and processing. In DAACO, authors employed the ant colony optimization. To calculate the probabilities, they used laxity as the main heuristic. The load balancing heuristic works by determining a score as a fraction of used resources over the total resources of a node [17]. Generally speaking, the objective of any task scheduling mechanism in fog computing is targeted at the benefit of users or service providers. From users' perspective, metrics such as makespan, budget, deadline, security, and cost are important. Meanwhile, makespan and response time are playing an important role for guaranteeing the QoS. On the other hand, since monetary costs are important for service providers, criterion such as load balancing, resource utilization, and energy efficiency gets more important. According to the above, a task scheduling technique, which minimizes completion time and saves monetary cost, will satisfy Service Level Agreement (SLA) signed by the users. This is the main reason that the authors in [34] have proposed TSGA; a genetic algorithm task scheduling algorithm with the aim of minimizing the makespan while maintaining a high utilization (to mitigate monetary costs). To reach this goal, the fitness function is considered as a weighted-sum of two fractions: the solution makespan, and the minimum makespan, and the solution utilization and the highest utilization possible [34]. It should be mentioned that to provide a fair comparison between FUSION and other techniques, all of the parameter settings for the existing algorithms (DAACO, CARTGA, and TSGA) are all gathered from their original papers.

B. EVALUATION RESULTS DISCUSSIONS

As mentioned earlier, the simulation results are based on 27 scenarios (3 deadline intervals and 9 different sets of task lists).

For the result visualizations, we used the mosaic plot or Marimekko chart. In this visualization, the algorithms are blocks or mosaics with different colors. A mosaic that is placed above any other mosaic has greater value in the vertical axis (metrics). If two or more mosaics are placed next to each other, it means they have the same value in the vertical axis. Using these charts allows us to simply compare our results both in general and relative manners. Considering the mosaic plots, we can compare the efficiency of FUSION with all the other implemented algorithms by simply counting the number of times FUSION's mosaic is on top of the others. In the same way, we can compare any two algorithms with respect to each other. The total number of tasks in each iteration (9 cases) constitutes the horizontal axis. Therefore, since we have three deadline intervals and nine different number of tasks, we have a total number of 27 cases for every metric of the comparison. The mosaic chart helps us to gather all of the required information in a single chart for every deadline interval. Fig. 7 shows the mosaic plot of the guarantee ratio. Any algorithm that stands on top of another algorithm is assumed to have a better guarantee ratio. According to Fig. 7, considering all the 27 scenarios, FUSION (DECOMP and OWA) have the highest guarantee ratio in 17 cases. The other two real-time algorithms (CARTGA [16] and DAACO [17]) have the highest guarantee ratio in 15 and 13 cases, respectively. The comparison can be seen in Table 5. In Fig. 7(c), we can see that all the real-time algorithms have executed all their tasks and have a guarantee ratio of one. This is because the deadline interval is large enough for the tasks to be finished before their deadlines.

Table 6 and Table 7 show the cases where two approaches of FUSION had a higher guarantee ratio compared with the other algorithms. The OWA approach had a higher guarantee



(a) Deadline: 1-3 (s) (b) Deadline: 2-4 (s) (c) Deadline: 4-6 (s)

FIGURE 8. Evaluation and comparison of FUSION's makespan with state-of-the-art meta-heuristic algorithms.

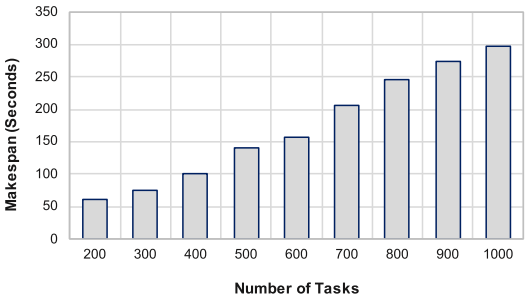


FIGURE 9. Relation between number of tasks and the expected makespan.

TABLE 5. Cases of highest guarantee ratio among 27 cases.

OWA	DECOMP	CARTGA	DAACO
17	17	15	13
63%	63%	55.6%	48%

TABLE 6. Comparison of the cases that OWA has a higher guarantee ratio against others.

OWA	CARTGA	OWA	DAACO	OWA	DECOMP
10	6	11	5	10	5

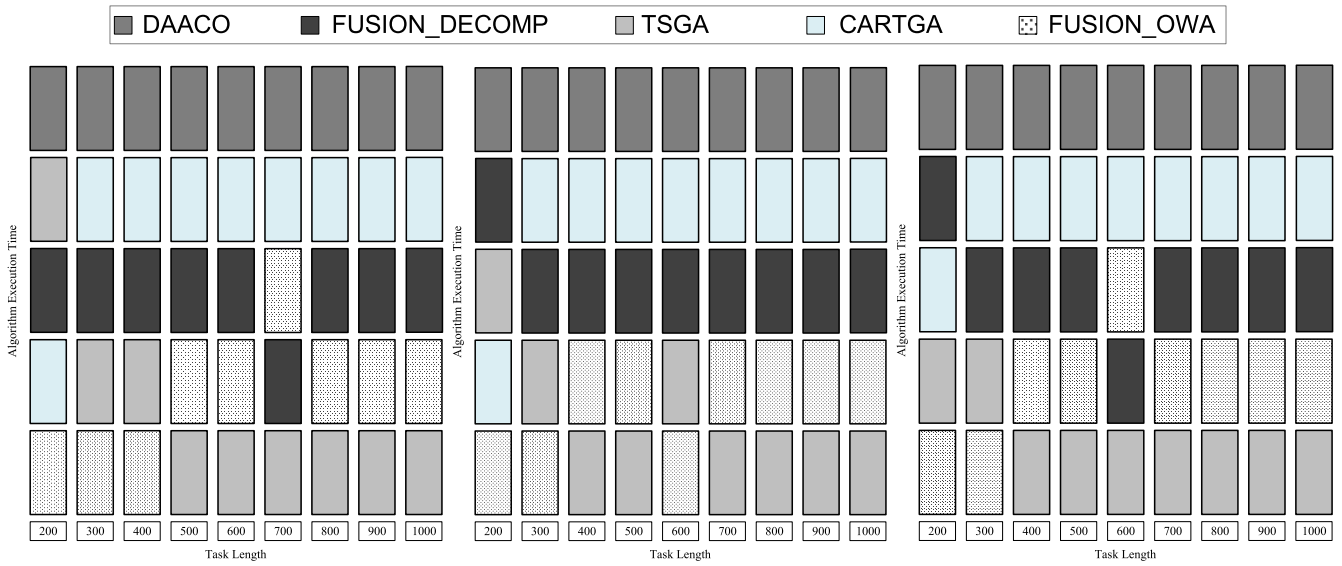
TABLE 7. Comparison of the cases that DECOMP has a higher guarantee ratio against others.

DECOMP	CARTGA	DECOMP	DAACO	DECOMP	OWA
7	7	9	7	5	10

ratio compared to CARTGA [16] in 10 cases, and in 11 cases compared to DAACO [17], while in only 6 and 5 cases OWA had a lower guarantee ratio compared to CARTGA [16] and DAACO [17], respectively. The FUSION DECOMP approach had a higher guarantee ratio in 9 cases, and a lower guarantee ratio in 7 cases compared to DAACO [17]. However, compared to CARTGA [16], both had a higher guarantee ratio in 7 cases.

Fig. 8 shows the comparison of makespan. Makespan depends on the guarantee ratio; because based on our simulation results represented in Fig. 9, it is expected that more number of executed tasks increases the makespan. In Fig. 8(c) where all the real-time algorithms experienced the same guarantee ratio, we can observe that CARTGA [16] had the lowest makespan in 3 cases. In other cases, OWA had a lower makespan compared to CARTGA [16] in 4 cases, and a higher makespan in 5 cases. In Fig. 8(b) we can observe that in cases of tasks size 400, 600, 700 and 800, that OWA has the highest guarantee ratio, its makespan was lower than DECOMP, DAACO [17], and CARTGA [16]. In the case of task size 300, OWA had a lower makespan compared to CARTGA [16] and DECOMP. This means despite having a higher guarantee ratio, OWA could execute tasks in a lower makespan compared to other algorithms.

To summarize the above, let's focus on the main objective of FUSION, which is to increase the guarantee ratio with the least possible makespan. According to Fig. 7, one can observe that by assuming longer deadlines for the tasks, all of the techniques behave relatively the same as all of them have the chance to schedule and execute as many tasks as possible before their deadlines. Nevertheless, when the deadlines are short, different techniques act differently (This scenario is much more realistic for real-time applications). In such cases, when the number of tasks is relatively low



(a) Deadline: 1-3 (s) (b) Deadline: 2-4 (s) (c) Deadline: 4-6 (s)

FIGURE 10. Evaluation and comparison of FUSION's execution time with state-of-the-art meta-heuristic algorithms.

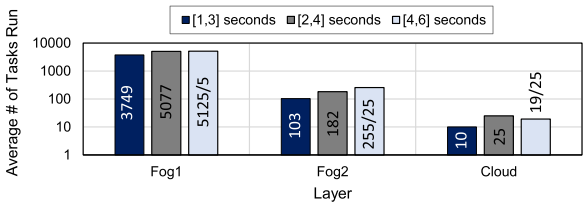


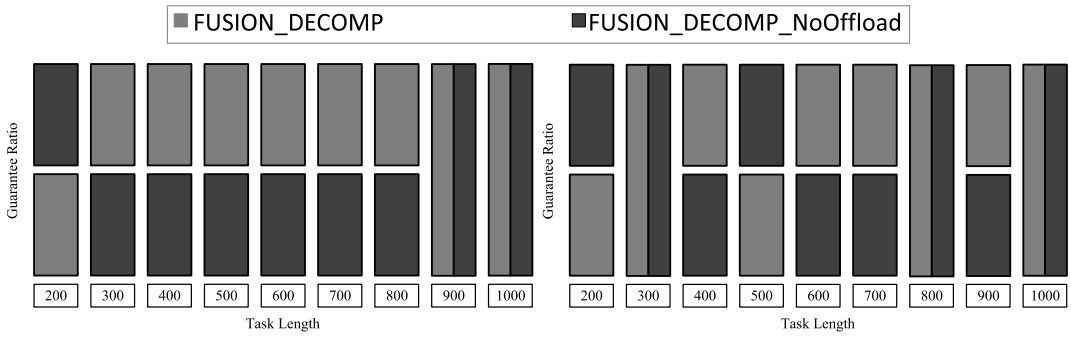
FIGURE 11. Average number of tasks processed at each layer.

(200-600), the DECOMP has provided a higher guarantee ratio compared to all of the other mechanisms. The OWA approach also resides in second place in terms of guarantee ratio. On the other hand, when the number of tasks gets higher (>700), OWA provides a better guarantee ratio compared with the DECOMP approach. This evaluation indicates that for large task sets, we need to use the OWA version of the FUSION. On the other hand, when the number of executed tasks increases, it is expected that the makespan will also increase. So, according to Fig. 8, while FUSION has provided a relatively higher makespan among the existing techniques, according to Fig. 7, it has executed more real-time tasks before their deadlines (especially when the tasks have short deadlines).

Fig. 10 shows the algorithm execution time. We measured the execution time from the moment the experiment started until its termination. Since the proposed algorithm by TSGA [34] didn't consider real-time tasks, it performed very poorly during task assignment and resulted in fast termination. It's clearly shown that FUSION (OWA and DECOMP) perform faster and have a lower algorithm execution time. Because CARTGA [16] and DAACO [17] use meta-heuristic algorithms to find the solution. These algorithms' execution time depends on the initial solutions, the underlying operations of moving toward the goal, the evaluation function, and the number of iterations. Therefore, they are expected to take more time with respect to fuzzy ranking methods.

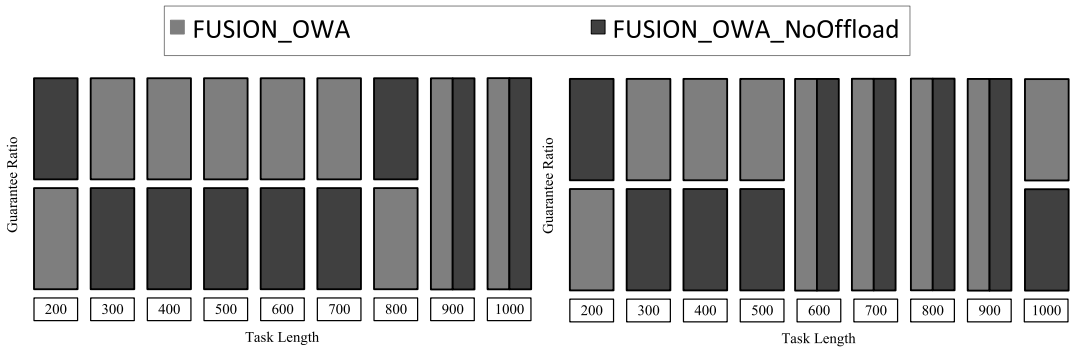
Fig. 11 show the average number of tasks assigned to each layer for execution categorized by each deadline interval and task size. Out of the total 5400 tasks, we can observe the average number of tasks executed at the first fog layer increases with the increase in deadline interval. It also shows how the scheduling and offloading algorithms manage to utilize the first fog layer (closest to IoT devices) to execute tasks in order to have less latency and communication delay. Average Task Type is used to analyze the behavior of FUSION's offloader. It's not always beneficial to offload almost every task to a node with higher computational capacity. FUSION's offloader deals with this issue by considering latency, task size, bandwidth, and the C_i/P_i.

Now, we compare FUSION (OWA and DECOMP) with and without offloader. Generally speaking, larger tasks increase the computation delay (execution time), and communication delay, which can affect the start time of other tasks, leading to higher makespan and more miss rate of deadlines [86]. Nevertheless, in long run perspective, it is expected that offloading larger tasks *in advance* could be more beneficial against offloading them at last, in terms of both throughput and makespan. Fig. 12 and Fig. 13 show the benefit of using offloader to achieve a higher guarantee ratio. In the DECOMP approach we can observe that using offloading is 55.6% beneficial and results in a higher guarantee ratio. In the OWA approach it can be seen that using offloader is 50% beneficial and results in a



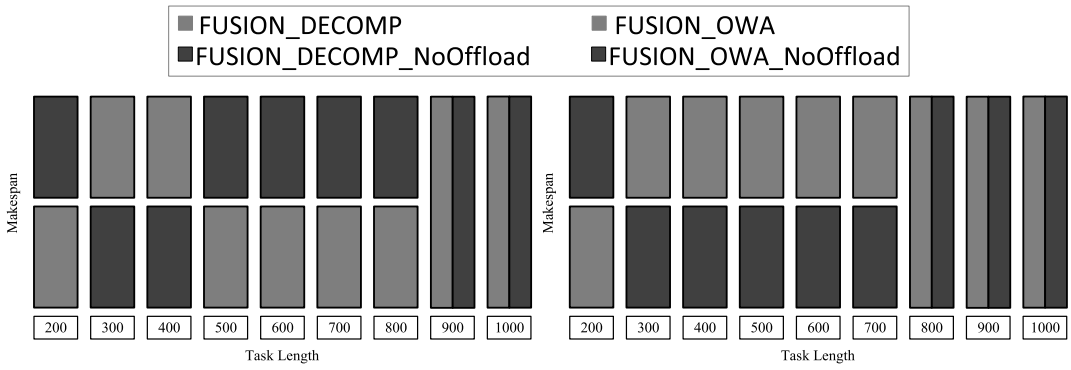
(a) Deadline: 1-3 (s) (b) Deadline: 2-4 (s)

FIGURE 12. Guarantee ratio evaluation in case of using offloader in DECOMP approaches.



(a) Deadline: 1-3 (s) (b) Deadline: 2-4 (s)

FIGURE 13. Guarantee ratio evaluation in case of using offloader in OWA approaches.



(a) FUSION_DECOMP in Deadline 4-6 (s) (b) FUSION_OWA in Deadline 4-6 (s)

FIGURE 14. Makespan evaluation in case of using offloader in DECOMP, and OWA approaches.

higher guarantee ratio. In the case of 4-6 seconds of deadline interval, using or not using offloader produced the same results, therefore we compared their makespan. Fig. 14 shows the makespan comparison of using and not using offloader.

In the DECOMP approach we can see that using offloader could improve the makespan in 55.6% of cases in the last deadline interval. Whereas in the OWA method we do not observe this trend. Surely, this doesn't mean that we must not use the offloader when using OWA just because the result showed that OWA had a higher makespan in the last deadline interval. Instead, the results showed that OWA's guarantee ratio has improved using the offloader in all deadline intervals. Therefore, the main objective is to achieve the highest guarantee ratio possible.

C. DISCUSSION ON COMPLEXITY AND SCALABILITY

In GA there are parameters such as: iterations, initial population, mutation and cross-over rates and weight parameters

used for weighted sum fitness function according to [87]. All the mentioned parameters need to be determined beforehand. For instance, having low number of iterations can result in bad results. The Mutation rate indicates the rate of generating diversity among chromosomes. The Cross-Over rate indicates how often a new chromosome resembling its parents is added to the population. Therefore, having a lower mutation rate means less diversity and less unexplored results. Having a lower cross-over rates means less mating between parents and less potential offspring with a higher fitness value.

As for the fitness function with weighted-sum approach, the weight parameters determines the fitness of chromosomes. Having poorly-tuned weight parameters can lead to selection of undesirable chromosomes. Mostly, these parameters are obtained using trial and error. This conditions also hold for other meta-heuristic algorithms such as: Particle Swarm Optimization (PSO) and Ant-Colony Optimization (ACO) optimizations. Both have parameters that need to be determined beforehand. For instance, in ACO there are parameters such as: number of ants, maximum iterations, exponents for the local heuristic and pheromone values, global pheromone update and initial pheromone values according [17]. In PSO, there are parameters such as: initial positions, initial population, number of iterations, weight parameters of weighted-sum fitness evaluation function and random coefficients of global and personal best solutions according to [88]. All of these issues impose complexity and increase the number of operations.

On the other hand, in FUSION, the input and output membership functions can be defined without having to tune any parameters beforehand. And sometimes membership functions of other problems can be re-used. The weights in the OWA approach are determined regardless of the problem, from the membership function of the relative quantifier which in our case is *Almost All* and is the same in most problems. The operations of computing ranks and ordered weighted averaging is much simpler with respect to evaluations in GA, PSO and ACO. Since the values which are used to compute ranks can be simply obtained from the membership function of that fuzzy predicate.

In summary, according to the execution time results (Fig. 10), the evolutionary methods require more time to complete execution, while FUSION outperforms them in this criterion. The reason for this is that the number of objectives in our case is relatively small (three), and all the membership functions for the fuzzy predicates, whether input or output, are predefined. Furthermore, the membership function for the quantifier in our problem is considered somewhat universal, and the ranking is based on finding a weighted-sum value. Consequently, the overall complexity is significantly lower than that of the evolutionary algorithms, which require predefined and tuned parameters.

It is worthy to mention that evolutionary algorithms are iterative, and each iteration involves multiple operations on the solutions within the search space. Traversing the search space to find a near-optimal solution in these algorithms is time-consuming and complex (depending on the algorithm type). In contrast, the proposed method's implementation is simple, and the number of operations is not as extensive as in evolutionary algorithms, resulting in reduced execution time.

Regarding the issue of scalability, FUSION is scalable in nature. Fuzzy-logic-based offloading mechanisms can be scalable in real-world complex fog computing systems due to several factors, including:

1) **Adaptive Decision Making:** Fuzzy logic solutions are particularly adept at managing uncertainty and imprecision, which are prevalent in dynamic fog computing environments. They can flexibly determine which tasks to offload by considering current network conditions, resource availability, and task requirements.
2) **Efficient Resource Allocation:** FUSION enhances resource allocation by simultaneously considering factors like latency, deadline, and load. This approach ensures efficient resource use, even as the system scales with more devices and tasks.
3) **Scalability through Hierarchical Structures:** The hierarchical structure of FUSION allows local fog nodes to make immediate offloading decisions, while higher-level nodes oversee broader resource distribution. Applying fuzzy logic at each level ensures both efficiency and scalability.
4) **Robust Failure Handling:** Fuzzy logic boosts the robustness of fog computing systems by gracefully managing failures. It dynamically adjusts offloading thresholds and redistributes tasks to prevent system overloads.

Overall, research indicates that fuzzy-logic-based offloading mechanisms enhance performance in real-world applications like healthcare, augmented reality, and IoT systems. These improvements include reduced network latency, shorter computation delays, and overall better system performance. Accordingly, FUSION can efficiently expand within intricate fog computing frameworks, maintaining effective and dependable performance as the system scales.

VI. CONCLUSION AND FUTURE STUDIES

In this paper, we introduced FUSION, a multi-objective decision making fuzzy algorithm for scheduling tasks in IoT networks. FUSION benefits from a fuzzy offloader module as a tool to help the scheduling algorithm achieve lower makespan and higher guarantee ratio. FUSION is less complex in terms of implementation and execution time. The results showed that FUSION, on average, had the highest guarantee ratio in 17 cases out of 27. The FUSION OWA approach resulted in lower makespan in cases where it achieved the highest guarantee ratio. In terms of algorithm execution time, in 89% of cases among 27 cases, FUSION terminated earlier than others. Using the FUSION offloader resulted in higher guarantee ratio in at least 50% of cases (among 18 cases), whereas not using it resulted in having higher guarantee ratio in 16.7% of cases (among 18 cases).

Furthermore, using FUSION offloader improved makespan in the FUSION DECOMP approach.

As part of our future studies, we will try to improve the FUSION fuzzy offloader rule base and decision making. Since we consider real-time tasks, one approach is to implement the Earliest-Deadline-First (EDF) algorithm on the cloudlet scheduler of VMs to ensure schedulability. We also aim to utilize multi-objective meta-heuristic algorithms that consider guarantee ratio, makespan, and utilization directly in their objective function. Furthermore, we also aim to consider other objectives such as: financial cost and energy consumption. Finally, considering a cluster of fog nodes managed by a resource-full orchestrator equipped with a machine learning scheduler is another field of study that is aimed for a potential field of study in the future. Additionally, ongoing research and development in fog computing are continually providing new insights and techniques to improve latency and persistence. One of these approaches, which is a potential field of research in the future is edge intelligence. Edge intelligence allows for data to be processed and analyzed locally, at or near the source, which means only relevant data is sent to the cloud, reducing the volume of data transmission and associated delays. The edge intelligence, along with the distributed nature of fog infrastructures allows for more efficient data processing and quicker response times, leading into less latency and maximization of communication routes. Not to forget that evaluating all of the mentioned approaches along with more recent studies could be conducted on a real-world test-bed to obtain more precise results.

REFERENCES

[1] L. Bittencourt, R. Immich, R. Sakellariou, N. Fonseca, E. Madeira, M. Curado, L. Villas, L. DaSilva, C. Lee, and O. Rana, "The Internet of Things, fog and cloud continuum: Integration and challenges," *Internet Things*, vols. 3–4, pp. 134–155, Oct. 2018.

[2] M. Mukherjee, L. Shu, and D. Wang, "Survey of fog computing: Fundamental, network applications, and research challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 1826–1857, 3rd Quart., 2018.

[3] M. Aazam and E.-N. Huh, "Fog computing: The cloud-IoTIoE middleware paradigm," *IEEE Potentials*, vol. 35, no. 3, pp. 40–44, May 2016.

[4] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. MCC workshop Mobile cloud Comput.*, Aug. 2012, pp. 13–16.

[5] M. L. M. Peixoto, T. A. L. Genez, and L. F. Bittencourt, "Hierarchical scheduling mechanisms in multi-level fog computing," *IEEE Trans. Services Comput.*, vol. 15, no. 5, pp. 2824–2837, Sep. 2022.

[6] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the suitability of fog computing in the context of Internet of Things," *IEEE Trans. Cloud Comput.*, vol. 6, no. 1, pp. 46–59, Jan. 2018.

[7] C. Perera, Y. Qin, J. C. Estrella, S. Reiff-Marganiec, and A. V. Vasilakos, "Fog computing for sustainable smart cities: A survey," *ACM Comput. Surv.*, vol. 50, no. 3, pp. 1–43, Jun. 2017.

[8] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[9] Y. Cao, S. Chen, P. Hou, and D. Brown, "FAST: A fog computing assisted distributed analytics system to monitor fall for stroke mitigation," in *Proc. IEEE Int. Conf. Netw., Archit. Storage (NAS)*, Aug. 2015, pp. 2–11.

[10] T. N. Gia, M. Jiang, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fog computing in healthcare Internet of Things: A case study on ECG feature extraction," in *Proc. IEEE Int. Conf. Comput. Inf. Technol.; Ubiquitous Comput. Commun.; Dependable, Autonomic Secure Comput.; Pervasive Intell. Comput.*, Oct. 2015, pp. 356–363.

[11] M.-P. Hosseini, A. Hajisami, and D. Pompili, "Real-time epileptic seizure detection from EEG signals via random subspace ensemble learning," in *Proc. IEEE Int. Conf. Autonomic Comput. (ICAC)*, Jul. 2016, pp. 209–218.

[12] X. Masip-Bruin, E. Marín-Tordera, A. Alonso, and J. Garcia, "Fog-to-cloud computing (F2C): The key technology enabler for dependable e-health services deployment," in *Proc. Medit. Ad Hoc Netw. Workshop (Med-Hoc-Net)*, Jun. 2016, pp. 1–5.

[13] C.-Y. Huang, C.-H. Hsu, Y.-C. Chang, and K.-T. Chen, "GamingAnywhere: An open cloud gaming system," in *Proc. 4th ACM Multimedia Syst. Conf.*, Feb. 2013, pp. 36–47.

[14] C. Reiss and A. Tumanov, "Heterogeneity and dynamicity of clouds at scale: Google trace analysis," in *Proc. ACM Symp. Cloud Comput.*, Nov. 2021, pp. 1–13.

[15] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading tasks with dependency and service caching in mobile edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2777–2792, Nov. 2021.

[16] T. S. Nikoui, A. Balador, A. M. Rahmani, and Z. Bakhshi, "Cost-aware task scheduling in fog-cloud environment," in *Proc. CSI/CPSSI Int. Symp. Real-Time Embedded Syst. Technol. (RTEST)*, Jun. 2020, pp. 1–8.

[17] J. Fan, X. Wei, T. Wang, T. Lan, and S. Subramaniam, "Deadline-aware task scheduling in a tiered IoT infrastructure," in *Proc. IEEE Global Commun. Conf.*, Dec. 2017, pp. 1–7.

[18] J. Fan, J. Liu, J. Chen, and J. Yang, "LPDC: Mobility-and deadline-aware task scheduling in tiered IoT," in *Proc. IEEE 4th Int. Conf. Comput. Commun. (ICCC)*, Dec. 2018, pp. 857–863.

[19] M. D. Hossain, T. Sultana, V. Nguyen, W. U. Rahman, T. D. T. Nguyen, L. N. T. Huynh, and E.-N. Huh, "Fuzzy based collaborative task offloading scheme in the densely deployed small-cell networks with multi-access edge computing," *Appl. Sci.*, vol. 10, no. 9, p. 3115, Apr. 2020.

[20] D. Zhou, F. Chao, C.-M. Lin, L. Yang, M. Shi, and C. Zhou, "Integration of fuzzy CMAC and BELC networks for uncertain nonlinear system control," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2017, pp. 1–6.

[21] L. Abdullah, "Fuzzy multi criteria decision making and its applications: A brief review of category," *Proc.-Social Behav. Sci.*, vol. 97, pp. 131–136, Nov. 2013.

[22] J. An, M. Hu, L. Fu, and J. Zhan, "A novel fuzzy approach for combining uncertain conflict evidences in the Dempster–Shafer theory," *IEEE Access*, vol. 7, pp. 7481–7501, 2019.

[23] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "IFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw., Pract. Exper.*, vol. 47, no. 9, pp. 1275–1296, Sep. 2017.

[24] H. Chen, F. Wang, N. Helian, and G. Akanmu, "User-priority guided min-min scheduling algorithm for load balancing in cloud computing," in *Proc. Nat. Conf. Parallel Comput. Technol. (PARCOMPTECH)*, Feb. 2013, pp. 1–8.

[25] Y. Mao, X. Chen, and X. Li, "Max–min task scheduling algorithm for load balance in cloud computing," in *Proc. Int. Conf. Comput. Sci. Inf. Technol.* Cham, Switzerland: Springer, 2014, pp. 457–465.

[26] R. K. Mondal, E. Nandi, and D. Sarddar, "Load balancing scheduling with shortest load first," *Int. J. Grid Distrib. Comput.*, vol. 8, no. 4, pp. 171–178, Aug. 2015.

[27] D. C. Devi and V. R. Uthariaraj, "Load balancing in cloud computing environment using improved weighted round Robin algorithm for nonpreemptive dependent tasks," *Sci. World J.*, vol. 2016, pp. 1–14, Jan. 2016.

[28] Z. Cheng, P. Li, J. Wang, and S. Guo, "Just-in-time code offloading for wearable computing," *IEEE Trans. Emerg. Topics Comput.*, vol. 3, no. 1, pp. 74–83, Mar. 2015.

[29] M. Mukherjee, S. Kumar, M. Shojafar, Q. Zhang, and C. X. Mavromoustakis, "Joint task offloading and resource allocation for delay-sensitive fog networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–7.

[30] K. Li, G. Xu, G. Zhao, Y. Dong, and D. Wang, "Cloud task scheduling based on load balancing ant colony optimization," in *Proc. 6th Annu. Chinagrid Conf.*, Aug. 2011, pp. 3–9.

[31] T. Sen and H. Shen, "Machine learning based timeliness-guaranteed and energy-efficient task assignment in edge computing systems," in *Proc. IEEE 3rd Int. Conf. Fog Edge Comput. (ICFEC)*, May 2019, pp. 1–10.

[32] R. Siyadatzadeh, F. Mehrafrooz, M. Ansari, B. Safaei, M. Shafique, J. Henkel, and A. Ejlali, "ReLIEF: A reinforcement learning-based real-time task assignment strategy in emerging fault-tolerant fog computing," *IEEE Internet Things J.*, p. 1, 2023.

[33] M. K. Hussein and M. H. Mousa, "Efficient task offloading for IoT-based applications in fog computing using ant colony optimization," *IEEE Access*, vol. 8, pp. 37191–37201, 2020.

[34] H. T. T. Binh, T. T. Anh, D. B. Son, P. A. Duc, and B. M. Nguyen, "An evolutionary algorithm for solving task scheduling problem in cloud-fog computing environment," in *Proc. 9th Int. Symp. Inf. Commun. Technol. - SoICT*, 2018, pp. 397–404.

[35] S. Ghanavati, J. Abawajy, and D. Izadi, "An energy aware task scheduling model using ant-mating optimization in fog computing environment," *IEEE Trans. Services Comput.*, vol. 15, no. 4, pp. 2007–2017, Jul. 2022.

[36] Y.-D. Lin, Y.-C. Lai, J.-X. Huang, and H.-T. Chien, "Three-tier capacity and traffic allocation for core, edges, and devices for mobile edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 15, no. 3, pp. 923–933, Sep. 2018.

[37] A. Samanta, F. Esposito, and T. G. Nguyen, "Fault-tolerant mechanism for edge-based IoT networks with demand uncertainty," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16963–16971, Dec. 2021.

[38] X. Hou, J. Wang, Z. Fang, Y. Ren, K.-C. Chen, and L. Hanzo, "Edge intelligence for mission-critical 6G services in space-air-ground integrated networks," *IEEE Netw.*, vol. 36, no. 2, pp. 181–189, Mar. 2022.

[39] P. Zhang and M. Zhou, "Dynamic cloud task scheduling based on a two-stage strategy," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 2, pp. 772–783, Apr. 2018.

[40] D. Zeng, L. Gu, S. Guo, Z. Cheng, and S. Yu, "Joint optimization of task scheduling and image placement in fog computing supported software-defined embedded system," *IEEE Trans. Comput.*, vol. 65, no. 12, pp. 3702–3712, Dec. 2016.

[41] S. Chen, Y. Zheng, K. Wang, and W. Lu, "Delay guaranteed energy-efficient computation offloading for industrial IoT in fog computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2019, pp. 1–6.

[42] M. Kumar, G. K. Walia, H. Shingare, S. Singh, and S. S. Gill, "AI-based sustainable and intelligent offloading framework for IIoT in collaborative cloud-fog environments," *IEEE Trans. Consum. Electron.*, vol. 70, no. 1, pp. 1414–1422, Feb. 2024.

[43] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang, and A. X. Liu, "QoS driven task offloading with statistical guarantee in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 1, pp. 278–290, Jan. 2022.

[44] X. Tao, K. Ota, M. Dong, H. Qi, and K. Li, "Performance guaranteed computation offloading for mobile-edge cloud computing," *IEEE Wireless Commun. Lett.*, vol. 6, no. 6, pp. 774–777, Dec. 2017.

[45] K. Lone and S. A. Sofi, "Cost efficient task offloading for delay sensitive applications in fog computing system," *Social Netw. Comput. Sci.*, vol. 4, no. 6, p. 817, Oct. 2023.

[46] M. Maashi, E. Alabdulkreem, M. Maray, K. Shankar, A. A. Darem, A. Alzahrani, and I. Yaseen, "Elevating survivability in next-gen IoT-fog-cloud networks: Scheduling optimization with the metaheuristic mountain gazelle algorithm," *IEEE Trans. Consum. Electron.*, vol. 70, no. 1, pp. 3802–3809, Feb. 2024.

[47] A. Ali, S. A. A. Shah, T. Al Shloul, M. Assam, Y. Yasin, S. Lim, and A. Zia, "Multi-objective Harris hawks optimization based task scheduling in cloud-fog computing," *IEEE Internet Things J.*, vol. 11, no. 3, pp. 24334–24352, Jul. 2024.

[48] Y. Seraj, S. Fadaei, B. Safaei, A. Javadi, A. M. H. Monazzah, and A. M. A. Hemmatyar, "LIMO: Load-balanced offloading with MAPE and particle swarm optimization in mobile fog networks," 2024, *arXiv:2408.14218*.

[49] B. Ranjbar, B. Safaei, A. Ejlali, and A. Kumar, "FANTOM: Fault tolerant task-drop aware scheduling for mixed-criticality systems," *IEEE Access*, vol. 8, pp. 187232–187248, 2020.

[50] N. Saini and S. Saha, "Multi-objective optimization techniques: A survey of the state-of-the-art and applications: Multi-objective optimization techniques," *Eur. Phys. J. Special Topics*, vol. 230, no. 10, pp. 2319–2335, Sep. 2021.

[51] S. Dabia, E.-G. Talbi, T. van Woensel, and T. De Kok, "Approximating multi-objective scheduling problems," *Comput. Oper. Res.*, vol. 40, no. 5, pp. 1165–1175, May 2013.

[52] B. Safaei, A. M. H. Monazzah, M. B. Bafroei, and A. Ejlali, "Reliability side-effects in Internet of Things application layer protocols," in *Proc. 2nd Int. Conf. Syst. Rel. Saf. (ICSRS)*, Dec. 2017, pp. 207–212.

[53] R. Kaur, V. Laxmi, and Balkrishan, "Performance evaluation of task scheduling algorithms in virtual cloud environment to minimize makespan," *Int. J. Inf. Technol.*, vol. 14, no. 1, pp. 79–93, Feb. 2022.

[54] L. Zhang, M. Ai, R. Tan, J. Man, X. Deng, and K. Li, "Efficient prediction of makespan matrix workflow scheduling algorithm for heterogeneous cloud environments," *J. Grid Comput.*, vol. 21, no. 4, p. 75, Dec. 2023.

[55] A. Motamedhashemi, B. Safaei, A. M. H. Monazzah, and A. Ejlali, "DATA: Throughput and deadline-aware genetic approach for task scheduling in fog networks," *IEEE Embedded Syst. Lett.*, p. 1, 2024.

[56] M. Mazouzi, K. Mershad, O. Cheikhrouhou, and M. Hamdi, "Agent-based reactive geographic routing protocol for Internet of Vehicles," *IEEE Access*, vol. 11, pp. 79954–79973, 2023.

[57] S. Yaqoob, A. Ullah, M. Akbar, M. Imran, and M. Shoaib, "Congestion avoidance through fog computing in Internet of Vehicles," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 10, pp. 3863–3877, Oct. 2019.

[58] M. Nakahara, D. Hisano, M. Nishimura, Y. Ushiku, K. Maruta, and Y. Nakayama, "Retransmission edge computing system conducting adaptive image compression based on image recognition accuracy," in *Proc. IEEE 94th Veh. Technol. Conf. (VTC-Fall)*, Sep. 2021, pp. 1–5.

[59] P. K. Donta, S. N. Srirama, T. Amgoth, and C. S. R. Annavarapu, "Survey on recent advances in IoT application layer protocols and machine learning scope for research directions," *Digit. Commun. Netw.*, vol. 8, no. 5, pp. 727–744, Oct. 2022.

[60] P. K. Donta, T. Amgoth, and C. S. Rao Annavarapu, "Congestion-aware data acquisition with Q-learning for wireless sensor networks," in *Proc. IEEE Int. IoT, Electron. Mechatronics Conf. (IEMTRONICS)*, Sep. 2020, pp. 1–6.

[61] P. Bedi and P. Khurana, "Sentiment analysis using fuzzy-deep learning," in *Proceedings of ICETIT 2019: Emerging Trends in Information Technology*. Cham, Switzerland: Springer, 2020, pp. 246–257.

[62] K. Singh and A. K. Verma, "TBCS: A trust based clustering scheme for secure communication in flying ad-hoc networks," *Wireless Pers. Commun.*, vol. 114, no. 4, pp. 3173–3196, Oct. 2020.

[63] M. J. dos Santos and E. A. de M. Fagotto, "Cloud computing management using fuzzy logic," *IEEE Latin Amer. Trans.*, vol. 13, no. 10, pp. 3392–3397, Oct. 2015.

[64] S. Mehamel, K. Slimani, S. Bouzefrane, and M. Daoui, "Energy-efficient hardware caching decision using fuzzy logic in mobile edge computing," in *Proc. 6th Int. Conf. Future Internet Things Cloud Workshops (FiCloudW)*, Aug. 2018, pp. 237–242.

[65] N. Dhanya, G. Kousalya, P. Balarksihnan, and P. Raj, "Fuzzy-logic-based decision engine for offloading IoT application using fog computing," in *Handbook of Research on Cloud and Fog Computing Infrastructures for Data Science*. Hershey, PA, USA: IGI Global, 2018, pp. 175–194.

[66] F. Basic, A. Aral, and I. Brandic, "Fuzzy handoff control in edge offloading," in *Proc. IEEE Int. Conf. Fog Comput. (ICFC)*, Jun. 2019, pp. 87–96.

[67] M. A. Benblidia, B. Brik, L. Merghem-Boulahia, and M. Esseghir, "Ranking fog nodes for tasks scheduling in fog-cloud environments: A fuzzy logic approach," in *Proc. 15th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2019, pp. 1451–1457.

[68] O. K. Cu and P. R. K. Sathia Bhama, "Fuzzy based energy efficient workload management system for flash crowd," *Comput. Commun.*, vol. 147, pp. 225–234, Nov. 2019.

[69] B. Igried, A. Alsarhan, I. Al-Khawaldeh, A. Al-Qerem, and A. Aldweesh, "A novel fuzzy logic-based scheme for malicious node eviction in a vehicular ad hoc network," *Electronics*, vol. 11, no. 17, p. 2741, Aug. 2022.

[70] I. Jabri, T. Mekki, A. Rachedi, and M. B. Jemaa, "Vehicular fog gateways selection on the Internet of Vehicles: A fuzzy logic with ant colony optimization based approach," *Ad Hoc Netw.*, vol. 91, Aug. 2019, Art. no. 101879.

[71] L. A. Zadeh, "Fuzzy logic," *Computer*, vol. 21, no. 4, pp. 83–93, Apr. 1988.

[72] H. K. Lee, E. Paillet, and W. Peeters, "A consistency criterion for optimizing defuzzification in fuzzy control," in *Foundations of Generic Optimization: Applications of Fuzzy Control, Genetic Algorithms and Neural Networks*, vol. 2, 2008, pp. 403–431.

[73] M. Štěpnička, U. Bodenhofer, M. Daňková, and V. Novák, "Continuity issues of the implicational interpretation of fuzzy rules," *Fuzzy Sets Syst.*, vol. 161, no. 14, pp. 1959–1972, Jul. 2010.

[74] S. Ganguly and D. Samajpati, "Distributed generation allocation on radial distribution networks under uncertainties of load and generation using genetic algorithm," *IEEE Trans. Sustain. Energy*, vol. 6, no. 3, pp. 688–697, Jul. 2015.

[75] D. G. Burkhardt and P. P. Bonissone, "Automated fuzzy knowledge base generation and tuning," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Mar. 1992, pp. 179–188.
[76] Z. Huang and Q. Shen, "A new fuzzy interpolative reasoning method based on center of gravity," in *Proc. 12th IEEE Int. Conf. Fuzzy Syst.*, vol. 15, Oct. 2003, pp. 25–30.
[77] V. Nguyen, T. T. Khanh, T. D. T. Nguyen, C. S. Hong, and E.-N. Huh, "Flexible computation offloading in a fuzzy-based mobile edge orchestrator for IoT applications," *J. Cloud Comput.*, vol. 9, no. 1, pp. 1–18, Dec. 2020.
[78] R. R. Yager, "General multiple-objective decision functions and linguistically quantified statements," *Int. J. Man-Mach. Stud.*, vol. 21, no. 5, pp. 389–400, Nov. 1984.
[79] R. R. Yager, "Constrained OWA aggregation," *Fuzzy Sets Syst.*, vol. 81, no. 1, pp. 89–101, Jul. 1996.
[80] H. Zeng, M. Di Natale, A. Ghosal, and A. Sangiovanni-Vincentelli, "Schedule optimization of time-triggered systems communicating over the FlexRay static segment," *IEEE Trans. Ind. Informat.*, vol. 7, no. 1, pp. 1–17, Feb. 2011.
[81] J. Meng, H. Tan, X.-Y. Li, Z. Han, and B. Li, "Online deadline-aware task dispatching and scheduling in edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 6, pp. 1270–1286, Jun. 2020.
[82] A. S. Alfakeeh and M. A. Javed, "Intelligent data-enabled task offloading for vehicular fog computing," *Appl. Sci.*, vol. 13, no. 24, p. 13034, Dec. 2023.
[83] R. Buyya, R. Ranjan, and R. N. Calheiros, "Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: Challenges and opportunities," in *Proc. Int. Conf. High Perform. Comput. Simul.*, Jun. 2009, pp. 1–11.
[84] P. Cingolani and J. Alcalá-Fdez, "JFuzzyLogic: A Java library to design fuzzy logic controllers according to the standard for fuzzy control programming," *Int. J. Comput. Intell. Syst.*, vol. 6, no. 1, p. 61, 2013.
[85] P. Cingolani and J. Alcalá-Fdez, "JFuzzyLogic: A robust and flexible fuzzy-logic inference system language implementation," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, Jun. 2012, pp. 1–8.
[86] Z. Xue, C. Liu, C. Liao, G. Han, and Z. Sheng, "Joint service caching and computation offloading scheme based on deep reinforcement learning in vehicular edge computing systems," *IEEE Trans. Veh. Technol.*, vol. 72, no. 5, pp. 6709–6722, May 2023.
[87] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: A tutorial," *Rel. Eng. Syst. Saf.*, vol. 91, no. 9, pp. 992–1007, Sep. 2006.
[88] S. Mostaghim and J. Teich, "Strategies for finding good local guides in multi-objective particle swarm optimization (MOPSO)," in *Proc. IEEE Swarm Intell. Symp.*, Apr. 2003, pp. 26–33.

ARYA MOTAMEDHASHEMI received the B.Sc. and M.Sc. degrees in computer engineering from the Sharif University of Technology (SUT), Tehran, Iran, in 2017 and 2022, respectively. Since 2019, he has been a member of the Embedded Systems Research Laboratory (ESRLab). As a Research Assistant, he is collaborating with the Reliable and Durable IoT Application and Networks (RADIAN) Laboratory, SUT. His research interests include task scheduling in the Internet of Things (IoT) and cloud/fog networks.

BARDIA SAFAEI received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2021. As a Ph.D. Visiting Researcher, he was with the Chair for Embedded Systems, Karlsruhe Institute of Technology, Germany, from 2019 to 2020. Currently, he is a Faculty Member with the Department of Computer Engineering, Sharif University of Technology, where he is the Director of the Reliable, and Durable IoT Applications and Networks (RADIAN) Laboratory. He has been a member of the National Elites Foundation, from 2016 to 2020. His research interests include power-efficiency and dependability challenges in the IoT, WSN, MANET, and fog computing. He received the ACM/SIGAPP Student Award in the 34th ACM/SIGAPP Symposium on Applied Computing (SAC'19). He has served as a Reviewer for prestigious international journals and conferences, including IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, *IEEE Communications Magazine*, IEEE WF-IoT, IEEE IoT JOURNAL, and IEEE TRANSACTIONS ON MOBILE COMPUTING.

AMIR MAHDI HOSSEINI MONAZZAH received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2017. He was a member of the Dependable Systems Laboratory, from 2010 to 2017. As a Visiting Researcher, he was with the Embedded Systems Laboratory, University of California, Irvine, CA, USA, from 2016 to 2017. As a Postdoctoral Fellow, he was with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, from 2017 to 2019. He is currently a Faculty Member with the School of Computer Engineering, Iran University of Science and Technology (IUST), Tehran, Iran. His research interests include investigating the challenges of emerging nonvolatile memories, hybrid memory hierarchy design, and IoT applications.

JÖRG HENKEL (Fellow, IEEE) is with Karlsruhe Institute of Technology. Previously, he was a Research Staff Member with the NEC Laboratories, Princeton, NJ. He has received six best paper awards, among others ICCAD, ESWeek, and DATE. He has led several conferences as a General Chair, including ICCAD and ESWeek. He serves as the steering committee chair/member for leading conferences and journals for embedded and cyber-physical systems. He coordinates the DFG Program SPP 1500 "Dependable Embedded Systems" and a site Coordinator of the DFG TR89 Collaborative Research Center "Invasive Computing." He is the Chairman of the IEEE Computer Society, Germany Chapter. He served as the Editor-in-Chief for *ACM Transactions on Embedded Computing Systems*, for two terms. He is currently the Editor-in-Chief of *IEEE Design & Test Magazine* and is/has been Associate Editor of major ACM and IEEE journals.

ALIREZA EJLALI received the Ph.D. degree in computer engineering from the Sharif University of Technology, Tehran, Iran, in 2006. He is currently an Associate Professor of computer engineering with the Sharif University of Technology. From 2005 to 2006, he was a Visiting Researcher with the Electronic Systems Design Group, University of Southampton, Southampton, U.K. In 2006, he joined the Sharif University of Technology, as a Faculty Member with the Department of Computer Engineering. From 2011 to 2015, he was the Director of the Computer Architecture Group. His research interests include low power design, real-time embedded systems, and fault-tolerant embedded systems.