Intelligent Latency-Aware Tasks Prioritization and Offloading Strategy in Distributed Fog-Cloud of Things

Chinmay Chakraborty, *Senior Member, IEEE*, Kaushik Mishra, *Member, IEEE*, Santosh Kumar Majhi, *Member, IEEE*, and Hemanta Kumar Bhuyan

***Abstract*—Offloading the dynamic tasks with fog computing is envisioned as a viable option for prolonging resource-limited constraints and improving the computational and communicational latency for delay-sensitive IoT applications. Besides, the priority of tasks and the target layers for offloading them to minimize the incurred service latency is a prime concern in layered computing architecture. To leverage the efficiency of the underlying computing nodes for the tasks' heterogeneity and computational requirements with deadline constraints, this article presents a fuzzy logic technique to prioritize the tasks based on their resource requirements and associated deadline. For efficient scheduling, an elitism-based multipopulation Jaya is proposed to map these disparate groups of tasks to a cluster amalgamation of computational-rich heterogeneous computing nodes. Moreover, a compatibility-based heuristic offloading strategy is devised to determine compatible computing nodes to offload the computations considering the availability of resources and communicational time from the respective IoT devices. Finally, extensive simulations are carried out with conflicting scheduling parameters appraising the efficacy of the proposed strategy over existing algorithms. The percentages of improvements of the proposed algorithm over the compared algorithms are 35% and 28% for average waiting. time and average service latency, respectively.**

***Index Terms*—Deadline, fog computing, fuzzy logic, IoT, latency, priority-aware, task offloading.**

Manuscript received 30 March 2022; revised 18 April 2022; accepted 29 April 2022. Date of publication 10 May 2022; date of current version 13 December 2022. This work was supported by All India Council for Technical Education, New Delhi, India, under RPS Project Grant 8-83/FDC/RPS (POLICY-1) 2019-20. Paper no. TII-22-1339. *(Corresponding author: Kaushik Mishra.)*

Chinmay Chakraborty is with the Birla Institute of Technology, Mesra 835215, India (e-mail: cchakrabarty@bitmesra.ac.in).

Kaushik Mishra is with the Sambalpur University Institute of Information Technology, Burla 768019, India (e-mail: kaushikmishra1991@gmail.com).

Santosh Kumar Majhi is with the Veer Surendra Sai University of Technology, Burla 768018, India (e-mail: smajhi_cse@vssut.ac.in).

Hemanta Kumar Bhuyan is with the Vignan's Foundation for Science, Technology and Research, Guntur 522213, India (e-mail: hmb.bhuyan@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3173899.

Digital Object Identifier 10.1109/TII.2022.3173899

I. Introduction

A TREMENDOUS latency-sensitive data are generated every day from the widespread deployment of distributed IoT devices, which requires to be processed in no time due to the associated deadline However, while processing in cloud nodes, it incurs a huge transmission delay and network congestion due to the physical gap between IoT devices and cloud servers [1]–[3]. Therefore, fog computing has evolved as a promising technology to leverage the inherent limitations of cloud computing. In traditional fog-assisted computing, the generated tasks from the IoT devices are offloaded to the computing nodes for processing via the intermediate nodes. However, the primary consensus of fog computing is to process the deadline-aware computations by the computational-rich resources to minimize the communicational latency [5]. Moreover, computing nodes take different processing times due to the disparate requirements of tasks generated by IoT devices. Besides, deadline-based tasks can be hard- and soft-deadline-based tasks. In addition, the priority constraint is associated with each task due to the sensitivity of the information for such applications. These tasks need to be offloaded onto the respective target layer for efficient processing while meeting the required QoS objectives. Therefore, to address these two critical yet sensitive issues, this research uses a fuzzy logic strategy to determine the target layers considering tasks requirements (e.g., task size, associated deadline, network bandwidth, and delay sensitivity). Moreover, the tasks with high priority and hard deadlines are not suitable for processing in the local fog nodes due to fog nodes' computational and storage-limited capabilities. Therefore, it compels to offload the tasks onto the cloud layer based on their requirements while meeting the desired deadline and QoS constraints. Thus, a layered framework, such as IoT-fog-cloud architecture is more efficient for processing the latency-sensitive tasks by the resource-intensive computing nodes while satisfying the QoS criteria.

The primary objective of this research is to design a latency-aware offloading strategy for minimizing the latency considering tasks' priorities while meeting the deadline and other conflicting QoS constraints. Besides, it aims to reduce the service rate while maximizing resource utilization. The contributions of this research are enlisted as follows.

1) Implement a strategy to minimize the offloading time for latency-sensitive applications considering deadline,

latency constraints, and resource heterogeneity for concurrent execution of dependent tasks.

2) Devise a model using fuzzy logic for classifying tasks and determining the target layers for offloading them.
3) Propose an elitism-based multipopulation Jaya (EMPJ) algorithm to map the dynamic tasks onto multicluster fog nodes for scheduling to have optimal results.

A plethora of literature [2]–[12], [21]–[23] has implemented various offloading strategies to address this issue. However, all these existing strategies have not considered delay-sensitive tasks with the associated deadline and priority. The computing nodes can process fewer tasks without any deadline or delay. Nonetheless, the latency-sensitive applications nowadays generate most of the tasks with disparate high-end requirements (such as deadline, priority, delay, etc.), which need to be processed by computationally efficient nodes to reduce the transmission latency. Most of the existing research assumed independent tasks with homogeneous resources without considering the dependent and priority-aware tasks with heterogeneous resources.

The rest of this article is organized as follows. Section II presents the system architecture followed by the computational model. The proposed latency-aware task prioritization and offloading strategy are discussed in Section III. Section IV demonstrates the performance evaluations with comparative analysis. Finally, Section V concludes this article.

II. Framework for Fog-Cloud of Things and Model Formulation

A. System Architecture

A three-layered hierarchical framework (see Fig. 1) consisting of three tiers, such as the IoT layer (Tier 1), fog layer (Tier 2), and cloud layer (Tier 3), is considered. The working nature of each layer is illustrated as follows.

First, the IoT layer consists of different smart devices that generate a bulk amount of data through terminal nodes, sensors, actuators, and embedded systems implanted in IoT smart devices. The data are associated with disparate specifications (deadline, priority, latency rate, length, etc.). Due to the limited computing and storage capacity, it gathers and offloads these tasks to either the fog layer or the cloud layer through the edge of the network (fog layer 1).

Second, the fog layer is divided into two sublayers, i.e., fog layer 1 and fog layer 2. Fog layer 1 is considered as the edge of the network consisting of intermediate nodes, such as routers, switches, called gateways for routing the packets generated from IoT devices to fog layer 2 or cloud layer. Besides, fuzzy logic is implemented in this layer to prioritize and classify the tasks by determining the target layers for offloading. Next, fog layer 2 consists of numerous computing nodes (fog nodes) distributed geographically across this layer. All the fog nodes are clustered to form a multicluster strategy to process different tasks with disparate requirements. A cluster encompasses a mix of homogeneous and heterogeneous resources to meet the QoS objectives. Within a cluster, all the fog nodes are synchronized with a cluster head (CH). All the CHs of the fog layer 2 are connected with a primary controller, called the fog controller (FC). Moreover, the

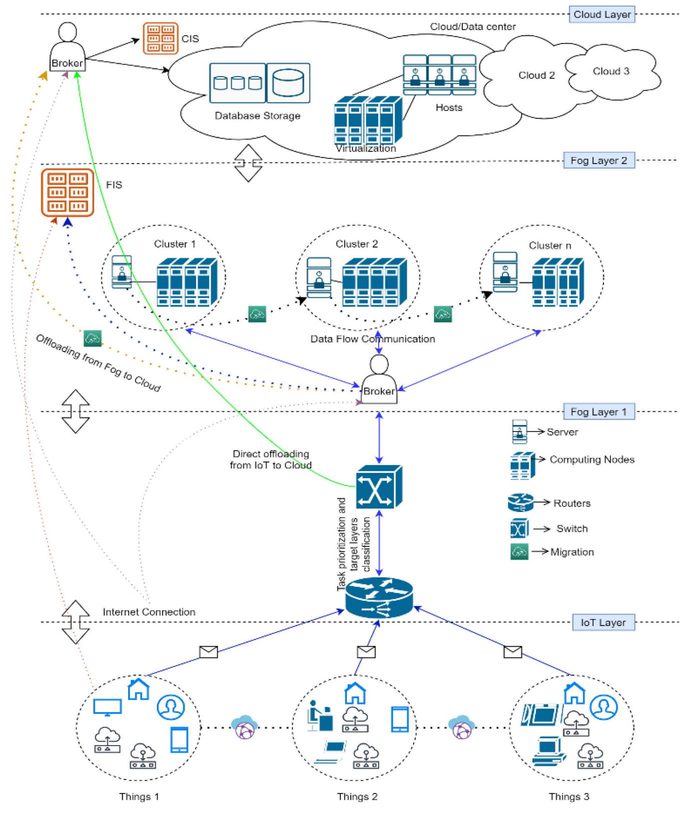

Fig. 1. Three-layered framework for IoT-fog-cloud.

FC consists of a load balancer, which distributes the loads evenly among clusters, a task buffer, which keeps all the ready tasks in a queue, and a resource monitor, which monitors the degree of consumption and availability of fog nodes collaboratively with each CH. Each cluster is having a scheduler managing the allocation and offloading of tasks from the fuzzy logic architecture (FLA).

Third, the cloud layer is primarily responsible for storing and processing the high-end tasks (high priority and hard deadline) to minimize the service time as well as latency to meet the deadline. This layer consists of centralized, computationally rich virtual machines (VMs) as computing nodes [13].

B. Computational Model

1) Task Model: A three-layered IoT-fog-cloud continuum is considered in which the number of D Internet-enabled devices (called IoT devices) denoted as $D = \{1, 2, \ldots, m\}$ and an array of M fog nodes denoted as $M = \{1, 2, \ldots, m\}$ are geographically distributed across fog layer 2. Furthermore, a set of S intermediate nodes denoted as $S = \{1, 2, \ldots, m\}$ is deployed in the fog layer 1 for task routing to suitable fog nodes or cloud VMs. Moreover, a series of H cloud VMs denoted as $H = \{1, 2, \ldots m\}$ are deployed in a centralized cloud datacenter. We assume that the memory usage and the CPU frequency of fog nodes are lower than the cloud VMs. Consider a set of dependent tasks denoted as $T = \{1, 2, \ldots n\}$ is generated by the deployment of the IoT devices and expressed in million instructions. Based on the requirements (priority, deadline, and

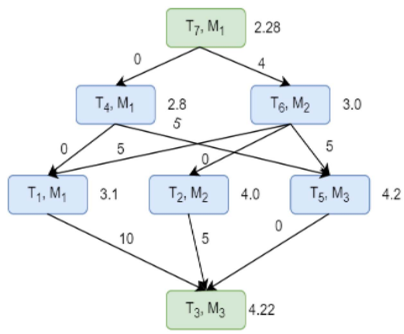

Fig. 2. Illustration of task-resource graph.

latency), tasks are processed locally in the fog layer or remotely by cloud VMs. Therefore, we denote the task mapping matrix as $X = R^{n \times |D \times M \times H|}$, where the task-resource $(i,\ j)$ entry is represented as $X_i^j = \{1,\ 0\}$, $i \in n,\ j \in m(M \cup H)$. In this mapping, 1 indicates the assignment of ith task on jth computing node and otherwise.

A directed acyclic graph (DAG) is used to represent the interdependency among tasks in a multiprocessing environment. It is denoted as $G = \{v,\ E,\ \tau, \varphi\}$, where v is the set of vertices represented as $v_{ij} = \{v_{11},\ v_{12}, \ldots,\ v_{nm}\}$, $i \in n,\ j \in m(D \cup M \cup H)$, E is the set of edges between two vertices, τ is the set of vertex weights that represents completion time, and φ is the set of edge weights that represents transmission time. The time required to transmit the interdependent data to the dependent task is referred to as the transmission time or data communication cost. The cost between two tasks is zero if both of them are processed by the same computing node. For instance, an edge $(e(v_{ij}, v_{kj}) \in E)$ between vertices v_{ij} and v_{kj} represents that the task k should get executed only after the execution of task i.

Fig. 2 illustrates a random task-resource graph with seven tasks and three computing nodes for a multiprocessing environment. In the graph, each vertex represents a pair of task-resource mapping and each edge between two consecutive vertices represents the interdependencies among them. The vertices highlighted in green color are called the entry (T_7) and the exit (T_3) vertices.

2) Latency-Aware Model: A task gets associated with the latency or delay when it is forwarded to a corresponding layer for processing and acknowledging it. Therefore, the latency rate for each type of task varies accordingly. For instance, the time taken to offload and process a task remotely would not be the same for those tasks executed and processed locally. The degree of latency for any task i offloaded by an IoT device d is characterized by three factors, such as transmission time, computation time, and acknowledgment time. Therefore, it can be estimated as the summation of the transmission latency for both fog and cloud layers.

a) Service latency in the fog layer: The service delay for a task i offloaded by an IoT device d indicates the service time to get responded by a fog node and is estimated as the aggregate of the following:

1) transmission latency for a task i to any jth fog node $(L^{\text{Fog}}_{T(i,\ j)})$;
2) computation latency for a task i to any jth fog node;
3) acknowledgment latency for a task i to the IoT device $(L^{F\_\text{Iot}}_{\text{Ack}(i)})$; and
4) the delay incurred by migrating the task i to the cloud $(L^{\text{Cloud}}_{\text{Migrate}(i,j)})$.

It is expressed as follows:

$$L^{\text{Fog}} = L^{\text{Fog}}_{T(i,\ j)} + L^{\text{Fog}}_{C(i,\ j)} + L^{F\_\text{Iot}}_{\text{Ack}(i)} + \alpha L^{\text{Cloud}}_{\text{Migrate}(i,\ j)} + L^{\text{Cloud}\_F}_{\text{Ack}(i)}. \quad (1)$$

Here, $\alpha (= \frac{T^i_{\text{sent}}}{T})$ is the ratio of total tasks sent from the fog layer to the cloud, and $L^{\text{Cloud}\_F}_{\text{Ack}(i)}$ is the acknowledgment for the migrated task i from the cloud to the jth fog node.

b) Service latency in the cloud layer: The high-end tasks are forwarded to the cloud via gateways S for processing. Thus, it also incurs some transmission delay as well as computation delay. Therefore, the total service latency experienced in the cloud layer is the sum of the following:

1) the transmission latency of transmitting the task i to the jth VM $(L^{\text{Cloud}}_{T(i,\ j)})$;
2) the computation latency of processing the task i on the jth VM $(L^{\text{Cloud}}_{C(i,\ j)})$; and
3) the transmission latency of acknowledging the IoT device d for the ith task $(L^{\text{Cld\_Iot}}_{\text{Ack}(i)})$.

Therefore, the total latency of the cloud is expressed as

$$L^{\text{Cloud}} = L^{\text{Cloud}}_{T(i,\ j)} + L^{\text{Cloud}}_{C(i,\ j)} + L^{\text{Cld\_Iot}}_{\text{Ack}(i)}. \quad (2)$$

1) *Transmission latency for offloading:* The transmission latency to offload a task i to any jth fog node or cloud VM is estimated as follows:

$$L_{T(i,\ j)} = \text{Len}_i \times L_{NW}. \quad (3)$$

Here, Len_i is the length of the ith task, and L_{NW} is the delay caused by transmitting a single byte of the task to the respective node.

2) *Computational latency:* The computation latency of a task i on the jth node is estimated as follows:

$$L_{C(i,\ j)} = \text{Len}_i \times \frac{SR(j)}{NT(j)}. \quad (4)$$

Here, $SR(j)$ and $NT(j)$ are the service rate of the jth fog node and the total number of tasks allocated to the jth fog node, respectively.

3) *Migration latency:* The delay caused by migrating the task to the cloud in an adverse situation is estimated as follows:

$$L^{\text{Cloud}}_{\text{Migrate}(i,\ j)} = \text{Len}_i \times \left(L_{NW} + \frac{SR(j)}{NT(j)} \right). \quad (5)$$

4) *Acknowledgment latency:* The transmission latency to send an acknowledgment from the fog and cloud directly to the IoT device is estimated as follows:

$$L_{Ack\ (i,\ j)} = Len_i \times L_{NW}. \quad (6)$$

III. Proposed Latency-Aware Tasks Prioritization and Offloading (LTPO)

This section elucidates the proposed latency-aware task prioritization and offloading strategy in a three-layered architecture followed by the proposed problem formulation. This proposed strategy consists of three phases as follows: (a) Task prioritization, (b) task scheduling, and (c) task offloading. The first phase deals with the assignment of the priority and determines the target offloading layer using an FLA. The second phase proposes an EMPJ algorithm for the task scheduling on the underlying computing nodes $(M \cup H)$, whereas the third phase deals with the offloading of tasks if a computing node is overloaded or fails. The problem is formulated as follows.

A. Problem Formulation

For maintaining the service latency and makespan trade-off, the task scheduling in fog–cloud architecture is modeled as a DAG $G = \{v, E, \tau, \varphi\}$. The set $M\{12,\ldots,m\} \cup H\{12,\ldots,m\} \in v$ consists of fog and cloud computing nodes, and $T\{12,\ldots,n\} \in v$ is the number of tasks. Each $i(\in T)$ in a vertex is a particle to be assigned to a computing node $(M|H)$ in the vertex. Each edge $e = \{v_i, v_j\}$ represents the interdependency among tasks in the considered multiprocessing environment. We assume that the incurred latency and completion time depend on the distance between v_i and v_j, and the current load of the node.

The objective of this research is to find the optimal mapping (X_i^j) of tasks to the underlying computing nodes to reduce the service latency and the degree of makespan. Consequently, the hard deadline (T_i^H) and the soft deadline (T_i^S) based tasks are to be allocated to the cloud VMs $j \;\; \forall j \in H$ and collaborative fog nodes j, $\exists j \in M$ to reduce the overall offloading time and service latency. However, the tasks without a deadline and low latency are offloaded to local fog nodes j, $\exists j \in M$. Therefore, the problem of task scheduling for our research can be modeled as follows: for a given set of tasks and computing nodes $(M \cup H)$, a task i is assigned to a jth computing node. When it is assigned, a great deal of service latency and completion time is experienced with low resource utilization. Therefore, the objective is to reduce the service latency $(L_{ij}^{\text{offloading}})$ (transmission, computation, and acknowledgment) and makespan $(\aleph_j)$.

Makespan ($\aleph$) can be defined as the maximum of all the completion time allocated to a jth computing node and expressed in milliseconds. If the completion time of the ith task on the jth computing node is denoted as CT_{ij}, then the makespan $(\aleph)$ for a jth computing node is expressed in (7), where L_i is the length of the task T_i and PT_j is the processing time of jth node

$$\aleph = \max \sum CT_{ij} \left(= \frac{L_i}{PT_j} \right), \text{ s. t. } \forall (M \cup H). \quad (7)$$

The mathematical formulation of the objective function is as follows:

$$\text{minimize } L_{ij}^{\text{offloading}} + \aleph_j \; \forall i \in T \; \forall j \in (D \cup M \cup H) \quad (8)$$

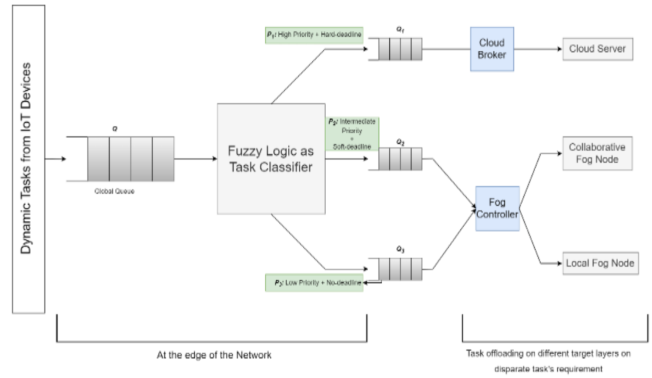

Fig. 3. Task prioritization at fog layer 1.

$subject\ to$ the following constraints:

$$\sum_{i=0}^{n} L_{ij} \times X_i^j \leq CT \;\; \forall j \in D \;\; \forall i \in \left(T_i^H \cup T_i^S\right) \quad (8a)$$

$$U_i^{CPU} \text{ and } \mathrm{U_i^{MM}} \leq \mathrm{d_j^{CPU+MM}}, \forall \mathrm{i} \in \mathrm{T}, \quad \forall j\, \{1,2,\ldots,m\} \in (M \cup H) \quad (8b)$$

$$X_i^j \in \{1,\ 0\} \quad (8c)$$

$$\sum_{i=1}^{|T|} X_i^j = 1 \;\; \forall i \in T \quad (8d)$$

$$L_{T(i,\ j)} \geq 0, \text{and } L_{\text{Ack}(i)} \geq 0. \quad (8e)$$

Constraint (8a) states that the associated deadline must be greater than the total completion time of the tasks. Constraint (8b) implies that the CPU frequency and the memory usage of a task should be less or equal to the jth computing node. Constraint (8c) represents the optimal mapping with discrete values. Constraint (8d) ensures that each task would be allocated to at most one computing node. Finally, constraint (8e) guarantees that the transmission time $(L_{T(i,\ j)})$ and the acknowledgment time $(L_{\text{Ack}(i)})$ should not be negative.

B. Tasks Prioritization

Each task is associated with some priority and deadline constraints. It is essential to classify the tasks based on their requirements and determine the target layers for offloading them to meet the QoS objectives. For task scheduling in a multiprocessing environment, reducing the incurred delay and meeting the deadline are two crucial factors. In this model, tasks are prioritized based on their requirements by the task classifier (see Fig. 3) at the fog layer 1. Besides, the fuzzy logic determines the target layers (see Fig. 4) for offloading in order to reduce the associated delay and the completion time while satisfying the deadline constraint. As shown in Fig. 3, the generated tasks from IoT devices are arrived and placed in a global queue (denoted as Q). Based on the tasks' requirements, we categorize each task into three groups, such as Group P_1, Group P_2, and Group P_3 using a FLA. This grouping of tasks facilitates the

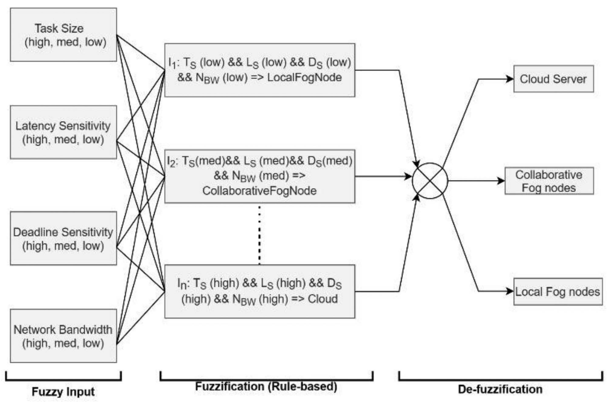

Fig. 4. Offloading decisions using FLA.

concurrent execution of each task at different layers thereby preventing aging and starvation in a multiprocessing environment. So, Group P_1 consists of all the tasks with high priority and hard deadline. These tasks require more resource-intensive and memory-intensive computing nodes for processing. Therefore, this category of tasks is processed by cloud VMs (denoted as Q_1) due to the resource-intensive nodes to minimize the average completion time thereby minimizing the service latency. Group P_2 encompasses tasks with intermediate-priority and soft deadline. This type of task does not comply with the deadline by compromising the service latency and is acceptable if a task fails to meet the deadline. Therefore, this category of tasks requires a mix of homogeneous and heterogeneous nodes to be processed. Hence, collaborative fog nodes (denoted as Q_2) are suitable for processing such tasks. Finally, Group P_3 consists of low-priority without deadlines requiring not much resource and memory capacity for getting serviced. Thus, it will be processed locally at local fog nodes (denoted as Q_3).

The proposed FLA shown in Fig. 4 consists of three primary units, such as fuzzy inputs, fuzzification, and defuzzification. Fuzzy inputs are the initial parameters of the fuzzification process. Based on the tasks' requirements, four input parameters are considered, such as task size (MI), latency sensitivity, deadline sensitivity, and network bandwidth. Each factor represents the tasks' heterogeneity for scheduling in fog-cloud of things. These input parameters are represented as lexical variables in the form of high, medium, and low. We assume that the hard and soft deadlines are represented with high and medium, and a task without a deadline is characterized as low. The fuzzifier in the fuzzification process accepts all the input parameters and computes against fuzzy membership functions defined in the fuzzy knowledge base. Membership functions are used to access the linguistic value for each fuzzy input. Four membership functions based on tasks' requirements and three specifications (high, med, and low) are defined. Furthermore, input parameters are processed by an inference engine. This engine generates fuzzy inferences or rules comprising a set of simple if-else conditions by covering all the probabilities of the system as well as application specifications. For instance, an inference $I_i,\ i \in \{1, 2, \ldots, n\}$ can be expressed as if the task size is low AND the latency sensitivity is low AND the deadline sensitivity is low AND the network bandwidth is low THEN offload the corresponding task to the local fog node, and otherwise. The resulted inferences are presented in Table I. Defuzzification is then performed to transform the fuzzy inferences into suitable values based on membership functions.

TABLE I
OBTAINED FUZZY INFERENCES

Fuzzy inputs				Output
Task Size	**Deadline sensitivity**	**Latency sensitivity**	**Network Bandwidth**	**Target Layer**
high	high	high	high	Cloud
high	high	high	med	Cloud
high	high	med	low	Cloud
high	high	med	high	Cloud
high	med	low	med	Collaborative fog node
high	med	low	low	Collaborative fog node
high	med	high	high	Cloud
high	med	high	med	Cloud
med	low	med	low	Collaborative fog node
med	low	med	high	Collaborative fog node
med	low	low	med	Local fog node
med	low	low	low	Local fog node
med	high	high	high	Cloud
med	high	high	med	Cloud
med	high	med	low	Cloud
med	high	med	high	Cloud
low	med	low	med	Collaborative fog node
low	med	low	low	Collaborative fog node
low	med	high	high	Collaborative fog node
low	med	high	med	Collaborative fog node
low	low	med	low	Local fog node
low	low	med	high	Local fog node
low	low	low	med	Local fog node
low	low	low	low	Local fog node

C. Tasks Scheduling

Since task scheduling is an NP-hard problem due to the involvement of the numerous conflicting QoS objectives, an efficient yet powerful technique is essential in the proposed three-layered architecture to minimize the waiting time and completion time. Therefore, we consider an EMPJ metaheuristic optimization algorithm for the optimal scheduling of tasks on the computing nodes. It is an extension of the standard Jaya algorithm to alleviate the performance. This algorithm is suitable for our proposed problem because it deals with multipopulation analogous to the multiple clusters in the fog layer. Moreover, it trades off between exploration and exploitation by evading the worst solutions. The new population (P_{new}) is generated through the old population (P_{old}) in the following:

$$P_{\text{new}} = \text{ round}\left(P_{\text{old}} + r \times P_{\text{old}}\right). \tag{9}$$

Each particle in the problem space undergoes a fitness function to get evaluated to find the best position among themselves. Afterward, each particle updates its position according to its fitness value using (8). If $X^k_{q,\ i}$ is the current position of the qth variable for the ith particle in the kth iteration, then the updated position ($\bar{X}^k_{q,\ i}$) is estimated using the following:

$$\begin{aligned}\bar{X}^k_{q,\ i} = X^k_{q,\ i}\ &+ n_1\left(X^k_{q,\ \text{best}^i} - \left|X^k_{q,\ i}\right|\right)\\ &- n_2\left(X^k_{q,\ \text{worst}^i} - \left|X^k_{q,\ i}\right|\right).\end{aligned} \tag{10}$$

Algorithm 1: Task Scheduling Algorithm Using Binary EMPJ.

Input Number of IoT devices D_j, number of dynamic dependent tasks T_i, number of computing nodes $N_j,\ j \in (M \cup H)$.

Output Optimal mapping X_i^j of task T_i to a computationally efficient computing node N_j.

1. ***for*** i = 1: n, evaluate particles' fitness through (8);
   if $(F < BEST_i)$
   Set the new fitness value as the $BEST_i$;
2. ***for*** i = 1: n, find an updated class of positioning by estimating particles' position using (10);
3. Modify the solution by transforming current solutions into discrete solutions through (11) and (12);
4. continue till the maximum iteration is reached;
   while (maximum iteration ≠ met)
   Produce a new series of the population through (9);
   if $(P_{new} > P_{old})$
   augment the optimal solutions in the new population to the $(P_{new} - P_{old})$ solutions;
   else if $(P_{new} < P_{old})$
   augment the optimal solutions in the current population to the P_{new} solutions;
   else if $(P_{new} < m)$
   assign $P_{new} = m$;
   else
   Output the optimal mapping;
5. Return the optimal solution as the efficient mapping of tasks to computing nodes;

Here, n_1 and n_2 are the arbitrary numbers (0.5), $X^k_{q,\ \text{best}^i}$ and $X^k_{q,\ \text{worst}^i}$ are the best and the worst solutions for the ith particle in the kth iteration.

All these generated continuous solutions are not suitable to represent the assignment matrix (X_i^j). Hence, these are transformed into binary solutions for mapping tasks onto resources with discrete values. It is computed using the following:

$$\tan h\left(\left|\ \bar{X}^k_{q,\ i}\right|\right) = \frac{e^{\left(\left|2\ \bar{X}^k_{q,\ i}\right|\right)} - 1}{e^{\left(\left|2\ \bar{X}^k_{q,\ i}\right|\right)} + 1}. \tag{11}$$

The updated value of X_i^{k+1} is then represented in binary form using the following:

$$\bar{X}^k_{q,\ i} = \begin{cases} 1, \text{if rand}() < \tanh\left(\left|\ \bar{\mathrm{X}}^{\mathrm{k}}_{\mathrm{q,\ i}}\right|\right) \\ 0,\ \text{otherwise} \end{cases}. \tag{12}$$

Algorithm 1 illustrates the proposed EMPJ algorithm.

D. Tasks Offloading

For the task offloading, it has been assumed that the fog nodes, as well as the Cloud VMs, are connected with all the intermediate nodes at the edge of the network. We assume that the CH of each cluster synchronizes with either other CHs or cloud information service for the offloading of heavy computation-intensive tasks. Here, a heuristic approach is used for identifying a compatible computing node for each overloaded task to maintain the tradeoff among loads. So, the proposed heuristic identifies a compatible node j for the overloaded task T_k, which meets the deadline with minimum transmission and computation time.

For the migration, several processing cores (CPU), storage, and bandwidth are the resource usage constraints for each task (denoted as $\overrightarrow{T_i}$) to be offloaded on a jth computing node. Therefore, the total resources used $(\overrightarrow{R_{\text{used},j}})$ and the total resources available $(\overrightarrow{R_{\text{avail},j}})$ for each underloaded computing node $A_j^u\{\in (M \cup H)\}$ is estimated as follows:

$$\overrightarrow{R_{\text{used},j}} = \sum_{i=1}^{n} \overrightarrow{T_i} \tag{13}$$

$$\overrightarrow{R_{\text{avail},j}} = \overrightarrow{R_{\text{total}}} - \overrightarrow{R_{\text{used},j}}\,. \tag{14}$$

Here, $\overrightarrow{R_{\text{total}}}$ is the total resources of the jth underloaded node A_j^u.

Finding the similarity between the tasks of overloaded nodes (T_k^o) and the underloaded nodes (A^u), cosine similarity (σ) is evaluated using (15). The smaller value of σ means the greater similarity between tasks (T_k^o) with total resources. Now, the compatibility (θ) between each task with the underloaded nodes (A^u) is estimated to identify a suitable computing node based on the cosine similarity value and the degree of utilization. Hence, with the best compatibility, a suitable computing node for a task T_k is selected using (16), where β is set to 0.5

$$\sigma\ = \cos^{-1}\left(\frac{\overrightarrow{T_k} \times \overrightarrow{R_{\text{avail},\ j}}}{\left|\overrightarrow{T_k}\right|\left|\overrightarrow{R_{\text{avail},\ j}}\right|}\right) \tag{15}$$

$$\theta\ =\ \beta \times \sigma + (1-\beta) \times U_j. \tag{16}$$

The proposed heuristic-based tasks offloading policy maintains the tradeoff by preventing the computing nodes from getting overloaded or underloaded and selecting the most compatible node for the tasks offloading based on resource usage factors and the degree of utilization.

IV. Performance Evaluations

This section illustrates the performance evaluation of the proposed strategy and the comparative assessment of the existing literature [14]–[19] in terms of average waiting time, average latency rate, and the number of tasks meeting the deadline constraint.

A. Simulation Setup

We consider 60 IoT devices deployed widely that produce tremendous tasks in a span of Δt time with disparate lengths ranging from 0 to 15 000 (MI). To process these enormous, computational-intensive tasks, various fog nodes and VMs with disparate specifications are deployed in fog and cloud layers, respectively. The transmission bandwidth from IoT devices to the fog nodes and VMs is 102 400 Hz. Empirical simulations for each considered objective are carried out with 30 independent

TABLE II
SIMULATION PARAMETERS

Parameters	Values
Number of computing nodes ($M \cup H$)	294
Number of IoT devices (D)	60
Number of intermediate nodes (S)	10
Number of tasks (T)	3000
Bandwidth (MIPS)	102400 Hz
Storage capacity of the fog nodes	1024 MB
Storage capacity of the Cloud	1024 TB
Nodes' heterogeneity (Cores, Speed)	Large (4vCPUs, 10500) Medium (2vCPUs, 5500) Low (1vCPUs, 3500)
Tasks' heterogeneity (length, latency rate)	High (10000-15000, 0.1) Med (5000-10000, 0.3) Low (0-5000, 0.7)

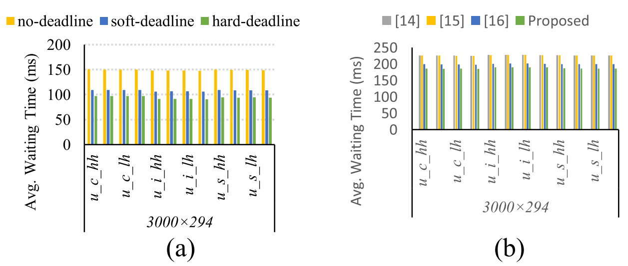

Fig. 5. Performance analysis of average waiting time (a) of different deadlines and (b) (tasks *x* nodes) heterogeneity.

runs and the mean values are considered for finding the optimal results. For the simulations, the iFogSim is used as a simulator over the CloudSim for enabling a virtualized environment. We assume 50 particles as the population size and 100 as the maximum termination point for the binary EMPJ algorithm. A real-world benchmark dataset [20] is considered to validate the effectiveness of the proposed algorithm. This dataset consists of the diverse tests set, such as uniformly generated data (u), nature of consistency (x) [consistent (i), semiconsistent (s), and inconsistent (i) tasks], tasks heterogeneity (t), and machine heterogeneity (m). These diversified tasks are represented in a matrix called the estimated time to compute matrix. This matrix consists of 12 instances of tasks set based on the tasks' requirements in the form of u_x_tm subject to high (h) and low (l). Table II summarizes the considered simulation parameters.

B. Performance Analysis on Average Waiting Time

This time refers to the time taken by each task while waiting in the ready queue till getting allocated to the compatible computing node for computation. This time impacts the performance, especially when the tasks are of different types with disparate requirements. Therefore, this time should be reduced to improve the overall performance. Problems like starvation and aging could be possible for both high and low priority-based tasks. Due to the concurrent execution of these tasks by computationally intensive computing nodes on different target layers, these two problems have been avoided to maintain a minimized waiting time. Fig. 5 depicts the performance of the average waiting time for different existing algorithms.

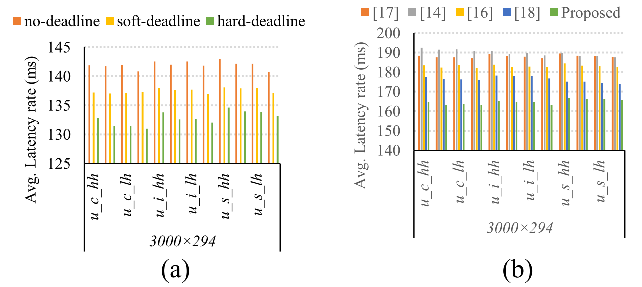

Fig. 6. Performance analysis of average latency rate (a) for different deadlines and (b) (tasks *x* nodes) heterogeneity.

Fig. 5(a) illustrates the waiting time for the tasks with different deadlines. As depicted in Fig. 5(a), the time taken by hard-deadline-based tasks is less in comparison to the soft-and-no-deadline-based tasks due to the associated priorities, different requirements, and computation by disparate computationally-intensive nodes. Fig. 5(b) presents the obtained simulation results for the proposed algorithm with the existing methods [14]–[16]. It is evident that the proposed method tackles the tasks with different latencies and deadlines effectively. Consequently, the waiting time for each task is considerably reduced, which in turn impacts the average waiting time. Moreover, the proposed method outperforms other compared methods.

C. Performance Analysis on Average Latency Rate

This factor minimizes the latency time for each task on completion during offloading, computation, and acknowledgment. So, this rate influences the performance of the system. Due to the disparate requirements of tasks, this parameter varies for different groups of tasks. When the size of the tasks increases along with their communicational and computational requirements, it becomes difficult to reduce the associated latency. Therefore, the proposed approach uses a FLA to determine the offloading layer for processing to minimize the average latency. Moreover, Fig. 6(a) shows the performance analysis of tasks with different deadlines for increasing tasks. It depicts that the latency time of higher priority (hard deadline) based tasks is lower than the lower priority or without priority. Fig. 6(b) presents the performance comparison for the average latency rate of the proposed method versus the existing methods [14], [16]–[18]. The proposed method reduces the latency time considerably for the increasing tasks with disparate deadlines, delays, and priorities.

D. Performance Analysis on Several Tasks Meeting the Deadline

This constraint implies that the tasks satisfy their deadline with their requirements. This factor primarily depends on the average waiting time in the ready queue and the transmission time of each task on different target layers for processing. Fig. 7(a) depicts the satisfying rate for hard-deadline-based tasks. It is clear from the results that the time taken for meeting the deadline for hard-deadline-based tasks is lower than the

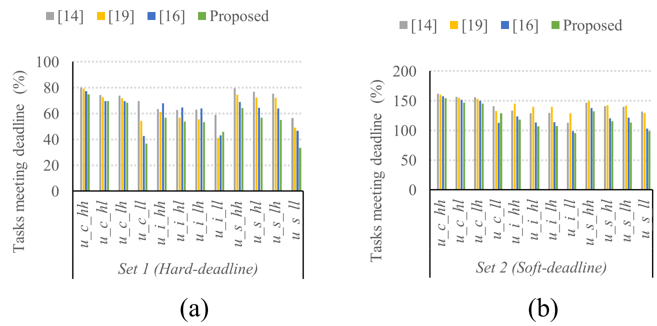

Fig. 7. Average number of tasks satisfying their deadlines. (a) Hard deadline. (b) Soft deadline.

soft-deadline-based tasks [see Fig. 7(b)]. In a hard-deadline-based approach, the tasks meet their required deadline while processing on a compatible computing node. In some cases, the soft-deadline-based tasks fail to meet the deadline due to the resource-limited, and computationally ill nodes. Therefore, the failed tasks are then offloaded to compatible target layers. Fig. 7(b) shows the performance analysis of the proposed method for soft-deadline-based tasks. As a result, the proposed method performs better than the compared methods [14], [16], [19] for a varying number of tasks.

The percentages of improvements of the proposed algorithm over the compared algorithms are 35% and 28% for average waiting time and average service latency, respectively.

V. Conclusion

This article addressed the task prioritization and offloading policy in a three-layered architecture with a fuzzy logic. The primary objective of this research was to reduce the average waiting time and incurred latency while satisfying the deadline constraints. Moreover, the proposed strategy considered the priority of each task and placed them in the corresponding queues to be scheduled by one of the compatible nodes. The concurrent execution of each type of task in different target layers prevented starvation and aging. Besides, it used a binary EMPJ algorithm to schedule the varying tasks to find the optimal mapping. Both the tasks' and machines' heterogeneity were considered to appraise the efficacy of the proposed algorithm. The obtained experimental results showed the effectiveness of the proposed algorithm over other compared algorithms for disparate QoS conflicting objectives.

As a part of future work, we plan to expand it by deploying containers in place of VMs and shifting toward serverless computing at the fog layer for more diversified and optimal results while satisfying the intensive-rich data.

References

[1] C. Chang, S. N. Srirama, and R. Buyya, "Internet of things (IoT) and new computing paradigms," in *Fog and Edge Computing: Principles and Paradigms*. Hoboken, NJ, USA: Wiley, Feb. 2019, pp. 1–23.

[2] L. Yang, K. Yu, S. X. Yang, C. Chakraborty, Y. Liu, and T. Guo, "An intelligent trust cloud management method for secure clustering in 5G enabled internet of medical things," *IEEE Trans. Ind. Informat.*, to be published, doi: 10.1109/TII.2021.3128954.

[3] T. G. Rodrigues, K. Suto, H. Nishiyama, and N. Kato, "Hybrid method for minimizing service delay in edge cloud computing through VM migration and transmission power control," *IEEE Trans. Comput.*, vol. 66, no. 5, pp. 810–819, May 2017.

[4] A. Yousefpour, G. Ishigaki, and J. P. Jue, "Fog computing: Towards minimizing delay in the internet of things," in *Proc. Int. Conf. Edge Comput.*, 2017, pp. 17–24.

[5] C. Fricker, F. Guillemin, P. Robert, and G. Thompson, "Analysis of an offloading scheme for data centers in the framework of fog computing," *ACM Trans. Model. Perform. Eval. Comput. Syst.*, vol. 1, no. 4, Sep. 2016, Art. no. 16.

[6] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in fog-cloud computing paradigm," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manage.*, 2017, pp. 1222–1122.

[7] D. G. Roy, D. De, A. Mukherjee, and R. Buyya, "Application-aware cloudlet selection for computation offloading in multi-cloudlet environment," *J. Supercomput.*, vol. 73, no. 4, pp. 1672–1690, 2017.

[8] J. Xu, B. Palanisamy, H. Ludwig, and Q. Wang, "Zenith: Utility-aware resource allocation for edge computing," in *Proc. Int. Conf. Edge Comput.*, 2017, pp. 47–54.

[9] J. Y. Zhang *et al.*, "Optimizing power consumption of mobile devices for video streaming over 4G LTE networks," *Peer-Peer Netw. Appl.*, vol. 11, no. 5, pp. 1101–1114, 2018.

[10] H. O. Hassan, S. Azizi, and M. Shojafar, "Priority, network and energy-aware placement of IoT-based application services in fog-cloud environments," *IET Commun.*, vol. 14, no. 13, pp. 2117–2129, 2020.

[11] Y. Yang, K. Wang, G. Zhang, X. Chen, X. Luo, and M.-T. Zhou, "MEETS: Maximal energy efficient task scheduling in homogeneous fog networks," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 4076–4087, Oct. 2018.

[12] J. He, J. Wei, K. Chen, Z. Tang, Y. Zhou, and Y. Zhang, "Multitier fog computing with large-scale IoT data analytics for smart cities," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 677–686, Apr. 2018.

[13] K. Mishra, R. Pradhan, and S. K. Majhi, "Quantum-inspired binary chaotic salp swarm algorithm (QBCSSA)-based dynamic task scheduling for multiprocessor cloud computing systems," *J. Supercomput.*, vol. 77, pp. 10377–10423, 2021.

[14] S. S. Tripathy, D. S. Roy, and R. K. Barik, "M2FBalancer: A mist-assisted fog computing-based load balancing strategy for smart cities," *J. Ambient Intell. Smart Environ.*, vol. 13, no. 3, pp. 219–233, 2021.

[15] Y. Yang, S. Zhao, W. Zhang, Y. Chen, X. Luo, and J. Wang, "DEBTS: Delay energy balanced task scheduling in homogeneous fog networks," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 2094–2106, Jun. 2018.

[16] S. Sharma and H. Saini, "A novel four-tier architecture for delay aware scheduling and load balancing in fog environment," *Sustain. Comput.: Inform. Syst.*, vol. 24, 2019, Art. no. 100355.

[17] C. Sonmez, A. Ozgovde, and C. Ersoy, "Fuzzy workload orchestration for edge computing," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 2, pp. 769–782, Jun. 2019.

[18] J. Almutairi and M. Aldossary, "A novel approach for IoT tasks offloading in edge-cloud environments," *J. Cloud Comput.*, vol. 10, no. 1, pp. 1–19, 2021.

[19] X. Xu *et al.*, "Dynamic resource allocation for load balancing in fog environment," *Wireless Commun. Mobile Comput.*, vol. 2018, 2018, Art. no. 6421607.

[20] T. D. Braun *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[21] Y. Wu, H. Guo, C. Chakraborty, M. Khosravi, S. Berretti, and S. Wan, "Edge computing driven low-light image dynamic enhancement for object detection," *IEEE Trans. Netw. Sci. Eng.*, to be published, doi: 10.1109/TNSE.2022.3151502.

[22] C. Chinmay, K. Amit, and J. P. C. R. Joel, "Novel enhanced-grey wolf optimization hybrid machine learning technique for biomedical data computation," *Comput. Elect. Eng.*, vol. 99, pp. 1–15, 2022, Art. no. 107778. [Online]. Available: https://doi.org/10.1016/j.compeleceng.2022.107778

[23] H. K. Bhuyan, C. Chakraborty, S. K. Pani, and V. Ravi, "Feature and sub-feature selection for classification using correlation coefficient and fuzzy model," *IEEE Trans. Eng. Manage.*, to be published, doi: 10.1109/TEM.2021.3065699.