Multimedia Tools and Applications (2023) 82:32305–32328
https://doi.org/10.1007/s11042-023-14565-0

Task scheduling for improved response time of latency sensitive applications in fog integrated cloud environment

Rishika Mehta[1] · Jyoti Sahni[2] · Kavita Khanna[3]

Received: 9 September 2021 / Revised: 20 May 2022 / Accepted: 31 January 2023 /
Published online: 3 March 2023

Abstract
Fog integrated Cloud Computing is a distributed computing paradigm where near-user end devices known as fog nodes cooperate with cloud resources hosted at distant datacentres for providing computational and storage services to end user applications. One of the most challenging issues in fog integrated cloud based system is task scheduling. Most of the existing scheduling approaches involve centralized decision making which fail to exploit the advantages that may be achieved by a decentralized approach, that directly maps with the distributed architecture of fog based systems. This work proposes a decentralized heuristic algorithm for scheduling real-time IoT applications bounded by tolerable latency as the Quality of Service (QoS) constraint. The proposed technique aims to take into consideration the resource constraints of the fog resources to yield a schedule that not only meets the QoS requirements defined in terms of tolerable latency but also improves the response time of applications hosted on a fog-cloud infrastructure. Performance evaluation on different IoT applications indicate that the presented algorithm delivers better performance by reducing response time by 11% on an average in comparison to the other state-of-the-art policies.

Keywords Internet of things · Fog computing · Task scheduling · Response time · CPU utilization

✉ Rishika Mehta
rishikamehta10@gmail.com

Jyoti Sahni
jyoti.sahni@ecs.vuw.ac.nz

Kavita Khanna
kavita.khanna@dseu.ac.in

[1] The NorthCap University, Gurugram, India

[2] Victoria University of Wellington, Wellington, New Zealand

[3] Delhi Skill and Entrepreneurship University, New Delhi, India

1 Introduction

In recent years, Internet of Things (IoT) has seen an exponential growth. It is being seen to have huge potential to provide solutions to various real world problems in nearly all aspects related to individuals, corporations, and society as a whole. Typical application of IoT involves developing smart environments in domains such as buildings, cities, healthcare, emergency, transport, agriculture, supply chain, retail, lifestyle, energy, environment, tourism and culture [36].

One important aspect of the majority of IoT applications is that they generate vast volume of data which is required to be analysed for subsequent decision making. Since cloud computing has been used effectively to process voluminous data owing to its high processing and storage capabilities [8, 37], the collaboration of IoT with cloud computing has been explored in many research efforts [4, 13]. However, a major issue observed with cloud computing is that cloud data centers are situated at a multi-hop radius from the IoT data sources. This means that all the requests and data must be transferred to remote cloud which significantly increases the response time of the applications. This may be suitable for some applications but is not desirable in the case of time critical IoT applications such as early warning systems in healthcare, manufacturing and meteorological services. Besides, IoT devices are geo-distributed, generating a massive amount of data which if forwarded to the cloud for analysis would overload the global Internet. If each tuple of IoT data is transmitted to the cloud, increased application response time, bandwidth saturation of network channels and degraded Quality of Service (QoS) might be experienced [23].

To overcome the aforesaid shortcomings of IoT and cloud integration, fog computing [9] has been devised that makes the use of elastic compute and storage resources at the edge of the network. The purpose is to bring compute, storage, and networking facilities near to the end-users or the source of generation of data rather than dispatching it to the cloud [1, 6, 40]. The aim is not to replace cloud with fog but to serve the requests that cannot be satisfied timely by cloud alone. Fog computing produces faster response to events by dismissing a round trip to the cloud. It processes extremely time-sensitive data near to the *things* [11]. Most often, a collaboration of fog and cloud computing is required to cater for the QoS and resource requirements of large-scale IoT enabled systems. While fog nodes provide local services enabling low response-time and context awareness, cloud provides centralized services globally supporting long-term storage and complex analysis of data [12]. For requests which require long-term storage or demand comprehensive evaluation entailing archival data-sets (e.g. data backups and analysis), fog nodes only serve as gateways or routers to direct the requests to the core cloud computing framework.

Any device with computing, storage, and networking capabilities lying in the path from IoT device to cloud can act as a fog node. Fog nodes are generally resource-constrained (such as networking devices like proxy servers, set-top boxes, switches, base stations, gateway routers etc. which can be installed near IoT devices/sensors), heterogeneous and hierarchically arranged in multiple levels forming a fog device layer between IoT device layer lying at the bottom and cloud data centre layer at the top of the distributed computation model [6, 13]. IoT applications hosted on fog-cloud environment are generally considered as a set of lightweight and interdependent tasks or modules which can be deployed on to distributed fog and cloud resources [35]. Scheduling these application tasks onto distributed fog and cloud resources is therefore an optimal assignment problem. It involves assigning application tasks to the pertinent nodes such that the desired QoS requirements are met under the restrictions imposed by the limited capacity and heterogeneity of the underlying fog resources [3].

Below, two example scenarios are discussed where Fog Computing can be utilized to reduce the application response-time in order to enhance real-time experience of the end-users.

Traffic management system Intelligent Transport System (ITS) aims to improve traffic, timely identify possibilities of collision, reduce stop-time of vehicles and provide other value added services. ITS involves building a vehicular network where each vehicle is equipped with sensors, processors and wireless communication modules. While the processing capabilities of the processors embedded in vehicles are limited, ITS applications require frequent data processing and demand faster response. For this reason, many ITS applications leverage cloud infrastructure to execute and deliver services, which however, is not appropriate for many latency sensitive ITS applications. Furthermore, depending exclusively on the centralized cloud for these high priority services results in communication overhead, which exhausts the network bandwidth and increases the response time, consequently reducing the overall system performance. To provide response to the moving vehicles in real-time, road side units generally spread out in various parts of the city can be effortlessly promoted to function as fog nodes. These road-side units take region-level decisions and provide diverse services to intelligent vehicles, such as video-surveillance, navigation and regulation of traffic lights for reducing congestion. In contrast, cloud servers can be used for processing data for long term purposes such as study of usage models and predictive analytics.

Fall detection system Falls are considered risky for the elderly people as it can severely impact their health. For older people living alone, a stumble at home can be very risky as they might not be able to get assistance. Studies indicate that more than 20% of individuals who got admitted to the hospital following a fall have been on the ground for longer than an hour [22]. Specifically, half of the people who remain on the ground for more than an hour following a fall, die in a span of six months even without experiencing direct injury, reflecting a decline in general health [38]. A fall, if not assisted in time, can cause functional impairment in elderly people. In many cases, to detect the fall, accelerometer in smartphone is employed to continuously collect the movements of aged persons [42]. The sensed data is dispatched to the cloud and then, notifications from the cloud are sent to the patient's care-providers in the event of an emergency. Since accelerometer generates voluminous data, it requires high bandwidth as well as fast internet speed to send the data in its entirety from smartphones to the cloud. Continuously sending a large volume of data (i.e. hundreds of GBs) from the smartphone to the cloud increases the response time. For example, a late notification from the cloud to a doctor can lead to a delayed response in case of critical head-injury, which might result in death. In this case, the doctor could provide emergency assistance (e.g., preventing bleeding) to save the patient's life if the doctor was aware of the issue in real time. Therefore, the identification of a fall must be carried out in real-time which cannot be achieved if the application tasks are scheduled solely in the remote cloud. One of the possible ways to address this issue is to send the data from various patients to the proximal fog nodes where data can be processed in parallel enabling detection of the fall in a reasonable time-span. Besides reducing the response time, it also reduces the network congestion and overall cost of using cloud resources.

Both the scenarios discussed above depict potential response-time sensitive applications that may benefit by leveraging fog-cloud infrastructure and are suitable use-cases for our proposed work.

In this paper, we study the problem of task scheduling in a fog-cloud integrated environment with the objective of meeting the latency constraints and improving the overall response

time under resource limitations experienced at fog nodes. Specifically, the problem involves multiple users where each user submits a time-sensitive application as an interdependent set of tasks that is to be executed within its pre-defined user's tolerable latency; the objective is to not only meet the user defined tolerable latency but also to reduce the overall response time of the submitted applications. This not only helps meet the QoS requirements of latency sensitive applications but also improves performance as observed by the user by making the best use of available resources. We present a solution to this problem by proposing a distributed task scheduling strategy that takes advantage of fog computing environment while taking into consideration the resource constraints in order to generate a schedule that executes the tasks of multiple IoT applications within user-specified tolerable latency while improving the overall response time of the applications.

The organization of the paper is as follows. Section 2 reviews the literature on task scheduling in fog computing. Section 3 presents the DAG-based application structure and the fog-cloud architecture considered in this work. Section 4 describes the problem definition, followed by Section 5 where the proposed task scheduling algorithm is presented. Performance evaluation based on the experiments is detailed in Section 6 and finally, Section 7 concludes the work.

2 Related work

Task scheduling on distributed resource(s) is an NP-hard problem as elucidated in [20]. To deal with this intractable problem, it is assertive to apply heuristic or metaheuristic optimization techniques for yielding comparative or near-optimal solutions. To meet the demands of delay-sensitive applications, most of the existing works propose utilizing fog computing for task scheduling. In this section, some current contributions of the field are surveyed.

Guevara and Fonseca developed a task scheduling technique based on Integer Linear Programming for reducing the makespan of submitted IoT based applications [18]. The authors employed a classifier at the edge of the network to label the submitted applications according to their QoS requirements. The labels can be Best-effort, CPU-bound, Streaming, Conversational, Interactive, Real-Time or Mission-critical; out of which the authors considered real-time as well as delay-tolerant applications. Based on the assigned label and resource availability, the scheduler determines fog or cloud resource where the job should be scheduled. This approach is evaluated on EEG tractor beam game and Video Surveillance applications. Results depict that the proposed technique reduces makespan compared to the traditional task scheduling techniques.

Mahmud et al. proposed a heuristic-based technique to cater to latency-aware module placement over the fog environment [24]. In this module placement policy, two algorithms are proposed. The first algorithm focusses on meeting the user-defined tolerable latency by treating latency-sensitive and latency-tolerant applications differently. The second algorithm optimizes the resource usage by relocating an application module from an under-occupied node to an over-occupied node in the same cluster based on the first-fit policy. However, the policy does not serve user requests faster by reducing the response time as it does not favour the placement of tasks in the lower level fog nodes even if there is enough resource availability.

Skarlat proposed a conceptual architecture for provisioning of the resources and placement of the services in the fog environment [31]. Authors proposed the idea of fog colonies which consist of arbitrary number of fog cells and employed genetic algorithm to place the services in fog cells. This algorithm is evaluated in terms of resource usage, response time, processing cost and deadline adherence.

Liu et al. introduced a multi-objective mathematical model by performing computation offloading in the fog system to create a trade-off between delay, energy, and cost parameters [21]. By utilizing queuing theory, different queue models are created for fog resources, mobile nodes, cloud data centers, and wireless channels. This optimization strategy with multiple objectives is converted into a single objective optimization approach which is solved using Integer Point Method (IPM). Simulation results depict the supremacy of the approach over state-of-the-art approaches. However, the employed model is centralized due to which it suffers from low availability and scalability as the central node is susceptible to single point of failure.

Yang et al. developed a task scheduling approach utilizing homogeneous fog infrastructure [39]. The policy employs Lyapunov optimization strategy for reducing energy consumption along with the service delay. Various mathematical models for offloading the tasks, data transmission, handling of the incoming traffic, and queuing are built to develop a comprehensive analytical framework. Simulation results as well as theoretical study showcase the superiority of this presented approach when compared with Least Busy Scheduling and Traditional Random Scheduling policies; however, low scalability is an issue with this strategy.

Ni et al. proposed a dynamic strategy for allocation of resources on the basis of Priced Timed Petri Nets (PTPN) in fog environment [27]. The central idea of this strategy is to improve cost, makespan as well as the efficiency of the fog resources. The authors created PTPN models which dynamically allocate the tasks to the fog devices. Empirical results of this approach depict efficient resource selection and better utilization of chosen resources when compared with existing approaches. However, this strategy can be further improved by incorporating learning function in resource provisioning strategy to offer better services to the clients.

Bitam et al. introduced bee life optimization strategy to deal with the job scheduling problem [5]. This technique works in multiple phases. In the first phase, an end-user forwards a service request to the nearby fog resource. In the next phase, the fog resource forwards the information relevant to the request to an administrator node. In the third phase, the centralized administrator breaks down the job into a number of tasks and executes bee life algorithm to determine an optimal schedule. In the fourth phase, tasks are processed on the fog nodes as per the schedule. In the next phase, all the fog nodes forward the results to the administrator node. The administrator then combines the results and forwards the response to the end user via a primary fog node. The effectiveness of this approach is determined based on the makespan and total memory consumption in executing all the tasks. However, this algorithm functions as a centralized algorithm and follows static scheduling which is not effective for the real-time environments in which new traffic keeps on entering the system when existing requests are already being served.

Xuan-Qui et al. introduced a heuristic-based approach for task placement problem to find an optimal balance between the monetary cost and task execution time in fog-cloud infrastructure [28]. There are three stages to the task placement approach. In the first stage, the priority of the task is determined. In the next stage, a suitable fog or cloud node is allocated to the task in order to attain an optimum balance between the cloud cost and the makespan. In the last stage, the resulting schedule from the previous stage is fine-tuned to satisfy the predefined tolerable latency. To reduce the total makespan, the best node for each task on the critical path is selected.

Brogi and Forti proposed a generic model for IoT application deployment over the fog environment [7]. The authors accessed the quality of considered framework based on bandwidth utilization and latency metrics. Whilst creating pertinent applications, model based tools can be utilized during design, deployment and run time phases. Authors implemented one such tool, FogTorch in Java. By applying various aspects relevant to fog, certain concerns have

been carefully considered including (i) the proposed solution disregards the communication technologies at each tier, focusing solely on their quality of service (ii) the fog nodes can have different computational capabilities with no dependence on particular hardware or software (iii) all kinds of components of the application and IoT devices are suited.

Kafhali and Salah proposed a queuing based analytical model to examine the effectiveness of fog-cloud integration in an IoT environment [14]. The presented model depends on following metrics: QoS, amount of fog nodes required, and maximum supported load. Given the two metrics, the third one can be determined. The analytical framework consists of edge devices, cloud data center and cloud gateway. From the analytical framework, numerical formulas are formulated for the key parameters such as throughput, CPU utilization, average message requests, loss rate and response time. Extensive simulations were performed using discrete-event simulator to show the effectiveness of the presented approach.

Guerrero et al. introduced a decentralized algorithm for the placement of services in a fog system [17]. The central idea of this policy is that it places the most requested services near to the end-users. The approach used depicts its effectiveness through hop count metric as the user-nearness indicator. The approach determines the task priority according to its request rate. The most requested services are the ones with the highest priority. The algorithm places the most popular services near the clients and migrates lesser demanded services to the fog devices lying up in the hierarchy. The policy reduces usage of the network and improves service latency for the most requested services. However, it increases the service latency and count of service migrations for the less requested services.

Ramasubbareddy and Sasikala proposed a response time based approach for task offloading in a cloudlet environment [29]. A mobile user submits a task request to the centralized cloudlet controller. The centralized controller creates and keeps a record of the response times of the cloudlet servers and selects an appropriate cloudlet server on the basis of minimum response time. The proposed strategy tries to balance the load on the computational servers while choosing an apt computational server each time. Experiment based evaluation of this technique with existing approaches shows outcomes in favour of this policy.

Gupta et al. introduced a distributed task scheduling strategy, Edgeward for scheduling tasks based on the resource requirements [19]. This policy schedules the tasks in each leaf-to-root path between end-devices and cloud. Tasks from the different paths are combined if they are allocated the same device and shifted to the parent device to combine both the instances even if the nearby devices have adequate resource availability. The results of this policy are compared with cloud-based placement strategy to showcase the benefits of task placement in the fog environment.

Deng et al. presented a mathematical framework for the problem of allocation of workload examining a balance between power consumption and transmission delay in fog-cloud environment [12]. The primal problem(PP) is decomposed into sub-problems, which are solved using existing optimization techniques. It aims to find an equilibrium between power consumption and delay in the fog subsystem using convex optimization techniques. The second sub-problem is a mixed integer nonlinear programming (MINLP) problem. Since MINLP is computationally complex to solve, generalized blenders decomposition (GBD) is employed to find an equanimity between computation delay and power consumption in the cloud computing environment. Considered as an assignment problem, the last subproblem tries to reduce the transmission delay from fog device to cloud resources for a defined rate of traffic by using Hungarian method. It is evident from this policy that by exploiting fog resources, network bandwidth is saved and communication latency is reduced which considerably

enhances the overall performance. However, this workload allocation policy follows a centralized strategy that is less favoured for fog computing system.

Zeng et al. proposed a heuristic algorithm by exploiting the energy harvesting potential of the fog nodes by utilizing service composition in the fog-cloud environment [41]. This policy is developed for Cyber Physical Systems by making the use of green energy in the fog resources. By considering the distribution of fog nodes in a large geographic area, partial requirement of the energy as green energy can be fulfilled by making use of natural resources like wind and sun. In order to enhance the throughput of the system, proposed strategy considers service replica deployment, load balancing and source rate control. CPS energy-efficient application mapping approach is formulated as a mixed integer linear programming problem. Since this problem is NP-Hard, it is addressed by developing heuristic approach. Experimental evaluation of the presented approach yields near-optimal results. However, the proposed strategy is centralized due to which it has low scalability.

Craciunescu et al. designed a latency aware task mapping technique in healthcare domain [10]. The proposed algorithm tries to improve the response time by mapping majority of the tasks locally or on the fog devices rather than forwarding to a cloud data center. This approach recognizes the undesirable events such as leaking of gas in a home-setup and reports the occurrence by issuing a notification to the user in a shorter period of time. The algorithm is empirically tested in a real-time home environment and is able to identify the inappropriate events with more than 90% accuracy.

Gu et al. developed a resource management algorithm for cost minimization in a fog integrated medical cyber physical system [16]. An MINLP problem is formulated while also considering task distribution, virtual machine placement and base station association in order to minimize the cost. Further mixed integer linear programming problem is introduced to reduce the computational complexity of MINLP and finally, a heuristic algorithm is designed which has low complexity. The evaluation results show a noticeable reduction in delay as well as cost.

Stavrinides and Kratza proposed a heuristic based dynamic scheduling strategy for IoT workflows [33]. The approach tries to place computation intensive tasks on cloud while communication intensive tasks are placed on fog resources. The authors claim that the proposed policy makes effective use of slack time by scheduling ready tasks on the fog or cloud resources. The algorithm is tested on synthetic workload and is able to attain lesser deadline miss ratio but at significant monetary cost.

Table 1 offers a brief overview of the related works on task scheduling in a distributed fog-cloud environment. Most of the existing task offloading strategies follow centralized task scheduling approach. In centralized approach, all the tasks are submitted to the centralized broker which makes task scheduling decisions [26]. The major drawbacks of this strategy include additional latency due to decisions being transmitted between centralized broker and other computational servers, additional network overhead, chances of centralized broker being susceptible to single point of failure and scalability issues. Moreover, the existing approaches mostly focus on meeting user-defined tolerable latency without simultaneously focusing on reducing application response time which is desirable in real-time IoT applications. Therefore, we propose to tackle the problem of task scheduling in a decentralized manner, where each computational server takes independent decisions on task scheduling based on its resource availability to not only meet pre-defined user's tolerable latency but also reduce the overall application response time. Additionally, the proposed policy reduces hop count of the application tasks and optimizes resources in the fog-cloud distributed environment.

Table 1 Comparison of discussed related works

Work	Application Type		Parameters Considered					
	Inter-dependent tasks	Mutually Independent tasks	Response Time	Hop Count	Tolerable latency	Distributed Placement	Resource Utilization	Scalability
Guevara and Fonseca [18]	✓		✓				✓	
Mahmud et al. [24]	✓				✓	✓	✓	✓
Skarlat et al. [31]	✓		✓		✓		✓	
Liu et al. [21]		✓						
Yang et al. [39]		✓	✓					
Ni et al. [27]	✓		✓					
Bitam et al. [5]		✓					✓	
Xuan-Qui et al. [28]	✓		✓		✓			
Brogi and Forti [7]	✓		✓					
Kafhali and Salah [14]		✓	✓				✓	
Guerrero et al. [17]	✓			✓		✓	✓	✓
Ramasubbareddy and Sasikala [29]		✓	✓					
Gupta [19]	✓					✓		✓
Deng et al. [12]		✓	✓					
Zeng et al. [41]	✓							
Craciunescu et al. [10]	✓		✓					
Gu et al. [16]		✓	✓					
Stravrinides and Kratza [33]	✓		✓		✓	✓		
This Work	✓		✓	✓	✓	✓	✓	✓

3 The system model

This section discusses the application model, fog resource model and overall computing system architecture for resource provisioning and task scheduling in the fog integrated cloud environment.

3.1 Application model

An IoT application is generally modelled as a workflow based application $W = (T, E)$ [35], where $T = \{t_1, t_2, \ldots, t_n\}$ is a set of vertices that denotes the application tasks and $E = \{e_{12}, e_{13}, \ldots, e_{mn}\}$ is a set of directed edges that represents control and data dependencies among the tasks. The data and control dependencies among the tasks t_i and t_j is represented by an edge e_{ij} where $e_{ij} \varepsilon E$; $t_i, t_j \varepsilon T$ and $t_i \neq t_j$.

The precedence constraint (depicted by e_{ij}) between the tasks t_i and t_j signifies that child task t_j cannot initiate to execute until parent task t_i has completed its execution. Therefore, a child task does not execute until each of its parent tasks are finished with the execution and the corresponding control and data dependencies are met. Figure 1 shows an exemplary workflow model for fall detection.

3.2 Fog resource model

Figure 2 below depicts the fog resource model considered in this work. The model consists of geographically distributed, hierarchically arranged, heterogeneous and resource constrained computation, storage and networking facilities.

The hierarchical structure of the fog nodes in the model conforms to the real world fog based architecture where the capacity of fog nodes increase while moving up in the hierarchy [2]. Further, the lower level fog nodes reside closer to the end-user devices and typically provide application interfaces. Most often, the fog nodes take their scheduling decisions based on the resource availability. Our work makes a number of assumptions including (i) the fog nodes at the same level are homogeneous while those at different levels are heterogeneous in terms of computational capacity (processing core count, RAM and Bandwidth) (ii) nodes from a given level connected to the same parent form a cluster and (iii) a particular fog node at any given time instance belongs to only one cluster. Nodes within a cluster interact with each other

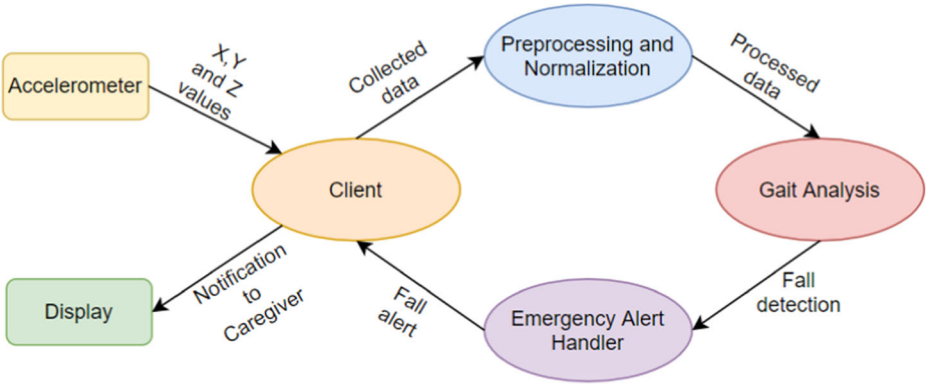


Fig. 1 Application model of elderly fall detection

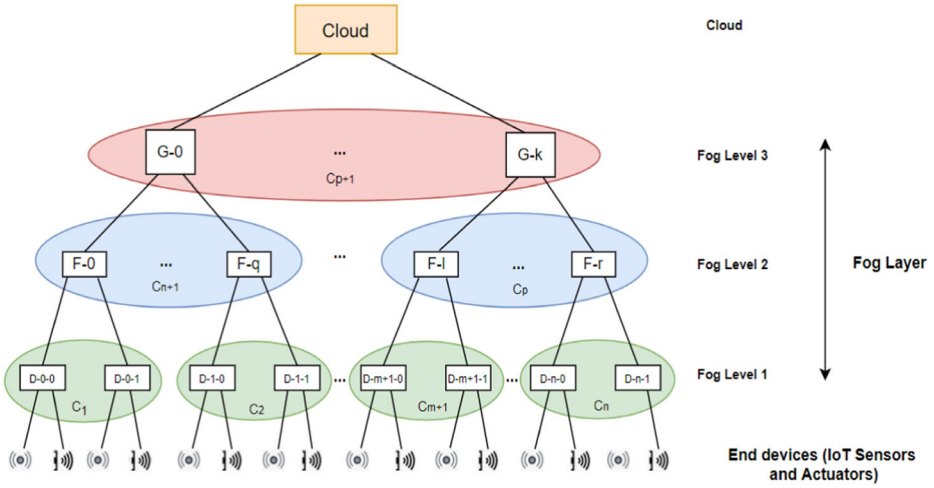


Fig. 2 Clusters in Fog Layers

through networking standards such as Constrained Application Protocol (CoAP), a REST based web transfer protocol. Inter-nodal communication latency is thus considered negligible among the nodes of the same cluster [24, 32].

3.3 System model task scheduling framework

Figure 3 depicts the task scheduling framework in a fog-cloud environment that is considered in this work. Each fog node has a component Resource Management System (RMS) responsible for scheduling the tasks of an application.

An end-user connects with the nearest fog node in order to submit the workflow together with its QoS requirements i.e. tolerable latency and resource specifications. Tolerable latency is the

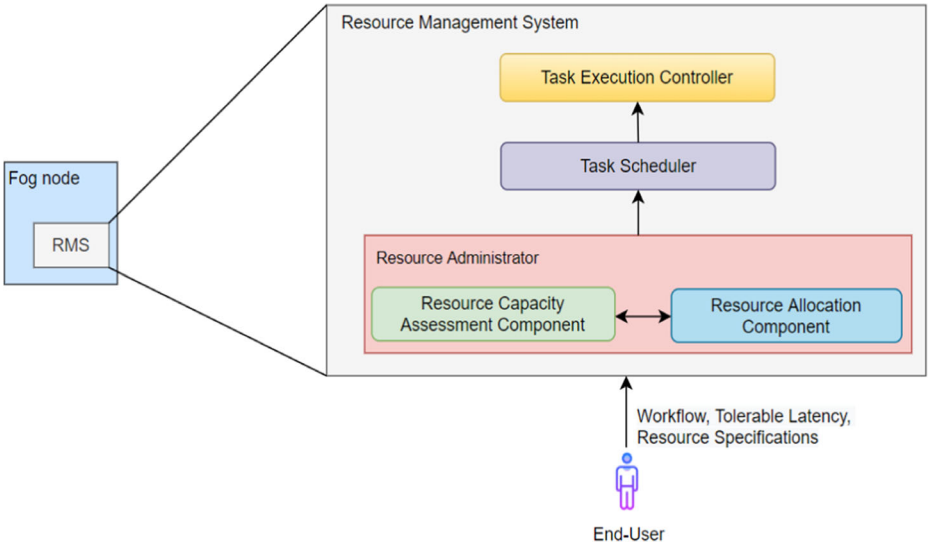


Fig. 3 System Model for Task Scheduling in Fog-Cloud Environment

maximum allowed time period up to which all tasks of an application should complete their execution while resource specifications describe the CPU, Memory or Bandwidth requirements of the application. After an application workflow is submitted to a fog node, the RMS in the node analyses the resource requirements, allocates the appropriate resources (if available), schedules and monitors execution of the application's tasks. If the required resources are not available, the fog node forwards it to other fog node in accordance with a predefined policy. Predominantly, RMS has three components: *Resource Administrator, Task Scheduler*, and *Task Execution Controller*. *Resource Administrator* component has further two sub-components: *Resource Capacity Assessment* component and *Resource Allocation* component. *Resource Capacity Assessment* component determines the resource requirements of a task while the *Resource Allocation* component allocates the required resources to the task and releases them after the task execution. The *Task Scheduler* keeps a track of the order of execution of the different tasks mapped onto the fog node while the *Task Execution Controller* executes the scheduled task.

4 Problem definition

Considering a group of n users denoted by the set $U = \{U_1, U_2, \ldots, U_n\}$ where each user U_i submits a workflow based IoT application A_i consisting of a set of tasks $M = \{t_1, t_2, \ldots, t_m\}$ along with its tolerable latency τ_i; the objective is to execute these applications on a fog-cloud infrastructure while honouring the tolerable latency and also improving the response time observed by the users.

Formally, given a set $S = \{(A_1, \tau_1), (A_2, \tau_2), \ldots, (A_n, \tau_n)\}$ of workflow based IoT applications, where each application A_i is associated with a user's tolerable latency τ_i, and a set of fog-cloud resources $FC = \{\{F_1, F_2, \ldots, F_j\}, \{C_1, C_2, \ldots, C_k\}\}$ with different processing capabilities, the objective is to find a schedule of the tasks of the different applications onto the fog-cloud resources such that the QoS requirements w.r.t tolerable latency are met and the response time is minimized. It is to be noted that the tolerable latency defined by the user depicts a threshold value for response time and if the observed response time exceeds the predefined tolerable latency, the user experience deteriorates considerably which is not desirable.

Evidently, the response time of an IoT application relies on two important factors: a) processing time of the different tasks, that depends upon how different tasks of the application are mapped onto the fog-cloud devices and b) transmission time required for task offloading and receiving the response back to the user device and is given by Eq. 1 as follows.

$$RT_i = \sum_t P_t^s + TT_i \quad : t \in A_i; s \in FC \tag{1}$$

where, P_t^s represents the processing time of application's task t scheduled on the computational server s. TT_i is the transmission time involved in offloading of tasks and for communication of control signals/data between the different tasks of the application that are hosted on diverse resources in the fog-cloud architecture. Specifically, transmission time is incurred when application tasks (including code and/or data) are offloaded on to the computational servers via an uplink channel or via a downlink channel during transmission of response to the requesting IoT devices.

Therefore, the total transmission time is the consolidation of time taken to relay the tasks to the computational resources and the time incurred to send the response from computational server to the user which is defined in Eq. 2 as follows.

$$TT_i = TT_i^u + TT_i^d \tag{2}$$

where, TT_i^u is the uplink transmission time and TT_i^d is the downlink transmission time defined in Eqs. 3 and 4 below.

$$TT_i^u = TT_{E-G}^u + \sum_{m-1} TT_{G-K}^u \tag{3}$$

where, E is the user end device, $G \varepsilon F$ is the client gateway, $K \varepsilon FC$ is any node within the fog-cloud infrastructure.

$$TT_i^d = TT_{S-E}^d \tag{4}$$

where, S is the node where the exit task of the application is scheduled.

Keeping the above points in consideration, a task scheduling algorithm for improved response time of IoT applications, IRTTS, is proposed in the next section. Table 2 lists the different notations used in the proposed algorithm.

Table 2 Notations

Symbol	Meaning
A	Set of applications in the system
τ	User-defined application tolerable latency
F	Set of fog nodes in the system
L	Set of all tasks in the system
M	Set of tasks belonging to same application $A_i \varepsilon A$; $M \subset L$
R	Set of all the resources (e.g. CPU, RAM, Bandwidth etc.)
G	Client Gateway
t_c	Current task to be scheduled
t_p	Previous task which is already scheduled
F_n	Current fog device under consideration for scheduling task $t_c \in M$
F_i^j	Fog node i on level j
$R_{Req}^{t_i}$	Resource requirements of task $t_i \in M$
$R_{occ}^{F_i}$	Amount of occupied resources on fog device $F_i \in$ F
$R_{cap}^{F_i}$	Maximum resource capacity of device $F_i \in$ F
$R_{cap}^{F_i^j}$	Maximum resource capacity of device F_i at level j; $F_i \in$ F
$R_{occ}^{F_i^j}$	Resource usage of device F_i at level j; $F_i \in$ F
R_{occ}^{C}	Amount of occupied resources on cloud
R_{cap}^{C}	Resource capacity of cloud
φ	Minimum Threshold limit for optimum resource usage
μ	Maximum Threshold limit for optimum resource usage
δ_{cd}	Perceived communication delay in the system (e.g. uplink latency)
ε	Equals to 1 if task is scheduled, 0 otherwise
C_i	Cluster of fog nodes
$P_{t_i}^{F_k}$	Processing time of task t_i in node $F_k \varepsilon F$; $t_i \varepsilon M$
B_{t_i}	Maximum allowed latency budget of each task $t_i \in M$
γ_{C_i}	Maximum inter-nodal communication delay in cluster C_i
Host(t_i)	Fog node where task t_i is scheduled; $t_i \in M$
Parent(F_i)	Parent of fog node $F_i \in$ F
TT(F_i,F_j)	Perceived communication delay between two nodes F_i and F_j; F_i, $F_j \in$ F
D_s	Sensor emitting data
D_a	Actuator receiving the response

5 The proposed task scheduling algorithm - IRTTS

In order to initiate an application's execution, the end-user connects with the nearest fog node and submits the application together with its specification. At this point, the scheduling process starts in coordination with the RMS (Fig. 3) of the fog node. Scheduling decisions are made by the *Resource Administrator* in coordination with the *Task Scheduler* of the RMS by employing a procedure – *ScheduleTask*. The procedure takes the following inputs: current node, to-be-placed task, previously placed tasks, perceived communication delay, application's tolerable latency, lower utilization threshold and an upper utilization threshold. Here, current node is the host where the parent task is scheduled and tolerable latency defines the maximum response time that can be observed by the end-user without adversely affecting the user-experience. For every task to be scheduled, the *Resource Administrator* along with *Task Scheduler* of the RMS computes *latency budget* for each task based on the task instruction length. This information, together with two utilization thresholds, *lower* and *upper,* is then used by the procedure *ScheduleTask* to find a task-node mapping. The objective behind using two utilization thresholds is to maintain a balance between the utilization of individual fog nodes, overall utilization of the fog-cloud infrastructure and the QoS requirements of the applications [15, 25]. As long as the resource utilization at lower level fog nodes is less than the lower threshold, any new task can be scheduled on these nodes, however when the utilization exceeds lower threshold, the proposed algorithm checks to see if the given task can be placed at upper level fog nodes without compromising the QoS. If the QoS is compromised, the new tasks are nevertheless placed at lower level nodes until the resource utilization at these node reaches the upper threshold. Thus, while lower threshold ensures that some amount of resources are always reserved at lower levels to cater to the applications with strict response-time requirements, upper utilization threshold prevents overprovisioning of the nodes.

The procedure starts by scheduling the entry task which is always scheduled at the nearest fog node known as the client gateway; once the entry task is scheduled on the client gateway, the scheduling process for the subsequent tasks is initiated which are then placed based on the application's tolerable latency and the resource constraints of the fog nodes. If a parent task is already scheduled on a given node, the S*cheduleTask* procedure checks to see if the child task can also be placed at the same node without exceeding the lower threshold. If it is not feasible to host the task on the same node where parent task is scheduled, task's latency budget is used to check if it can be routed to an uplink node. In the case when latency budget of the task does not allow it to be hosted on an upper level fog node, the procedure checks if upper threshold capacity of the current node can accommodate the task; otherwise, it tries to schedule the task onto some other node in the same cluster with an inter-nodal delay of γ_c. If no node in the cluster has adequate resources to host the task then the task is forwarded to its uplink node with perceived communication latency δ_{cd}, which might possibly result in QoS violation. This cycle continues until the current node is cloud; all the unmapped tasks are then scheduled onto the cloud resources. The run time complexity of the proposed algorithm is O(m*(c + k)), where c and k represent cluster size and number of levels in fog-cloud hierarchical infrastructure respectively. The pseudo-code of this procedure is shown in Algorithm 1.

Algorithm 1 IRTTS Task Scheduling Algorithm - ScheduleTask

ScheduleTask (F_n, t_c, t_p, δ_{cd}, τ, φ, μ)

1. ***if*** t_c is not null
2. $F_l \leftarrow$ Host(t_p) //F_l would be same as F_n for the child of entry task
3. ***if*** F_n is Cloud ***then***
4. Schedule unmapped tasks on Cloud
5. ***else***
6. ***if*** $\left(R_{Req}^{t_c} \leq \left(R_{cap}^{F_n} * \varphi\right) - R_{occ}^{F_n}\right)$ //φ is the lower threshold
7. $R_{occ}^{F_n} \leftarrow R_{occ}^{F_n} + R_{Req}^{t_c}$
8. $\delta_{cd} \leftarrow \delta_{cd} + TT(F_l, F_n)$ // TT(F_l, F_n) would be zero if $F_l = F_n$
9. TL $\leftarrow$ TL $+ TT(F_l, F_n) + P_{t_c}^{F_n}$
10. Increment count_tasks and set F_l to F_n
11. Add t_c to list of placed tasks on node F_n
12. Set ε to true
13. ***else***
14. $F_p \leftarrow$ Parent (F_n)
15. ***if***($TL + P_{t_c}^{F_p} + TT(F_l, F_p)) \leq B_{t_c}$ //Check if task can be scheduled on parent node
16. ScheduleTasks (F_p , t_c , t_p, δ_{cd}, τ, φ, μ)
17. ***else***
18. ***if*** $(R_{Req}^{t_c} \leq (R_{cap}^{F_n} * \mu) - R_{occ}^{F_n})$ //μ is the upper threshold
19. $R_{occ}^{F_n} \leftarrow R_{occ}^{F_n} + R_{Req}^{t_c}$
20. $\delta_{cd} \leftarrow \delta_{cd} + TT(F_l, F_n)$
21. TL $\leftarrow$ TL $+ TT(F_l, F_n) + P_{t_c}^{F_n}$
22. Increment count_tasks and set F_l to F_n
23. Add t_c to list of placed tasks on node F_n
24. Set ε to true
25. ***else*** //Check cluster nodes of current node F_n
26. *For each node F_i in Cluster C_n*
27. ***if*** ($R_{Req}^{t_c} \leq (R_{cap}^{F_i} * \mu) - R_{occ}^{F_i}$)
28. $R_{occ}^{F_i} \leftarrow R_{occ}^{F_i} + R_{Req}^{t_c}$
29. $\delta_{cd} \leftarrow \delta_{cd} + TT(F_l, F_i)$
30. TL $\leftarrow$ TL $+ TT(F_l, F_i) + P_{t_c}^{F_i}$
31. $count\_tasks \leftarrow count\_tasks + 1$
32. $F_l \leftarrow F_i$
33. Add t_c to list of placed tasks on F_i
34. set ε to true
35. *break*
36. ***End-if***
37. ***End-if***
38. ***End-if***
39. ***End-if***
40. ***End-if***
41. ***if*** (ε == false)
42. $F_p \leftarrow$ Parent (F_n)
43. ScheduleTasks (F_p , t_c , t_p, δ_{cd} , τ, φ, μ) //Scheduling the task on uplink node
44. ***End-if***
45. $t_p \leftarrow t_c$
46. $t_c \leftarrow$ Fetch the next task from the list of tasks, M
47. ScheduleTasks(F_n, t_c, t_p, δ_{cd} , τ, φ, μ)
48. ***if*** exit task is scheduled on F_n
49. $RT \leftarrow$ TL $+$ TT(F_n, ud) //Response time is calculated
50. ***End-if***
51. ***else***
52. Return
53. ***End-if***

6 Performance evaluation

This section evaluates the performance of our proposed algorithm by conducting simulation tests in iFogSim [19]. We evaluated our proposed task scheduling technique against centralized task placement policy CASSIA-RR [18] as well as distributed techniques LAMP [24] and Edgeward [19]. CASSIA-RR [18] is a recently proposed task scheduling policy aimed at minimization of makespan of IoT applications. The Latency-aware module placement (LAMP) algorithm [24], is one of the most cited algorithms for placing application modules in distributed fog environment while meeting the service delivery latency. Edgeward [19], on the other hand, is an in-built distributed task scheduling strategy in iFogSim.

CASSIA-RR is a centralized task scheduling policy which employs randomized rounding technique to develop an approximation to the Integer Linear Programming based exact CASSIA-INT strategy. The authors employed a classifier at network's edge to label IoT applications as Best-effort, CPU-bound, Streaming, Conversational, Interactive, Real-Time or Mission-critical. Based on the class of application, appropriate processing layer is chosen where the task is scheduled. LAMP policy considers DAG-based IoT applications. It treats applications with strict tolerable latency differently from the ones with relaxed tolerable latency requirements. It schedules the tasks with stricter latency requirements horizontally in the cluster while the tasks with relaxed latency requirements are placed vertically upwards. Additionally, this policy performs forwarding of the modules in order to optimize the resources. Edgeward on the other hand, schedules application's tasks following First-In-First-Scheduled strategy on leaf-to-root paths between users and the cloud on the basis of availability of resources on the computational servers. The task instances from different paths are merged if scheduled on the same device and migrated to parent if required. Both these algorithms (LAMP and Edgeward) take local decisions for scheduling the task and does not get influenced by any other entity similar to the working of the policy proposed in our work. However, Edgeward [19] is a general policy, that schedules tasks based on the resource constraints of the resources and does not aim towards satisfying the user-specified tolerable latency that has been taken into account in CASSIA-RR [18], LAMP [24] as well as in our work.

For evaluating the effectiveness of our proposed algorithm, a tolerable latency is necessary to be defined for each considered application. If the tolerable latency is set considerably high then there would be sufficient slack time to accommodate all the application tasks on the cloud which would result in a longer observed response time. Therefore, a comprehensive analysis demands evaluation on all possible tolerable latencies: Strict, Moderate, and Relaxed. Different tolerable latency range are defined according to the rule mentioned in Eq. 5.

$$Tolerable\ Latency\ (\tau) = (1-\alpha)*RL(A_i) \tag{5}$$

where:

a) $RL(A_i)$ is the maximum tolerable latency, beyond which the user experience is adversely affected. For the experiments, Maximum Tolerable latency $RL(A_i)$ of applications, are defined similar to the tolerable latencies for real-time applications [34].
b) α is the tolerable latency factor defined in Table 3.

Table 3 Tolerable latency Range

Tolerable latency	Range
Strict	$0.5<\alpha\leq0.75$
Moderate	$0.25<\alpha\leq0.5$
Relaxed	$0<\alpha\leq0.25$

For the experiments, in each category, tolerable latency value were randomly generated within the specified range.

6.1 Experimental setup

For evaluating the performance of our proposed work, a fog-cloud environment is modelled as a tree-based topology where nodes are assumed to be hierarchically arranged across four levels, comprising of three levels of fog nodes-FL_1, FL_2 and FL_3 and a top-most level of cloud resources hosted in datacentres. The resource capacity of fog nodes in terms of bandwidth and storage is considered sufficiently high to accommodate all the tasks assuming computational capacity as the only constraint for scheduling the CPU intensive tasks.

To evaluate the effectiveness of the proposed policy, three IoT-based applications are simulated (Table 4), similar to the ones employed in [5]. The instruction lengths of different tasks in these applications have been chosen to represent three different types of applications - light-weight (A_1), medium-weight (A_2) and heavy-weight(A_3). The topology of all the applications is similar to the one shown in Fig. 4.

To evaluate the proposed algorithm on different configurations of fog-cloud architecture, different simulation environments were created by changing the numeration of fog resources and the count of connected end-users. The simulation parameters employed in the experimentation are listed in Table 5. Experiments were performed on each application by considering various possible values from the specified range of tolerable latencies.

6.2 Experimental results

An extensive evaluation of the presented approach and its comparison with CASSIA-RR [18], Latency Aware Module Placement (LAMP) [24] and Edgeward [19] is performed. To this end, fog environments with different topology configurations are simulated and experiments are

Table 4 Instruction length of each task of an application

Application	$Task_1$	$Task_2$	$Task_3$	$Task_4$
A_1	1.5×10^9	1×10^9	2×10^9	0.5×10^9
A_2	1×10^9	1.5×10^9	3×10^9	1×10^9
A_3	0.5×10^9	2×10^9	3.5×10^9	2×10^9

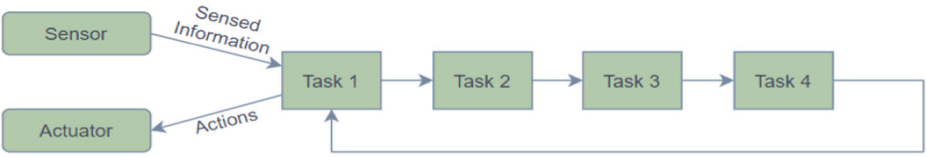


Fig. 4 Topology of IoT Application

Table 5 Simulation parameters and their values

Evaluation Parameters	Value
Processing Rate (MIPS)	
Cloud	20,00,000
Fog Node Level 3	12,000
Fog Node Level 2	8000
Fog Node Level 1	4000
Topology Configuration	
No. of Gateways	[4, 8, 40]
No. of Fog nodes per gateway	[4, 8, 40]
No. of Users per Fog Node	[8, 37]
Communication Delay (ms)	
Fog Node Level 3 to Cloud	85–125
Fog Node Level 2 to Fog Node Level 3	55–75
Fog Node Level 1 to Fog Node Level 2	25–45
End device to Fog Node Level 1	5–15
Inter-nodal communication delay among Fog Cluster Nodes (ms)	2–5
No. of Applications	3
No. of users	[8–256]
Fractional Selectivity	1

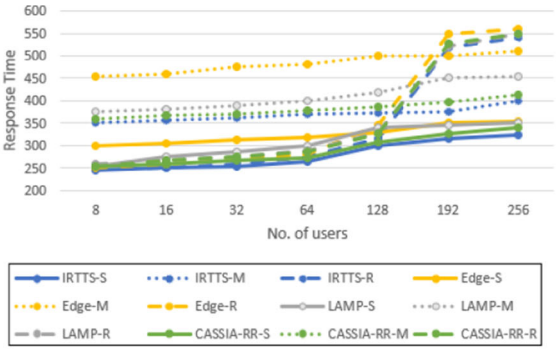
carried out to assess the proposed algorithm in terms of response time, QoS requirements met, CPU utilization and tasks distribution under different values of tolerable latencies depicting different applications with strict, moderate and relaxed latency requirements.

Response time We measure the response time separately for each application by varying the number of users under three different constraints on tolerable latency i.e. strict, moderate and relaxed. In the following figures, IRTTS represents the outcomes of our proposed algorithm, and those of Edgeward, Latency-aware module placement and CASSIA-Randomized Rounding are reflected by Edge, LAMP and CASSIA-RR respectively.

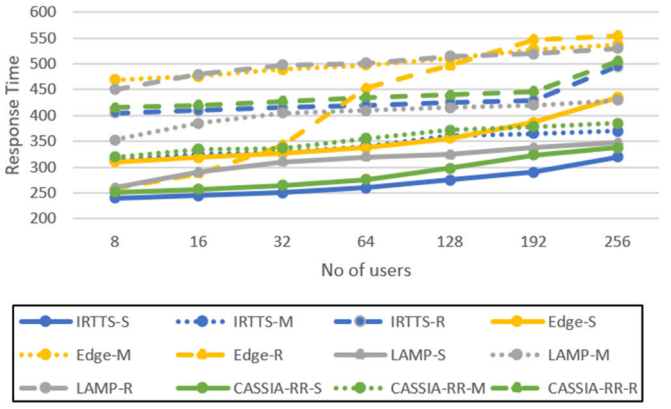
From the analysis of the results in Fig. 5(i), 5(ii), and 5(iii), it can be observed that IRTTS delivers the least response time in comparison to Edge, LAMP and CASSIA-RR. This is because the proposed approach schedules inter-dependent tasks on the same computational device, if possible, or on the devices in the same cluster. This results in reduction of the communication delay thereby, significantly improving the response time of the applications. However, all the three policies Edge, LAMP and CASSIA-RR do not favor task mapping near the end-users. Specifically, LAMP tries to place the tasks as far away from the users as possible unless QoS in terms of application delivery deadline is not violated; CASSIA-RR maps the tasks based on the assigned application label at the suitable distant layer of the hierarchical fog-cloud infrastructure beyond which QoS tends to get dishonored; Edge on the other hand does not examine the possibility of horizontal task mapping on the cluster nodes, consequently, forwarding the tasks to higher levels whenever current node is not able to host the given task. Our proposed policy, therefore, outperforms all the three policies Edge, LAMP and CASSIA-RR for each category of tolerable latency. It is significant to mention here that each evaluation test is performed at least 25 times and the average of the results obtained are depicted.

QoS requirements met Figure 6(i), (ii), and (iii) represent the overall percentage of schedules satisfying Strict, Moderate and Relaxed tolerable latencies for all three types of applications

(i) Application A_1



(ii) Application A_2



(iii) Application A_3

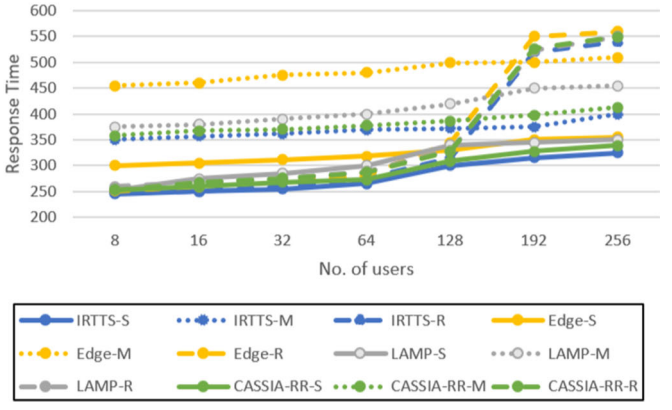
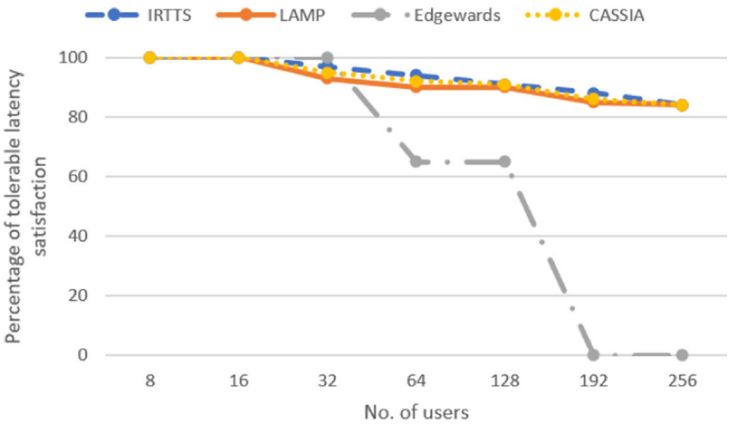
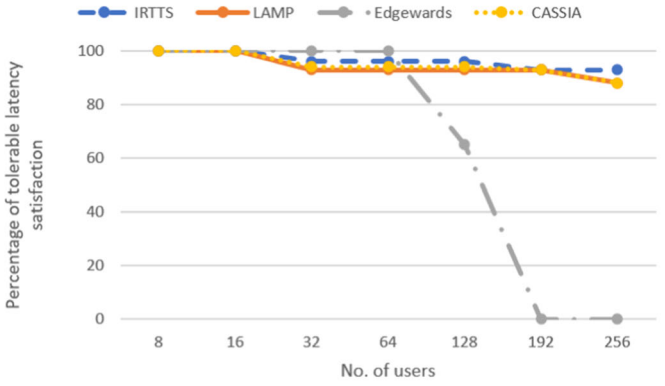


Fig. 5 Response Time of applications under different Tolerable latency Constraints

(i) Strict Tolerable Latency



(ii) Moderate Tolerable Latency



(iii) Relaxed Tolerable latency

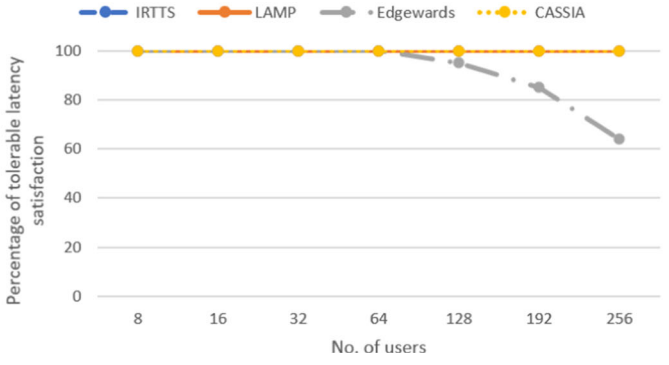


Fig. 6 Percentage of tolerable latency met for different number of users and tolerable latencies

under varying number of users. As can be seen, IRTTS performs better even for the applications with strict tolerable latency requirements. This is because our policy reserves some amount of resources at the lower level fog nodes to support applications with strict latency requirements. Table 6 below explicitly depicts the percentage of users of the different applications A_1, A_2 and A_3 for which the latency observed was within the tolerable latency defined for the various categories-Strict, Moderate and Relaxed. Evidently, the proposed IRTTS algorithm outperforms all the compared algortihms, followed by CASSIA-RR and LAMP while Edge exhibits lowest number of applications that are able to meet their QoS requirements. This is so because the proposed scheduling algorithm maps tasks onto computational servers with an objective to minimize the application's response time and therefore tries to place the tasks as close to the user as possible. On the contrary, the other three policies do not aim better response time and therefore always place tasks away from the user. Since fog-cloud resources may suffer from performance variation due to sharing of underlying resources and virtualization [30], such an approach may lead to situations where tasks are hosted on to distant resources and applications are not able to meet their QoS requirements due to the associated communication delays. The proposed algorithm therefore is able to deliver a better performance in comparison to Edge, LAMP and CASSIA-RR policies even in the presence of performace variation of fog-cloud resources.

CPU utilization Figure 7 represents the CPU utilization of the computational resources at each hierarchical level of fog-cloud infrastructure for different tolerable latencies i.e. Strict, Moderate and Relaxed. CPU utilization of a computational device is calculated as the ratio between utilized and available CPU resources. CPU usage of each of the three fog levels and cloud (Level 4), is therefore computed by taking mean of the CPU utilization of all the nodes at that level as shown in Eq. 6. CPU usage of the cloud is calculated as shown in Eq. 7. Each point corresponds to mean of the level-wise CPU utilization for all the applications.

$$U_F(j) = \frac{\sum_{|F_i^j|} \left(\frac{R_{occ}^{F_i^j}}{R_{cap}^{F_i^j}} \right)}{|F_i^j|} \tag{6}$$

where, F_i^j is the i^{th} computational node at level j of the fog-cloud hierarchical structure.

Table 6 Percentage of tolerable latency met for different applications and tolerable latency factors

Tolerable Latency Factor	Technique	A_1	A_2	A_3
Strict	*Edgeward*	71.4	71.4	42.85
	LAMP	97.14	88.57	85.71
	CASSIA-RR	97.14	89.32	86.24
	IRTTS	97.14	91.43	88.57
Moderate	*Edgeward*	71.4	71.4	57.14
	LAMP	97.14	91.43	91.43
	CASSIA-RR	97.14	92.35	92.36
	IRTTS	100	94.28	94.28
Relaxed	*Edgeward*	94.28	94.28	88.57
	LAMP	100	100	100
	CASSIA-RR	100	100	100
	IRTTS	100	100	100

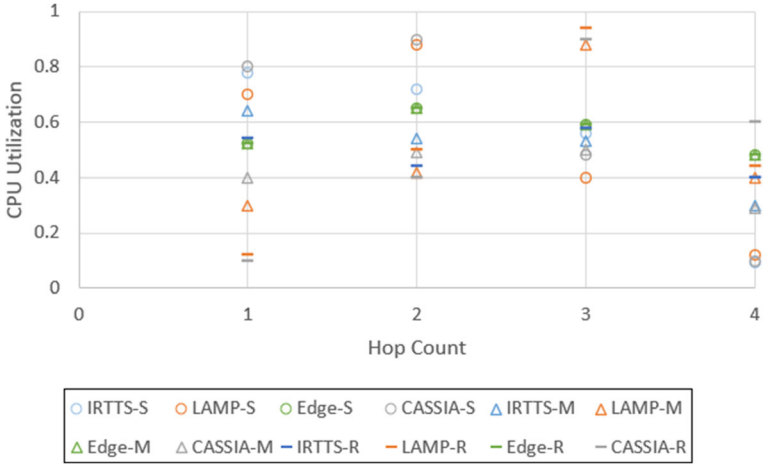


Fig. 7 CPU utilization of computational servers

$$U_C(j) = \frac{R^C_{occ}}{R^C_{cap}} \tag{7}$$

For each value belonging to the defined range of Strict, Moderate or Relaxed tolerable latency; CPU utilization for the proposed policy is highest for the level near to the end-users (i.e. lowest level) and lowest for the resources at the cloud whereas no such pattern can be observed for the compared policies Edge, LAMP and CASSIA-RR. This is because of the fact that the proposed policy tries to utilize the lower level resources before mapping the tasks on the resources at higher levels. In addition, IRTTS ensures that utilization of the resources does not go beyond the upper threshold in order to prevent performance degradation. However, Edge, LAMP and CASSIA-RR utilize computational servers that are distant from the users and therefore many a times result in sub-optimal user-observed performance.

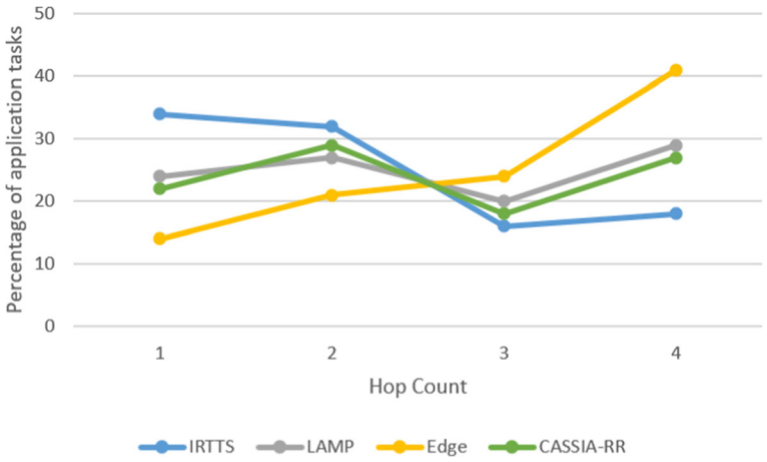


Fig. 8 Percentage of tasks scheduled and level number

Tasks distribution Figure 8 shows the distribution of tasks at different levels of fog-cloud hierarchical infrastructure. To measure the proximity degree of task and client, we used hop count metric. Hop count depicts the level number of the computational server hosting the task. The lower the value of hop count, the higher is the task proximity. The optimization objective of our algorithm is to improve the response time which is intrinsically achieved by reducing the value of hops for the application tasks. From the figure, it can be inferred that IRTTS schedules more number of tasks at lower levels whereas Edge, LAMP and CASSIA-RR map majority of the tasks at the higher levels. Since Edge, LAMP and CASSIA-RR tend to map the applications with borderline latency requirements on the resources at higher levels, overall response time for these applications might increase. The proposed algorithm hosts such applications at lower levels thereby lowering the response time and hence the hop count.

In conclusion, we can say that our policy reduces the application's response time, makes better use of available resources, and also tries to place the user tasks as proximal to the user's location as feasible. The proposed approach thus results in reduced communication delays in comparison to the other three policies.

7 Conclusion

Fog Computing environment offers enormous possibilities for executing IoT-based applications. In this work, we proposed a solution to the problem of scheduling IoT applications on to the hierarchically distributed fog and cloud resources in order to meet the user-specified latency constraint and simultaneously, improving the response time which is desirable for real-time IoT based applications. Towards this, a decentralized heuristic algorithm is designed which tries to assign as many tasks to resources in the bottom tier as feasible without causing resource over-utilization, while also preserving enough resources to handle applications with strict response time requirements, making it suitable for handling large-scale real-time applications. We evaluated our proposed approach by considering different categories of tolerable latency-strict, moderate and relaxed on three different IoT application profiles – light-weight, medium-weight and heavy-weight. Empirical results depict that our proposed policy outperforms the state-of-the-art heuristics by making efficient use of available resources, honouring the tolerable latency, reducing the response time observed by the users and consequently improving the hop count of the application tasks.

In future, we aim to improve this policy by reducing the operating cost, energy and memory consumption of fog and cloud resources. In addition, we aim to extend this work to include user profiles for ranking instances of IoT applications.

Declaration

Conflict of interest The authors have no competing financial interests or personal relationships that influence the work reported in this paper.

References

1. Architecture Working Group OpenFog Consortium (2017) Openfog reference architecture for fog computing. OPFRA001 20817, 162
2. Ashrafi TH, Hossain MA, Arefin SE, Das KD, Chakrabarty A (2018) IoT Infrastructure: Fog Computing Surpasses Cloud Computing. In: Iot infrastructure: fog computing surpasses cloud computing, In intelligent communication and computational technologies (pp. 43–55). Springer, Singapore
3. Basu S, Karuppiah M, Selvakumar K, Li KC, Islam SH, Hassan MM, Bhuiyan MZA (2018) An intelligent/cognitive model of task scheduling for IoT applications in cloud computing environment. Futur Gener Comput Syst 88:254–261
4. Biswas R, Giaffreda R (2014) IoT and cloud convergence: opportunities and challenges. In 2014 IEEE world forum on internet of things (WF-IoT) (pp. 375-376). IEEE
5. Bitam S, Zeadally S, Mellouk A (2018) Fog computing job scheduling optimization based on bees swarm. Enterprise Inform Syst 12(4):373–397
6. Bonomi F, et al (2012) "Fog computing and its role in the internet of things." Proceedings of the first edition of the MCC workshop on Mobile cloud computing
7. Brogi A, Forti S (2017) QoS-aware deployment of IoT applications through the fog. IEEE Internet Things J 4(5):1185–1192
8. Cai H, Xu B, Jiang L, Vasilakos AV (2016) IoT-based big data storage systems in cloud computing: perspectives and challenges. IEEE Internet Things J 4(1):75–87
9. Cisco delivers vision of fog computing to accelerate value from billions of connected devices (2014) Press release. Cisco. [Online]. Available: http://newsroom.cisco.com/release/1334100/Cisco-Delivers-Vision-of-Fog-Computing-to-Accelerate-Value-from-Billionsof-Connected-Devices-utm-medium-rss.
10. Craciunescu R, Mihovska A, Mihaylov M, Kyriazakos S, Prasad R, Halunga S (2015) Implementation of fog computing for reliable E-health applications. In 2015 49th Asilomar conference on signals, systems and computers (pp. 459-463). IEEE
11. Dastjerdi AV, Buyya R (2016) Fog computing: helping the internet of things realize its potential. Computer 49(8):112–116
12. Deng R, Lu R, Lai C, Luan TH, Liang H (2016) Optimal workload allocation in fog-cloud computing toward balanced delay and power consumption. IEEE Internet Things J 3(6):1171–1181
13. Doukas C, Maglogiannis I (2012) Bringing IoT and cloud computing towards pervasive healthcare. In 2012 sixth international conference on innovative Mobile and internet Services in Ubiquitous Computing (pp. 922-926). IEEE
14. El Kafhali S, Salah K (2017) Efficient and dynamic scaling of fog nodes for IoT devices. J Supercomput 73(12):5261–5284
15. Goswami P, Mukherjee A, Maiti M, Tyagi SKS, Yang L (2021) A neural network based optimal resource allocation method for secure IIoT network. IEEE Internet of Things Journal
16. Gu L, Zeng D, Guo S, Barnawi A, Xiang Y (2015) Cost efficient resource management in fog computing supported medical cyber-physical system. IEEE Trans Emerg Top Comput 5(1):108–119
17. Guerrero C, Lera I, Juiz C (2019) A lightweight decentralized service placement policy for performance optimization in fog computing. J Ambient Intell Humaniz Comput 10(6):2435–2452
18. Guevara JC, da Fonseca NL (2021) Task scheduling in cloud-fog computing systems. Peer-to-Peer Network Appl 14(2):962–977
19. Gupta H, Vahid Dastjerdi A, Ghosh SK, Buyya R (2017) iFogSim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. Software: Prac Exp 47(9):1275–1296
20. Johnson DS, Garey M (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman&Co, San Francisco
21. Liu L, Chang Z, Guo X, Mao S, Ristaniemi T (2017) Multiobjective optimization for computation offloading in fog computing. IEEE Int Things J 5(1):283–294
22. Lord SR, Sherrington C, Menz HB, Close JC (2007) Falls in older people: risk factors and strategies for prevention. Cambridge University Press, Cambridge, GB
23. Mahmud R, Kotagiri R, Buyya R (2018) Fog computing: a taxonomy, survey and future directions. In: Internet of everything (pp. 103–130). Springer, Singapore

24. Mahmud R, Ramamohanarao K, Buyya R (2018) Latency-aware application module management for fog computing environments. ACM Transac Int Technol (TOIT) 19(1):1–21
25. Maiti M, Krakovich V, Shams SR, Vukovic DB (2020) Resource-based model for small innovative enterprises. Manag Decis 58:1525–1541
26. Nguyen T, Doan K, Nguyen G, Nguyen BM (2020) Modeling multi-constrained fog-cloud environment for task scheduling problem. In *2020 IEEE 19th international symposium on network computing and applications (NCA)* (pp. 1-10). IEEE
27. Ni L, Zhang J, Jiang C, Yan C, Yu K (2017) Resource allocation strategy in fog computing based on priced timed petri nets. IEEE Internet Things J 4(5):1216–1228
28. Pham XQ, Man ND, Tri NDT, Thai NQ, Huh EN (2017) A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing. Int J Distri Sensor Netw 13(11): 1550147717742073
29. Ramasubbareddy S, Sasikala R (2019) RTTSMCE: a response time aware task scheduling in multi-cloudlet environment. International journal of computers and applications, 1-6
30. Schad J, Dittrich J, Quiané-Ruiz JA (2010) Runtime measurements in the cloud: observing, analyzing, and reducing variance. Proc VLDB Endowment 3(1–2):460–471
31. Skarlat O, Nardelli M, Schulte S, Borkowski M, Leitner P (2017) Optimized IoT service placement in the fog. SOCA 11(4):427–443
32. Slabicki M, Grochla K (2016) Performance evaluation of CoAP, SNMP and NETCONF protocols in fog computing architecture. In NOMS 2016-2016 IEEE/IFIP network operations and management symposium (pp. 1315-1319). IEEE
33. Stavrinides GL, Karatza HD (2019) A hybrid approach to scheduling real-time IoT workflows in fog and cloud environments. Multimed Tools Appl 78(17):24639–24655
34. Suznjevic M, Saldana J (2015) Delay limits for real-time services
35. Taneja M, Jalodia N, Davy A (2019) Distributed decomposed data analytics in fog enabled IoT deployments. IEEE Access 7:40969–40981
36. Vermesan O, Friess P (2014) Internet of things applications-from research and innovation to market deployment. Taylor & Francis, p 364
37. Wang L, Ranjan R (2015) Processing distributed internet of things data in clouds. IEEE Cloud Comput 2(1): 76–80
38. Wild D, Nayak U, Isaacs B (1981) How dangerous are falls in old people at home? Br Med J (Clin Res Ed) 282(6260):–266
39. Yang Y, Zhao S, Zhang W, Chen Y, Luo X, Wang J (2018) DEBTS: delay energy balanced task scheduling in homogeneous fog networks. IEEE Internet Things J 5(3):2094–2106
40. Yi S, et al (2015) "Fog computing: Platform and applications." 2015 Third IEEE workshop on hot topics in web systems and technologies (HotWeb). IEEE
41. Zeng D, Gu L, Yao H (2020) Towards energy efficient service composition in green energy powered cyber–physical fog systems. Futur Gener Comput Syst 105:757–765
42. Zhang T, Wang J, Liu P, Hou J (2006) Fall detection by embedding an accelerometer in cellphone and using KFD algorithm. Int J Comput Sci Network Sec 6(10):277–284