Container-based task scheduling for edge computing in IoT-cloud environment using improved HBF optimisation algorithm

Srichandan Sobhanayak* and Kavita Jaiswal

IIIT Bhubaneswar,
Odisha 751003, India
Email: srichandan@iiit-bh.ac.in
Email: a116010@iiit-bh.ac.in
*Corresponding author

Ashok Kumar Turuk and Bibhudatta Sahoo

NIT Rourkela,
Odisha 769008, India
Email: akturuk@nitrkl.ac.in
Email: bdsahu@nitrkl.ac.in

Bhabendu Kumar Mohanta and Debasish Jena

IIIT Bhubaneswar,
Odisha 751003, India
Email: c116004@iiit-bh.ac.in
Email: debasish@iiit-bh.ac.in

Abstract: In conventional cloud computing technology, cloud resources are provided centrally by massive data centres. Therefore, edge computing technology has been proposed, where cloud services can be extended to the edge of the network to decrease a network congestion. The management of the resources is a major challenge before the researcher. Therefore, in this paper, a task scheduling algorithm based on hybrid bacteria foraging optimisation (HBFA) has been proposed for allocating and executing an application's tasks. The proposed algorithm aims to minimise the completion time and maximise resource utilisation in the edge network. A rigorous simulation has been done to test performance of the proposed strategy to compare it with state of art algorithms. The proposed strategy shows better performance compared to the existing work.

Keywords: internet of things; IoT; cloud; edge computing; container.

Reference to this paper should be made as follows: Sobhanayak, S., Jaiswal, K., Turuk, A.K., Sahoo, B., Mohanta, B.K. and Jena, D. (2020) 'Container-based task scheduling for edge computing in IoT-cloud environment using improved HBF optimisation algorithm', *Int. J. Embedded Systems*, Vol. 13, No. 1, pp.85–100.

Biographical notes: Srichandan Sobhanayak is currently working as an Assistant Professor at the Department of Computer Science and Engineering, IIIT Bhubaneswar, Odisha, India. He received his PhD in Computer Science and Engineering from NIT Rourkela. His research interests include internet of things, cloud and grid computing, wireless network, network security, mobile computing, embedded system and distributed system. He has published more than 15 papers in international journals and conferences in these areas.

Kavita Jaiswal is currently pursuing her PhD at the Department of Computer Science and Engineering, NIT Raipur. She received her MTech degree in Computer Science and Engineering from IIIT Bhubaneswar in 2018 and completed her BE degree from GEC Bilaspur in 2015. She is also an IEEE student member. Her research interests includes internet of things, wireless sensor network, cloud computing and fog computing.

Ashok Kumar Turuk is currently working as a Professor at the Department of Computer Science and Engineering, National Institute of Technology, Rourkela, Odisha, India. He received his PhD in Computer Science and Engineering in 2005 from the Indian Institute of Technology, Kharagpur, India, and BTech and MTech in Computer Science and Engineering in 1992 and 2000, respectively from National Institute of Technology, India. His current research interests include optical network, wireless network, network security, mobile computing, embedded system and distributed system. He has published more than 50 papers in international journals and conferences in these areas.

Bibhudatta Sahoo is an Associate Professor at the Department of Computer Science and Engineering, National Institute of Technology, Rourkela, India. He is a member of the Communication and Computing Research Group, and a Professor in charge of Cloud Computing Research Laboratory. His research interests lie in the area of parallel and distributed systems, cloud computing, sensor network, algorithms for VLSI design, internet of things, software defined networks, multicore architecture, 5G networks and algorithmic engineering.

Bhabendu Kumar Mohanta received his BTech degree in Information Technology from V.S.S. University of Technology in 2007 and his MTech degree from College of Engineering and Technology, Bhubaneswar in 2012. Presently, he is pursuing his PhD at International Institute of Information Technology (IIIT) Bhubaneshwar. His research focuses are information security and IoT security and blockchain technology. He is also an IEEE student member.

Debasish Jena received his BTech degree in Computer Science and Engineering, his Management degree and his MTech degree in 1991, 1997 and 2002, respectively. He received his PhD from NIT Rourkela in 2010. Currently, he is working as an Associate Professor at IIIT Bhubaneshwar. In addition to his responsibility, he was also IT, Consultant to Health Society, Govt. of Orissa for a period of two years from 2004 to 2006. His research areas of interest are information security, cloud security, IoT security and blockchain. His professional memberships include IEEE, ACM, ISTE, IACSIT, MIE (I), CSI and OITS.

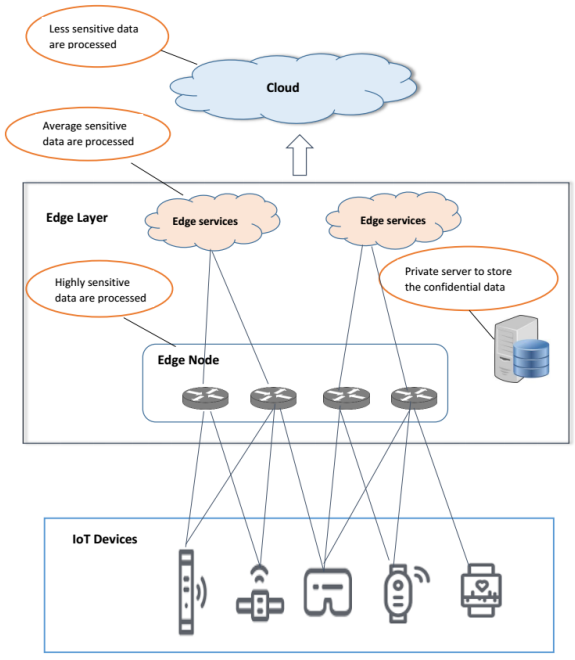
This paper is a revised and expanded version of a paper entitled 'IoT-cloud based framework for patient's data collection in smart healthcare system using Raspberry-Pi' presented at International Conference on Electrical and Computing Technologies and Applications (ICECTA) 2017, Ras Al Khaimah, United Arab Emirates, 21–23 November 2017.

1 Introduction

Internet of things (IoT) is the essential technology to form smart cities since it facilitates everyday objects or entities to provide information and services to users by collaborating and communicating with each other (Zhang et al., 2010). There is an explosive growth in the number of devices getting connected to the network with the rapid development of the IoT. When these devices simultaneously request resource services from the cloud data centre, they will take up a large amount of network bandwidth, and data transmission and information access will become slower. Moreover, if some delay-sensitive requests such as medical and emergency are uploaded to the remote cloud for processing, the delay caused by bandwidth constraints and resource bottlenecks of the cloud data centre will affect the quality of service (QoS). The conceptual approach that virtualises IoT devices and runs the user's task on the virtualised resources is known as edge/fog computing (Bonomi et al., 2014). The research community and academia are yet in search of the better definition of these terms (Elmroth et al., 2017). The theoretical foundation was established in many years back (Bonomi et al., 2014; Dastjerdi et al., 2016), but it fails to provide the solution for the resource provision and scheduling in IoT environment.

The architecture of edge computing is depicted in Figure 1 which introduces edge layer between the cloud and edge device to handle the delay-sensitive task. Fog computing is a paradigm for managing a highly distributed and possibly virtualised environment that provides network services between sensors and cloud data centres (Dastjerdi et al., 2016).

Figure 1 Architecture of edge computing (see online version for colours)



The fog or edge computing is a substantial change that is required to make IoT systems more efficient, scalable and robust. Scaling to billions of devices will only be possible if power is used efficiently through optimised computing and intelligent monitoring systems. So for this purpose Raspberry Pi is used as an edge device to optimise the process of analysing the sensor data with minimum power (Johnston and Cox, 2017).

Motivation: The proposed work is motivation from the real-time application that runs in European H2020 Factories of the Future Project Cloud-based Rapid Elastic Manufacturing (CREMA) (Schulte et al., 2014; Skarlat et al., 2016; Wu et al., 2013; Xu, 2012; Georgakopoulos et al., 2016; Kubler et al., 2016).

This paper extends our previous work (Jaiswal et al., 2017) with following added contributions:

- we formalise an optimisation solution that considers execution time of tasks and time limit on placement (deadline based)
- our objective is to maximise the resource utilisation in the edge network.

The rest of the paper is organised as follows: Section 2 gives literature survey, and in Sections 3 and 4, system framework and system model are discussed. In Section 5, we present the problem formulation. In Section 6, we propose a HBFA. Then the simulation results are presented in Section 7. Finally, the conclusion and future research directions are discussed in Section 8.

2 Literature survey

Sun et al. (2019) proposed an improved wolf colony search algorithm based on search strategy. Their work is of twofold: firstly, they introduce interaction strategy into travel behaviour and calling behaviour to promote the communication between artificial wolves, which can improve the information acquirement for wolves and enhance the exploring ability of wolves and secondly, they present adaptive siege strategy for siege behaviour, which guarantees that the new algorithm can obtain better collaborative search feature. Wang et al. (2019) proposed a cross-domain load balancing mechanism, CDLB, based on extensive messaging and presence protocol (XMPP) for SDN in the cloud data centre. Different from the poll method, XMPP-based push model is introduced in the proposed scheme, that can avoid wasting network and computing resources in a large-scale distributed network environment. Their proposed scheme enables all the controllers in the flat distributed control plane to share the same consistent global-view network information in real time through XMPP and XMPP publish/subscribe extension.

Singh (2018) proposed a hybrid of VNS, GA and PSO called HGVP in order to overcome the constraint of a poorly selected initial amount of particles in case of PSO-based scheduling for CCE.

Liu et al. (2015) proposed an energy-aware list-based scheduling algorithm called EALS for parallel applications in the context of service level agreement (SLA) on cloud data centres.

Zhang and Wang (2018) analysed advantages and shortages of classical load balancing algorithms based on dynamic feedback on a server cluster, and combined stimulated annealing with this strategy to put forward an optimised model of dynamic load balancing.

Digalwar et al. (2017) focus on dynamic energy optimisation using a well-established technique namely dynamic voltage and frequency scaling (DVFS).

The proposed algorithm guarantees periodic task deadlines and offers minimum aperiodic task response times. Huang et al. (2017) proposed a new task scheduling algorithm for heterogeneous computing platforms, called communication-aware earliest finish time (CEFT).

Xiong et al. (2017) proposed an improved energy-aware task consolidation algorithm to optimise the scheduling of tasks in cloud computing data centres. Their algorithm was developed based on the linear relationship between energy consumption and CPU utilisation. Tian and Yang (2019) proposed an efficient recovery scheme by using one-hop overlay multipath source routing, which is a post-failure recovery method.

They formulate the traffic allocation problem as tractable linear programming (LP) optimisation problem, whose goal is to minimise the worst-case network congestion ratio.

3 System framework

In this section, we discuss the framework of edge computing as appeared in Figure 2. The framework creates a real-time scenario of IoT tasks in an edge network. This helps to optimise the process of task placement and resource provisioning in the edge control node.

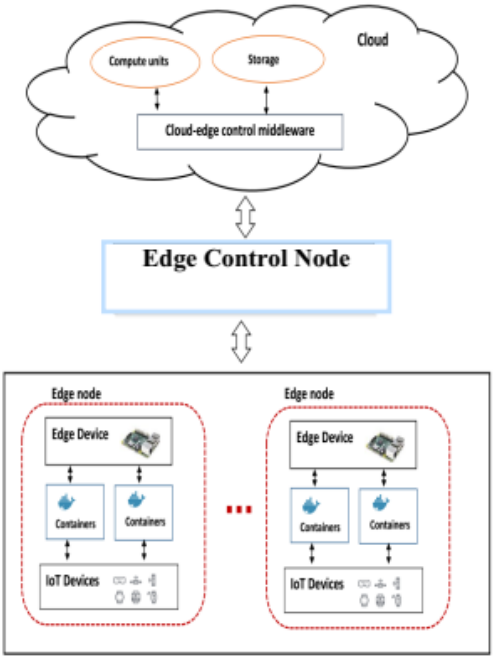
3.1 Cloud-edge control middleware

To control task unit execution in the cloud and support the edge network cloud-edge control middleware is presented as a central unit. The middleware manages the task which is not delayed sensitive and can't be executed by the edge control node.

3.2 Containers

To support virtualisation technology, we use containers and allow them to run on IoT devices. The sensor and actuator nodes connected to the IoT devices can be controlled and monitored through these virtualised containers.

Figure 2 Edge computing model (see online version for colours)



3.3 Edge control node

The main job of an edge control node is to support edge nodes. Edge control node accepts user's requests to execute IoT application. Furthermore, the edge control node will analyse the resource requirement of user's request and accordingly, it will assign the task to containers of an edge node or distribute the load to other edge node or in case the task cannot be handled by edge network then the request is propagated to the cloud-edge control middleware.

4 System model

The design goal of edge task placement problem (ETPP) is twofold, first, the model characterises edge network resources are given in Section 4.1 and second the IoT application model to be processed on edge resources is given in Section 4.2. Afterward, we design the ETPP in Section 5. In Table 1, we summarise the documentation utilised as a part of the ETPP.

4.1 Edge network

The execution and placement of tasks are supervised by the edge device H. It also manages the subordinate containers which are represented by the set Res(H). Containers $h^a \in Res(H)$ are associated with sensors and actuators. The edge control node coordinates all the communication within the edge network. As containers are deployed within the edge devices so the network link between the edge device and a particular container h^a is denoted by a negligible delay l_a^h. Edge devise and containers' placement utilisation is represented by P^H and P^{h^a}, respectively. Likewise, R^H and R^{h^a} denotes the RAM capacities of the edge device H and a container h^a, and M^H and M^{h^a} denotes the storage capacities of H and h^a, respectively. l^I denotes considerable delay between H and I.

Table 1 ETPP notation

Notation	*Definition*
c	Current time
τ	Interval between two rounds of the ETPP (in s)
O	Cloud
H	Edge device of the edge node
I	Nearest edge node
Res(H)	Containers in the edge node
P^H	Processing capacity of H
R^H	RAM capacity of H
M^H	Storage capacity of H
h^a	Containers in a edge node
P^{h^a}	Processing capacity of h^a
R^{h^a}	RAM capacity of h^a
M^{h^a}	Storage capacity of h^a
l^{h^a}	Link delay between H and h^a
l^o	Link delay between H and O
l^I	Link delay between H and I
S	Set of task units to be executed
S_n	Task unit
E_{S_n}	Task unit deadline
D_{S_n}	Queuing time of an task unit
K_{S_n}	Makespan of an task unit
t_{S_n}	Response time of an task unit
i_s	Task in an task unit
P_{i_s}	Processor demand of a task
R_{i_s}	RAM demand of a task
M_{i_s}	Storage demand of a task
K_{i_s}	Makespan of a task
Res_s^i(H)	All containers able to host a task i_s
N_d	Network delay
F_e	Finishing execution time of task
S_e	Starting execution time of task

A cloud-edge middleware is in charge of the resource utilisation in the cloud O; thus, it indicates a network bridge between the edge node and the cloud. Despite the fact that the edge nodes are thought to be self-governing, the cloud-edge control middleware can overrule the edge control node if necessary, e.g., if the failure occurs or edge node over-burden. The network link amongst H and O is represented by a considerable delay l^O.

4.2 IoT applications and tasks

The edge device in the edge node receives the requests for task unit executions. Suppose S be the set that consists of set of application to be executed that follows the DDF placement model (Giang et al., 2015). S_n consist of the set of task units where $S_n \in S$. The set S_n consists of task i_s where $i_s \in S_n$ needs to be placed in any

of the computational resources such as containers, the neighbouring node, and the cloud. To calculate the response time t_{S_n}, we consider:

1 the total makespan K_{S_n}

2 the total queuing time D_{S_n} of the task unit.

The total makespan of the task unit (K_{S_n}) is calculated as follows:

$$K_{S_n} = N_d + P_{i_s} \tag{1}$$

where $N_d \in \left\{l^{h^a}, l^o, l^I\right\}$, P_{i_s} is the makespan of each task $i_s \in S_n$ computed as follows:

$$P_{i_s} = F_e - S_e \tag{2}$$

where F_e is finish execution time of task, S_e starting execution time of task, and D_{S_n} is the total queuing time of the task unit which is calculated from extra time required to propagate the task to nearest node and the time required to deploy the task on current edge node which we denote as $D^c_{S_n}$. We represent the current queuing time as the elapsed time since the request for task unit execution, e.g., when a task unit is placed to a neighbour node, which may be stored in a variable $K^c_{S_n}$. The management operation that outlines and endorse task placement, accounted as task unit placement time of task unit D_{S_n} and follows our assumption. Every task unit has a deadline for queuing and execution E_{S_n} characterised by users of the task unit. Each task i_s is characterised by its requests of processor P_{i_s}, RAM R_{i_s}, and storage M_{i_s}, and by a task type. The task type specifies that a task i_s must be put on particular types of virtualised resources in the edge network. By considering all inclusive statement we define three distinctive task types viz. processing, sensing, and actuating tasks.

5 Problem formulation

The main responsibility of ETPP is to optimally schedule the task on the resource of an edge network. The functionality of ETPP is to run a task scheduling algorithm that places a task on any of the available resources, i.e., container, nearest node or the cloud. To process the execution of requested task units the edge control node continuously solves the ETPP, each τ time interval. At the point when the edge node receives the request, their scheduling is suspended to the next nearest optimisation round of the ETPP, designated by its start time c. When all the task unit's task are appropriately assigned, the task unit can begin its execution.

Problem variables: To form a task placement plan we first define the decision variables. The variables $z^{h^a}_{i_s}$, $z^I_{i_s}$ and $z^O_{i_s}$ are the binary variables symbolise whether task i_s to be placed on a container h^a, propagated to the nearest node I, or to the cloud O, respectively. For each task i_s, we consider a set of containers $Res^{i_s}(H)$, with $Res^{i_s}(H) \subseteq Res(H)$, which can have and execute i_s. This definition permits to effectively represent the similarity among the task type (i.e., processing, sensing, and actuating tasks) and the allotted resource. Alternatively, we can say that every container in $Res^{i_s}(H)$ are ready to execute the task i_s.

Goal function: According to equation (3), our main goal is to maximise the scheduling of the tasks on to the edge resources (rather than to cloud ones) while fulfilling resource demands of each task unit.

$$\max \sum_{S_n}^{S} \left(P(S_n) \cdot \sum_{i_s}^{S_n} \left(\sum_{h^a}^{Res^{i_s}(H)} z^{h^a}_{i_s} + z^I_{i_s} + z^O_{i_s} \right) \right) \tag{3}$$

It is disadvantageous to place a task to the nearest node or cloud when response time is reaching the deadline, as it incurs network and queuing delays. Therefore the coefficient $P(S_n)$ defined in equation (4) are used to rank the task unit requests for placement. $P(S_n)$ relies on $D^c_{S_n}$, which is calculated as the distance between the task unit deadline E_{S_n} and its queuing time in c. $D^c_{S_n}$ denotes the waiting time of a task unit before it is allocated to a resource. The main objective is to place the task to edge resources which have long waiting time for queuing, with respect to the task unit deadline (without violating the constraints mentioned below).

$$P(S_n) = \frac{1}{E_{S_n} - D^c_{S_n}} \tag{4}$$

First constraints, according to equations (5) and (6) the resource demands such as processor, RAM and storage of task placed on the edge should not be more than the available resource of the devices. Furthermore, the equation also allows to keep free the resources on every container by considering the percentage of system resources $\gamma \in [0, 1]$.

$$\sum_{S_n}^{S} \sum_{i_s}^{S_n} C_{i_s} z^{h^a}_{i_s} \leq \gamma C^{h^a}, \quad \forall h^a \in Res^{i_s}(H), \quad C = \{P, R, M\} \tag{5}$$

$$\sum_{S_n}^{S} \sum_{i_s}^{S_n} C_{i_s} z^I_{i_s} \leq \gamma C^H, \quad C = \{P, R, M\} \tag{6}$$

Second as mentioned in equation (7) it should guarantee that the response time t_{S_n} of each task unit not cross the deadline E_{S_n} of that task unit.

$$t_{S_n} \leq E_{S_n}, \forall S_n \in S \tag{7}$$

According to (8), the task unit response time t_{S_n} is computed as follow:

$$t_{S_n} = K_{S_n} + D_{S_n} \tag{8}$$

where K_{S_n} is the total makespan of task unit and D_{S_n} is the queuing time of task unit.

The makespan K_{S_n} can be calculated as the sum of the time required to execute all the task of task unit and the time required to communicate among the tasks. Therefore,

the task unit makespan interval K_{S_n} is the summation of network link delays of each task placed on individual virtualised resource multiplied by an according decision variable. The term $l(i_s, h^a)$, $l(i_s, O)$,and $l(i_s, I)$ denotes the makespan interval of a task i_s when it is executed on the container h^a, the cloud O, and the nearest node I, respectively. These terms are formalised in equations (10)–(12).

$$K_{S_n} = \sum_{i_s}^{S_n} \sum_{h^a}^{Res^{i_s}(H)} l(i_s, h^a) z_{i_s}^{h^a} + l(i_s, O) z_{i_s}^{O} + l(i_s, I) x_{i_s}^{I} \tag{9}$$

$$l(i_s, h^a) = l^{h_a} + p_{i_s} \tag{10}$$

$$l(i_s, O) = 2l^{O} + p_{i_s} \tag{11}$$

$$l(i_s, I) = 2l^{I} + p_{i_s} \tag{12}$$

The queuing time D_{S_n} of task unit includes the time intervened before each task laced appropriately either fog or cloud computational resources.

Precisely, D_{S_n} sums up the previous and expected queuing time which appears if any task $i_s \in S_n$ is disseminated to the nearest node. The expected queuing time depends on auxiliary variable w_{S_n}. The value of w_{S_n} is one if at least one task in S_n is placed to the nearest node I or zero otherwise. We introduce the task unit queuing time D_{S_n} as follows:

$$D_{S_n} = D_{S_n}^{c} + \tau w_{S_n} + \overline{T}_{DI}^{c} * w_{S_n} \tag{13}$$

If $w_{S_n} = 1$, the second and third term just give a contribution. In particular, τw_{S_n} models that, by placing the task to the nearest node, the task unit placement is delayed to τ time units, whereas $\overline{T}_{DI}^{c} * w_{S_n}$ models the extra queuing time consumed in the nearest node since the task unit execution starts.

Certainly the nearest node has two options either it quickly start the task unit execution depending on its resource requirement or place it to a nearest node by differing additional execution time. It is not viable in practice to find the expected queuing time in the nearest node as $\overline{T}_{DI}^{c}$ requires looking for presumptuous in time. Hence, by relying on supportable information we compute $\overline{T}_{DI}^{c}$. Which is acquired as the moving average of the parameter β on the most recent examined queuing time $T_{tI}^{c-\tau}$ per each task placed to the nearest node:

$$\overline{T}_{DI}^{c} = \beta N_{tI}^{c-\tau} + (1-\beta)\overline{T}_{DI}^{c-\tau} \tag{14}$$

where $\beta \in [0, 1]$ denotes the marking down factor of the moving average, $T_{DI}^{c-\tau}$ is the obtained queuing time of the task sent to I at time c$-\tau$, and $\overline{T}_{DI}^{c-\tau}$ is the estimated average queuing time in I as evaluated in c$-\tau$. Given that $|S_n|$ is the cardinality of S_n, by means of the equations (15) and (16) the variable w_{S_n} is modelled as a logical OR among the placement variables $Z_{i_s}^{I} with\, i_s \in S_n$.

It must be noted that w_{S_n} does not deliberate the situation when the task is placed to the cloud, Alike edge node the task in the cloud is deployed instantly without waiting for the optimisation process.

$$w_{S_n} \leq \sum_{i_s}^{S_n} x_{i_s}^{I}, \forall S_n \in S \tag{15}$$

$$w_{S_n} \geq \frac{\sum_{i_s}^{S_n} x_{i_s}^{I}}{\mid S_n \mid}, \forall S_n \in S \tag{16}$$

Finally, we describe that each task i_s can be placed or propagated on exactly one computational resource, i.e., container h^a, the nearest edge node I, or to the cloud O:

$$\sum_{h^a}^{Res^{i_s}(H)} (z_{i_s}^{h_a}) + z_{i_s}^{I} + z_{i_s}^{O} = 1, \quad \forall i_s \in S, \forall S_n \in S \tag{17}$$

6 Proposed HBF algorithm

In general, the task placement problem is considered as NP-complete (Huang et al., 2009). Thus, we exhibit a heuristic to understand the ETPP. The decision to outline and execute a HBFA to realise the ETPP depended on the prominence of HBFA in solving task scheduling problems in cloud-based situations. The main benefit of HBFA is that they permit to explore a large search space and give a feasible subjective solution in polynomial time.

The fitness value of each bacterium is obtained from the solution set that are evaluated against the fitness function given in, i.e., $J(i, j, k, l)$.

After analysing the Fitness(i, j + 1, k, l), the swim method starts and also to check that it should not walk too long. At each step of $\theta_i(j; k; l)$ we find the Fitness(i, j, k, l). We compare the current fitness function (J(i, j + 1, k, l)) at current step $\theta_i(j+1; k; l)$ with the fitness function $J(i, j, k, l)$of last step $\theta_i(j; k; l)$. If the outcome of Fitness of last is better than the current then the bacterium takes C(i) step in the same direction utmost amount of steps, N_s.

6.1 Bacteria representation

In the algorithm the bacteria represents a solution set for ETPP which gives the idea of assignment of task on to a resource. The size of the bacteria includes all the tasks from task units which are likely to be executed at time c, i.e., $\sum_{S_n}^{S} \mid S_n \mid$. Apart from a task placement plan we can also get the information about the system state from bacteria. The information comprise of response time of task units and required processor, RAM and storage capacities of edge resources.

Fitness function: The fitness function for each bacteria B is evaluated based on the fulfilment of following constraints:

1 φ indicates whether tasks satisfies capacities of processor, RAM and storage resources

2 γ indicates whether tasks are placed on the edge resources

3 μ indicates the 'death' of bacteria (deadline violations).

We consider the above mentioned three types of constraints for computing the fitness value of a bacteria B. First, we check whether the constraints $\forall \beta_p \in \varphi$ are satisfied or not. If satisfied the value will be 0 otherwise 1 as given in (18):

$$\delta_{\beta_p(B)} = \begin{cases} 0, & \text{if } \beta_p(B) \leq 0 \\ 1, & \text{if } \beta_p(B) > 0 \end{cases} \tag{18}$$

Likewise, $\delta_{\beta_c(B)}$ indicates whether the constraints from γ are satisfied or not, if satisfied then its value becomes 0 otherwise 1. Further $D(B)$ denotes the penalty distance of μ constraint, if constraint is satisfied then $\delta_{\beta_d(B)}$= 0 otherwise 1 as given in equation (19).

$$D(B) = \sum_{\beta_d \in \mu} \delta_{\beta_d(B)} \tag{19}$$

The fitness function is evaluated using the equation (20) where $\omega_{\beta_p(B)}$, $\omega_{\beta_c(B)}$ and ω_d are the weight factors for constraints φ, γ and μ respectively.

$$\begin{aligned} F(B) = & \sum_{\beta_p \in \varphi} \omega_{\beta_p}(1 - 2\delta_{\beta_p(B)}) \\ & + \sum_{\beta_c \in \gamma} \omega_{\beta_c}(1 - 2\delta_{\beta_c(B)}) - \omega_d D(B) \end{aligned} \tag{20}$$

The parameters used in algorithm are as follows:

p	dimension of the search space
S	the number of bacteria in the population
$C(i)$	it is random direction taken during tumble
N_c	chemotaxis steps
N_s	swimming length
N_{re}	reproduction steps
N_{ed}	elimination and dispersal events
P_{ed}	elimination and dispersal probability.

6.2 Initial position generation

The bacteria holds the solution of the problem domain that represents the solution for our problem which is the novelty associated with the proposed algorithm.

6.3 Hybrid chemotaxis

Chemotaxis is a process where the E.coli bacteria is carried in flagella, by tumbling (movement in the random direction) and swimming (motion in the same direction). We add the properties of genetic algorithm to make the process of chemotaxis as hybrid chemotaxis.

After analysing the $J(i, j+1, k, l)$, the swim method starts and also to check that it should not walk too long. At each step of $\theta_i(j; k; l)$ we find the $J(i, j, k, l)$. We compare the current fitness function $(J(i, j+1, k, l))$ at current step $\theta_i(j+1; k; l)$ with the fitness function $J(i, j, k, l)$ of last step $\theta_i(j; k; l)$. If the outcome of fitness of last is better than the current then the bacterium takes $C(i)$ step in the same direction utmost amount of steps, N_s.

We initiate a loop to find the fitness function for each bacteria in the bacteria set. Extract the first bacteria from the solution set. Find the fitness function given in equation (20) as $J(i, j, k, l)$ that corresponds initial position vector $\theta(i, j, k, l)$, where $\theta(i, j, k, l)$ denotes the current position of i^{th} bacterium at a value of η and λ, in j^{th} chemotaxis, k^{th} reproduction and l^{th} elimination and dispersal step. Assign $J_{last}(i, j, k, l) = J(i, j, k, l)$. Next, calculate $\theta(i, j+1, k, l)$ for $(j+1)^{\text{th}}$ chemotaxis step as follows:

$$\theta(i, j+1, k, l) = \theta(i, j, k, l) + C(i)\phi_j \tag{21}$$

$\theta(i, j+1, k, l)$ gives rise to the fitness function at $(j+1)^{\text{th}}$ chemotaxis step as $J(i, j+1, k, l)$. $C(i)$ is the size of the step taken in the random direction specified by the tumble. Generate a random direction ϕ_j, which the value is between interval $[-1, 1]$. It is the direction angle of the j^{th} Chemotaxis step.

If the $J(i, j+1, k, l)$ at $\theta(i, j+1, k, l)$ is better than the $J(i, j, k, l)$ at $\theta(i, j, k, l)$ then the bacterium takes another step of size $C(i)$, otherwise it tumbles in a random direction of size ϕ_j. This process continue until the boundary condition satisfy, i.e., the number of steps is greater than N_c.

Let $J_{last}(i, j, k, l) = J(i, j, k, l)$. During the chemotactic process the bacterium cell alternates between swims and tumbles. The swim, move and tumble process is carried out as follows.

6.3.1 Tumble

Generate a random direction ϕ_j, which the value is between interval $[-1, 1]$

6.3.2 Swim

Let $m = 0$ (counter for swim length).

The swim process continues until the $(m < N_s)$ condition is satisfied. In this place we calculate fitness value.

If $J(i,j+1,k,l) < J_{(last)}(i,j,k,l)$ then $J_{(last)}(i,j,k,l) = J(i,j+1,k,l)$. The swimming and tumbling of the bacteria depends on the value of $J(i,j+1,k,l)$ and $J(i,j,k,l)$, i.e., if $J(i,j+1,k,l)$ is greater than $J(i,j,k,l)$, the bacteria swim in step size of $C(i)$ else tumble in random direction of ϕ_j steps.

6.4 Hybrid-reproduction

To improve the BF and design HBF we incorporate the selection and reproduction of GA and implemented in BF when each bacterium in the solution set completes chemotactic steps.

In this paper $J^i_{(health)}$ denotes the accumulated fitness function value during the bacterium lifetime of bacteria i. Sort bacteria in order of ascending cost $J^i_{(health)}$.

6.4.1 Selection

Select the bacterial population and sort them in increasing order on the basis of minimum J_{health} (which is the health of the i^{th} bacteria). The J_{health} is calculated as follows: $J^i_{(health)} = \sum_{j=1}^{N_c+1} J(i,j,k,l)$. Each $J^i_{(health)}$ is the fitness value of bacteria i.

6.4.2 Crossover

The crossover is performed with the η value. The significance of this crossover process is. it gives new task execution order. It is done randomly in the bacterial population with the least fit in nature.

6.4.3 Mutation

The mutation is performed by taking least cost J_{health} value as done in previous chemotactic steps. The outcome of the reproduction step gives rise bacterias, i.e., half of the total population of bacteria. The discarded bacteria have higher $J_{(health)}$ values compared to the one selected for future production. The newly selected bacterias further split to fill the vacant places of discarded bacteria in bacteria population. This process again moves to chemotactic step as described above. The process continues until $k < Nre$.

6.5 Elimination-dispersal

The bacteria are dispersed to random locations in the optimisation after elimination. The elimination is done with the probability P_{ed} to keep the bacteria population constant. This process again moves to reproduction step as described above. The process continues until $k < N_{ed}$.

6.6 Termination

The termination process is similar to BF. The pseudo code of our hybrid bacteria foraging algorithm is given in Algorithm 1.

Algorithm 1 HBFA-based task scheduling algorithm

Input : Number of tasks and number of available cloud resources.
Output: Mapping of each tasks to resources

begin
  Initialise parameters P, N, N_c, N_s, N_{re}, N_{ed}, P_{ed}, $C(i)$ (i = 1,2, . . . , S) PM list,task unit list, η and λ for each bacterial.;
  `/* Elimination and dispersal loop starts */`
  while *($l+1 < N_{ed}$)* **do**
    `/* Reproduction loop starts */`
    while *($k+1 < N_{re}$)* **do**
      `/* Chemotaxis loop starts */`
      while *($k+1 < N_c$)* **do**
        for $i \leftarrow 1$ **to** S **do**
          `/* Calculate fitness functions` $J(i,j,k,l)$`, */`
          $J(i,j,k,l) = J(i,j,k,l) + J_{cc,t}(\lambda^i(i,j,k,l), P(i,j,k,l))$;
          Goal function F=$J(i,j,k,l)$;
          $J_{(last)}(i,j,k,l)$=F;
          ; `/* Hybrid chemotaxis starts */`
          Tumble: Generate a random direction $\phi_(i)$, which the value is between interval [–1, 1];
          Swim: let $m = 0$ `/* counter for swim length */`
          while *($m < N_s$)* **do**
            $m = m + 1$;
            if $J(i,j+1,k,l) < J_{(last)}(i,j,k,l)$ **then**
              $J_{(last)}(i,j,k,l) = J(i,j+1,k,l)$;
              Mutate: Mutation ;
              Calculate $J(i,j+1,k,l)$ as step 10;
            else
              $m = N_s$;
        ; `/* Hybrid reproduction starts */`
        Reproduction:;
        $J^i_{(health)} = \sum_{j=1}^{N_c+1} J(i,j,k,l)$;
        Mutate: Mutation;
        Crossover: Crossover;
      Elimination-dispersal:;
      with probability P_{ed}, eliminate and disperse each bacterium;
    Terminate:;
    End the Program and output the result.
end

7 Simulation results and discussion

Different parameters considered for simulation is presented in Tables 2 and 3 for BF and GA respectively. For HBFA we have considered both the tables. The proposed work is evaluated to show ETPP performance. We use fog

simulation tool kit (Gupta et al., 2017) to simulate edge network and cloud resource.

Table 2 Parameter for BF

Initialisation	No. of bacterial dimensions:	10
	No. of bacterial in colony:	50
Chemotaxis	No. of tumbles:	10
	No. of Swims:	8
Reproduction	No. of reproduction	10
	No. of reproductive bacteria:	25
Elimination and dispersal	No. of elimination dispersal events:	6
	Probability:	0.2
Termination	Maximum no. of elimination and dispersal events:	6

Table 3 Parameters for GA

Initialisation	No. of variables:		10
	No. of individuals:		200
Selection	Type		Tournament
	Generation gap		0.98
Reproduction	Crossover	Type	Uniform
		Probability (p_c)	0.8
	Mutation	Type	Random
		Probability (p_m)	0.2
Termination	Maximum generations		200

We need to modify the ifogsim as it does not support all the properties of edge network and cloud.

The changes that we have made to ifogsim are an application class modified to address the task unit deadline, queuing and response time of task unit.

We have modified the edge devices class to the model edge control node. The changes are made to support:

1 finding the nearest node

2 utilisation of edge resources.

In the proposed work, we have used four algorithms to compose: first fit algorithm which is based on the greedy heuristic, bacteria foraging algorithm (BFA), the genetic algorithm (GA) and hybrid bacteria foraging algorithm (HBFA).

Using the first fit greedy-based algorithm we schedule a task on to an edge node/server/nearest node/cloud based on the availability criteria.

The task's makespan is set based on the output of iFogsim that has been obtained from pre-experiment Tables 6 and 7 gives the detail of experimental setup. We have done the numerical analysis and presented in the table.

Each task requires some amount of resource such as processor, RAM, storage, etc. to complete its execution. The detail of resource demand of task is given in Table 6, while the capacity of edge resources is given in Table 5. The details of the task units deadline and queuing time are given in Table 7, each task unit composed of three tasks.

Table 4 Delays

Variable	*Time*
Network delay for cloud l^O	1 sec
Network delay for neighbour node l^I	0.5 sec
Network delay for container l^{h^a}	0 sec
Expected queuing time $\overline{T}^c_{DI}$	180 sec
Sampled queuing time $\overline{T}^{c-\tau}_{DI}$	120 sec

Table 5 Resource capacity

Resource	*Processor*	*RAM*	*Storage*
Edge device	1000 MIPS	512 MB	8 GB
Container	250 MIPS	256 MB	4 GB

Table 6 Task unit resource demands

Task	P_{i_s} *(MIPS)*	R_{i_s} *(MB)*	M_{i_s} *(MB)*	K_{i_s} *(s)*
Sense	50	20	10	0.9
Process	100	30	20	0.25
Actuate	50	10	20	0.5

Table 7 Task unit details

Task unit	E_{s_n} *(s)*	D_{s_n} *(s)*
S1	100	60
S2	250	0
S3	300	60
S4	350	60
S5	120	0

Table 8 Numerical analysis of response time

Task units		*Allocation*	*Makespan*	*Response time*
S_1	Sense	h_a	1.65	61.65
	Process	h_a		
	Actuate	h_a		
S_2	Sense	o	2.65	2.65
	Process	h_a		
	Actuate	h_a		
S_3	Sense	o	3.65	63.65
	Process	h_a		
	Actuate	o		
S_4	Sense	h_a	3.15	63.15
	Process	I		
	Actuate	o		
S_5	Sense	o	4.65	4.65
	Process	o		
	Actuate	o		

We have considered that the edge node consists of ten containers connected to an edge control node. The network link delays and queuing time are given in Table 4. We allocate the task to edge resources by considering the delays and makespan. We numerically analyse the response time of each task units and usage of different resources as depicted in Tables 8 and 9.

Table 9 Numerical analysis of utilisation

Resources	*Utilisation*		
	No. of task placed	*Total task*	*%*
Container	7	15	47
Neighbour node	1		6
Cloud	7		47

The aim of this assessment is to compare the performance of our proposed HBF algorithm with respect to other three approaches, i.e., first fit, BF, and GA in terms of response time of task units, the execution time of the task, placement time of the task, utilisation of resources, etc. The summary of performance evaluation is presented in Table 10.

Further, we employed a statistical method namely standard deviation to analyse the statistical significance of the results.

The results sampled in Table 11 provide information about the proposed algorithm performance under different parameters viz response time, utilisation, cost, etc. It can be observed that the standard deviation is in the range of 0.2 to 2.5 which confirms the stability of HBFA.

7.1 Deadline violations and task delays

In Table 10, the results show that in the first fit approach the deadline is violated for the task unit S_1 and S_5 respectively. In other approaches, i.e., BF, GA, and HBFA, there is no deadline violation, but they lead to different results for edge resource utilisation and task unit response time.

When we observe Figure 7(a) there are many peaks found in the graph as the available resource in edge node is less powerful compared to the cloud. In other cases, i.e., GA and HBFA the deadlines are not violated but the extra time incurred for single task execution on average is less in HBFA in comparison to GA. The response time of task unit and delay to execute a single task is shown in Figures 5(b) and 7(a). When we observe Figure 5(b) the GA does not overcome the deadline, and the task is processed soon after submission of task unit as GA distribute the task among the resources in such a way that it has unlimited suitable resources available in the network. However, when we observe the utilisation of GA, the cost of execution is high compared to HBFA. This is due to the increase in network delay.

The observation continues for all task units presented in the simulation we prioritise to discuss task unit for S_3 as it has the highest response time. The different parameters of the task unit are presented in Table 7 when the deadline is below 100 sec the task unit S_3 is placed on current edge nodes. When the deadline is in between 100 and 350 sec, the deadline of task unit S_3 approaches to S_4, and both of them are placed in the cloud and rest of the task reduces the network delay and giving rise to better response time. When the deadline is above 300, the task unit is placed within the edge network, and few are placed to nearest edge network because of which τ and $\overline{T}^{c}_{DI}$ contributes to the task unit response time. When we observe Figure 9(a) by keeping the deadline constant for some time when we increase the CPU capacity of edge control node the response time of task decreases.

The first fit and BF the decrease in response time is slower, compared to GA and HBFA the reason is that the GA and HBFA distribute the task evenly with an optimised task placement solution. When we increase the deadline from 50 to 70 and 70 to 100 the response time also vary accordingly as shown in Figure 9(b) and 9(c).

When we observe Figure 3(a) with an increase in the number of tasks the goal function increases. In case of first-fit, the goal function value is less as the distribution of task is not fair because of which some of the tasks are dropped that affects the goal function while in case of BF, GA and HBFA the task are appropriately distributed and helps in maximising the goal function. In Figure 3(a), 3(c), 3(b) we have demonstrated the variation of goal function with increase in the number of tasks for a constant deadline of 50, 70 and 100 sec.

7.2 Utilisation of the edge resource

From Figure 6(b), we observed that the HBFA algorithm utilises the cloud resources less compared to other approaches as the involvement of cloud leads to increase in cost and response time which violate the objective function. Another observation can be done that proposed HBFA utilises container of the current edge node and nearest node efficiently compare to other approaches. Therefore different approaches may lead to less weight factor in response to the fitness function.

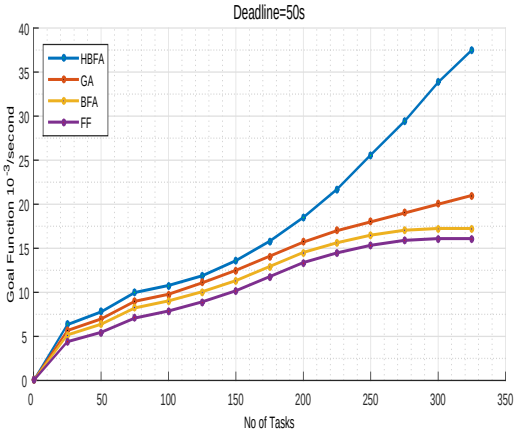
Table 10 Performance comparison

Approach	t_{s_n} *(s)*					$E_{s_n} - t_{s_n}$ *(s)*					*Utilisation*			*Cost (\$)*
	S_1	S_2	S_3	S_4	S_5	S_1	S_2	S_3	S_4	S_5	h_a	I	o	
First-fit	72.4	57.6	65.36	65.61	125.8	–1.4	192.4	234.64	284.39	–5.8	27	20	53	4.81
GA	66.45	49.3	64.21	64.52	81.2	30.55	200.7	235.79	285.48	38.8	40	35	25	0.52
BF	72.25	51.1	64.94	65.2	93.5	27.75	198.9	235.06	284.8	26.5	32	20	48	2.5
Proposed HBFA	59.55	45.25	62.95	62.45	72.65	40.45	204.75	237.05	287.55	47.35	50	46	4	0.09

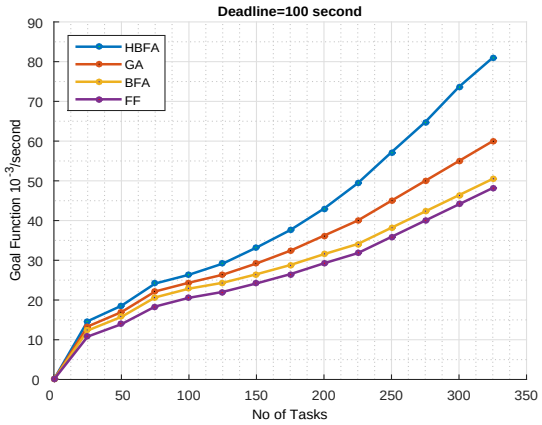
Table 11 Statistical analysis

Metrics	t_{s_n} *(s)*					E_{s_n} - t_{s_n} *(s)*					*Utilisation*			*Cost (\$)*
	s_1	s_2	s_3	s_4	s_5	s_1	s_2	s_3	s_4	s_5	h_a	I	o	
Proposed HBF	59.55	45.25	62.95	62.45	72.65	40.45	204.75	237.05	287.55	47.35	50	46	4	0.09
σ	1.41	0.12	1.35	0.32	0.21	1.48	1.38	0.29	0.25	0.27	3.92	4.38	1.6	0.2
BF	72.25	51.1	64.94	65.2	93.5	27.75	198.9	235.06	284.8	26.5	32	20	48	2.5
σ	2.41	1.12	2.35	1.32	1.21	2.48	2.38	1.6	1.25	2.27	5.92	6.38	2.6	1.2
GA	69.45	49.3	64.21	64.52	81.2	30.55	200.7	235.79	285.48	38.8	50	46	4	0.52
σ	1.69	0.15	2.35	0.75	0.67	1.52	1.48	0.65	0.45	0.37	4.92	5.38	1.9	0.9

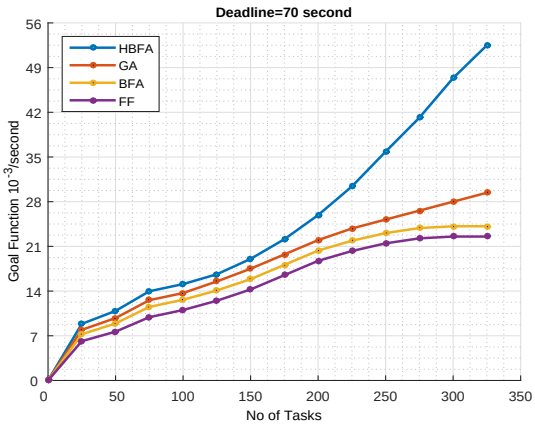
Figure 3 Simulation results, (a) impact of task on goal function with deadline = 50 sec (b) impact of task on goal function with deadline = 100 sec (c) Impact of task on goal function with deadline = 70 sec (see online version for colours)



(a)



(b)



(c)

In the case of GA, the utilisation of cloud is more when compared with HBFA, but the utilisation of cloud is less when compared with the other two approaches. In Figure 4, we present how the utilisation of current edge network getting affected in response to τ and $\overline{T}^{c}_{DI}$. We performed different experiments to calculate the goal function and observe the placement of task after taking different values for τ and $\overline{T}^{c}_{DI}$. Also we observed that the lowest the value of $\overline{T}^{c}_{DI}$ the nearest edge network execute the task faster and allows the edge control node to place most of the task to the nearest node when $\overline{T}^{c}_{DI}$ approaches to certain value it starts conflicting with deadline of task unit resulting in decrease in goal function and places most of the task on to the cloud. When $\overline{T}^{c}_{DI}$ increases significantly most of the task is placed to the cloud as current and nearest edge network are not sufficient to execute the task the above observation is also applicable to τ. When we vary the capacity of edge resources the number of task placement in edge network increases which in turn increases the goal function. In Figure 7(b), we present the variation of goal function by varying task unit deadline we conduct many experiments where the deadline for task unit ranges from 100 to 500 sec.

Figure 4 Impact of task on fitness function (see online version for colours)

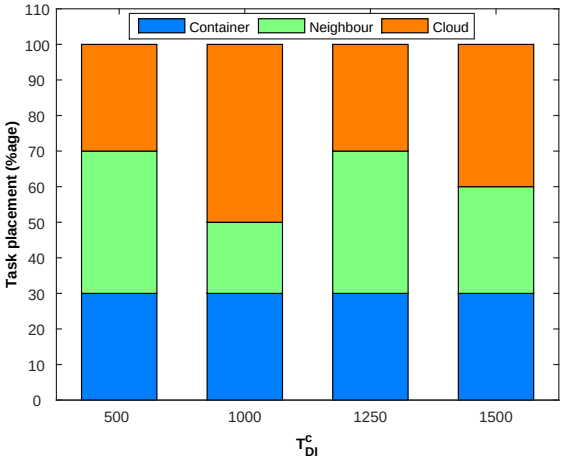
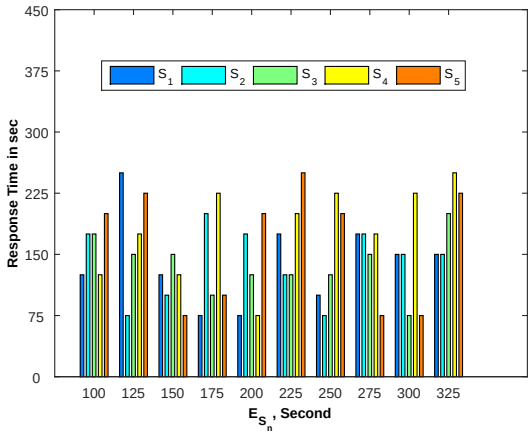
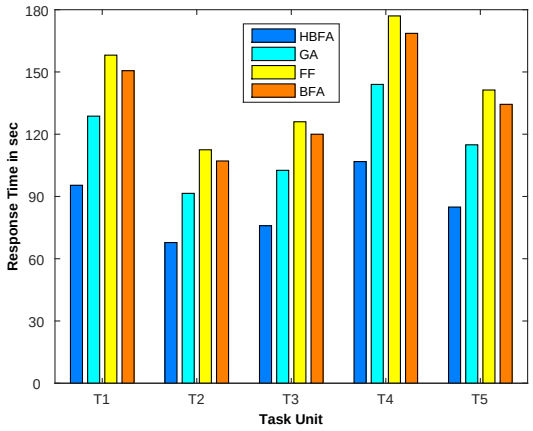


Figure 5 Simulation results 1, (a) impact of task unit deadlines on response times (b) response times of task units (see online version for colours)

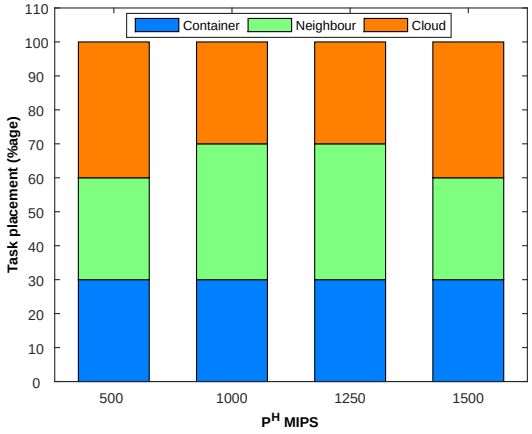


(a)

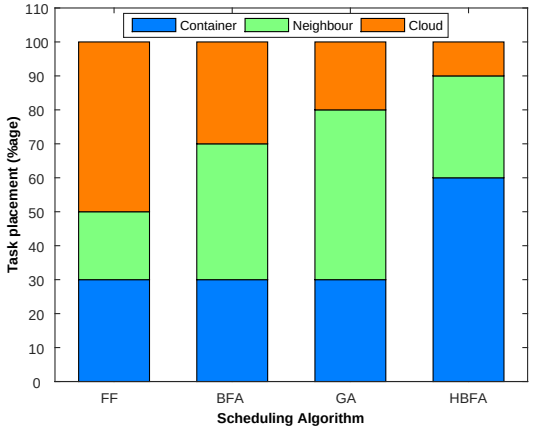


(b)

Figure 6 Simulation results 2, (a) impact of resources of edge device on task placement (b) utilisation of resources (see online version for colours)

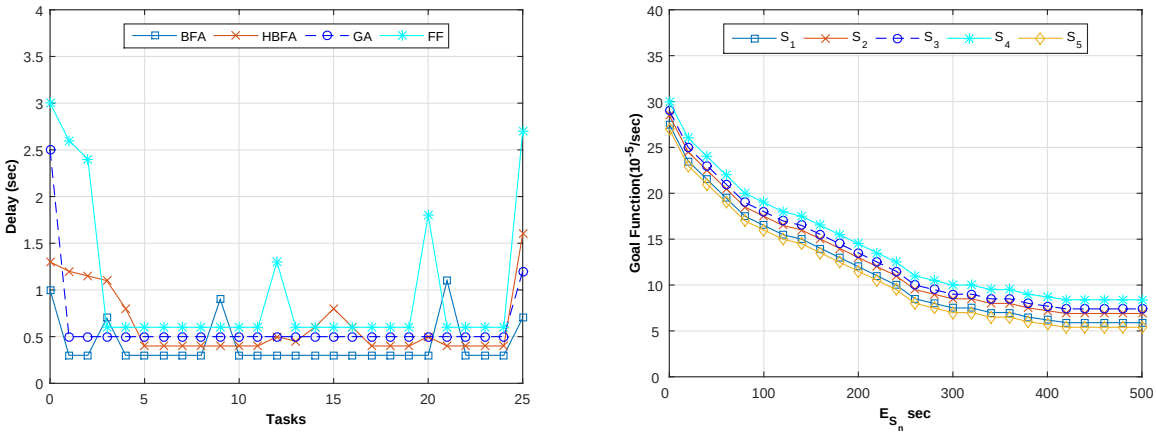


(a)



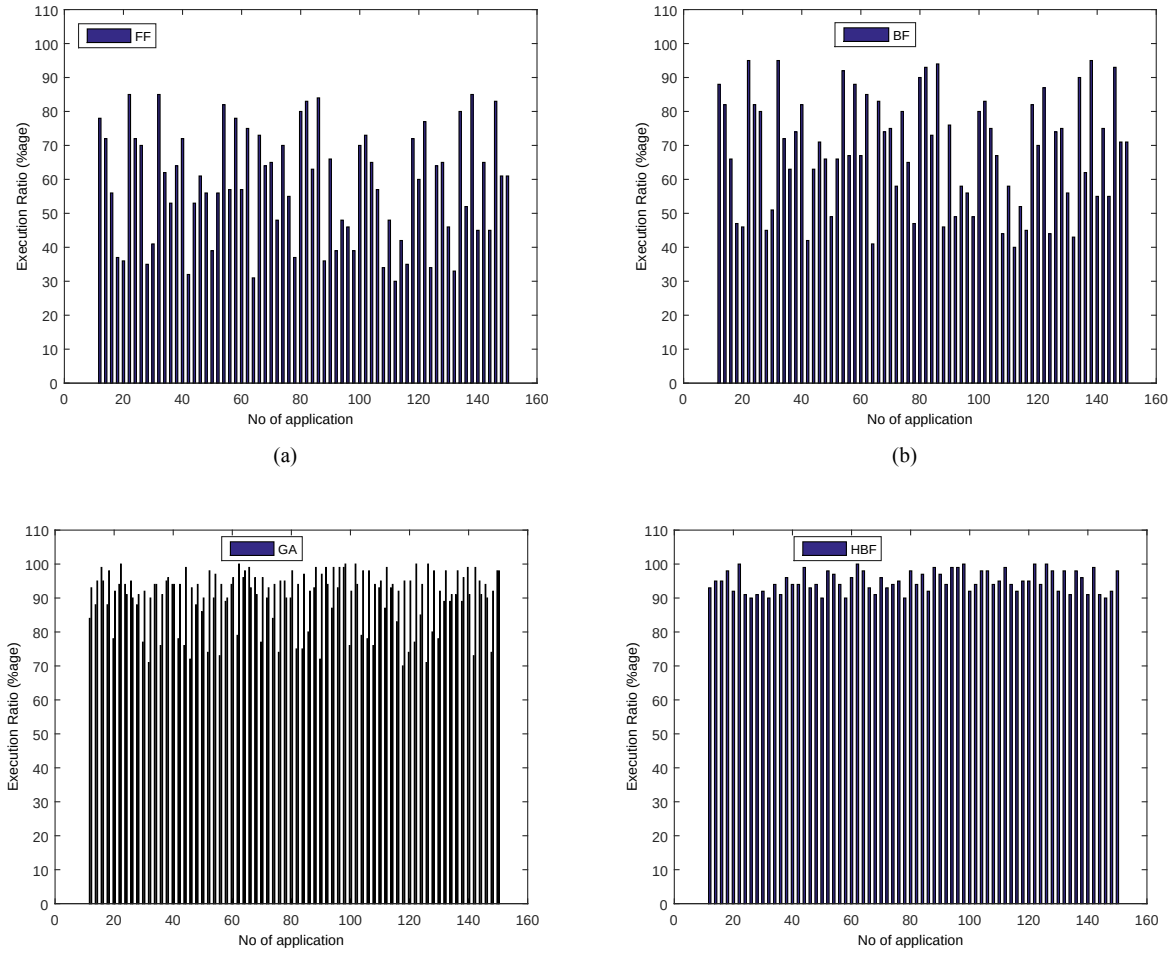
(b)

Figure 7 Simulation results 3, (a) task execution delays (b) impact of task unit deadlines on the goal function (see online version for colours)



(a) (b)

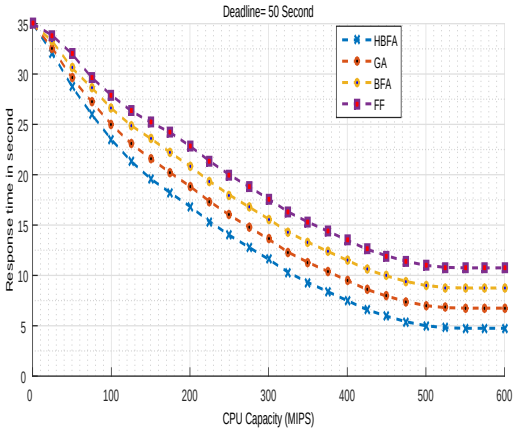
Figure 8 Simulation results, (a) analysis of task execution ratio of first-fit (b) analysis of task execution ratio of BF (c) analysis of task execution ratio of GA (d) analysis of task execution ratio of HBFA (see online version for colours)



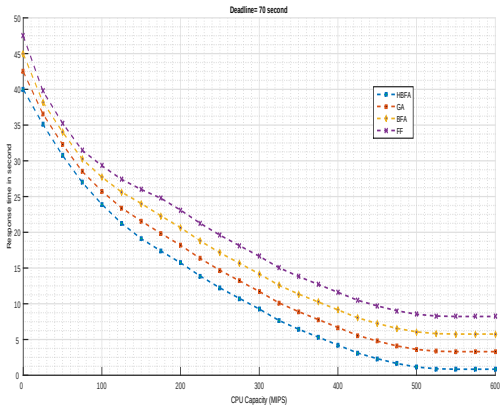
(a) (b)

(c) (d)

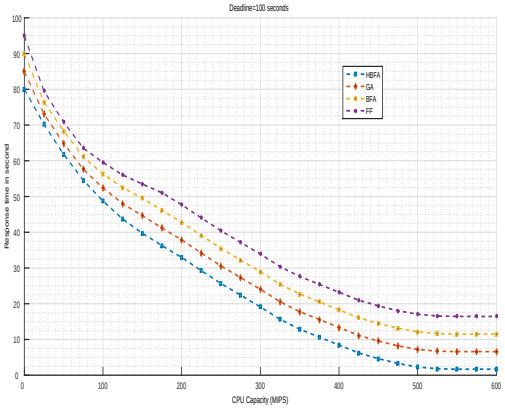
Figure 9 Simulation results, (a) impact of CPU capacity on response time with deadline = 50 sec (b) impact of CPU capacity on response time with deadline = 70 sec (c) Impact of CPU capacity on response time with deadline = 100 sec (see online version for colours)



(a)



(b)



(c)

The edge control node prefers to place the task either the nearest node or to the cloud that affects the goal function coefficient $\frac{1}{E_{S_n} - t_{s_n}}$. This is the only reason the task is preferred to place less in the current edge node more in nearest node or cloud. When we increase the deadline to a maximum that in turn affect the value $\overline{T}^{c}_{DI}$, i.e., when the deadline starts from 0 to 500 sec. The task is placed either in the nearest node or in the cloud which in turn increases the goal function slightly, but the coefficient of goal function reduces. In Figures 8(a), 8(b), 8(c) and 8(d) we compare the number of task units arrived from the user for execution in to the edge control node with the execution ratio of the task. The execution ratio defines the ratio of number of task units successfully executed to the number of task units arrived at an edge control node. In first-fit as shown in Figure 8(b) the ratio is very less because the first fit algorithm prefers to allocate the task unit to the local edge control node and these local edge control nodes do not have sufficient resources to host the application where in case of BF, HBFA, and GA the task unit distribution is done by considering nearest node and cloud resources because of which task execution ratio is significantly high in proposed HBFA in comparison to BF and GA as depicted in Figures 8(b), 8(c), and 8(d).

7.3 *The cost of execution*

The cost of task execution is depicted in Table 10. The cost of task execution is computed for the usage of the cloud as a product of the time in seconds of using cloud resources and the cost per processing in the cloud divided by the number of seconds in 1 billing time unit (BTU).

7.4 *Scalability*

To study the scalable performance of the proposed algorithm evaluated and compared with other standard algorithm BF, GA and first fit. The results have been depicted in the figures of the simulation result, where we analysed the variation in delay utilisation and cost. From the result, we observed that the proposed algorithm outperformed and remain consistent and optimised as compared to other strategies.

8 Conclusions and future work

Edge computing targets to utilise existing computational, networking and storage resources for the portrayal of IoT tasks in the native edge network. As of now, the adoption of edge computing is in its infancy, and there is an absence of the theoretical and practical basis for edge task placement and resource provisioning. The proposed work is motivated by the cloud manufacturing, and we have given the overall architecture of edge computing and proposed a scheduling strategy for tasks in the cloud edge environment. The proposed strategy is simulated and evaluated with state of the art algorithms, i.e., the first-fit, GA, BF. The comparison results show that the proposed method produces better scheduling results compared to other strategies. The proposed strategy produces solutions which on average experience a lower deployment delay by exploiting more cloud resources. The future work will be based on multiobjective optimisation, i.e., minimising energy consumption, resource discovery and resource estimation (Toczé and Nadjm-Tehrani, 2018) as the edge device runs with power constraint. Also, we aim to present a mathematical model for resource provisioning.

References

Bonomi, F., Milito, R., Natarajan, P. and Zhu, J. (2014) 'Fog computing: a platform for internet of things and analytics', *Big Data and Internet of Things: A Roadmap for Smart Environments*, pp.169–186, Springer.

Dastjerdi, A.V., Gupta, H., Calheiros, R.N., Ghosh, S.K. and Buyya, R. (2016) 'Fog computing: principles, architectures, and applications', *Internet of Things*, pp.61–75, Elsevier.

Digalwar, M., Gahukar, P., Raveendran, B.K. and Mohan, S. (2017) 'Energy efficient real-time scheduling algorithm for mixed task set on multi-core processors', *International Journal of Embedded Systems*, Vol. 9, No. 6, pp.523–534.

Elmroth, E., Leitner, P., Schulte, S. and Venugopal, S. (2017) 'Connecting fog and cloud computing', *IEEE Cloud Computing*, Vol. 4, No. 2, pp.22–25.

Georgakopoulos, D., Jayaraman, P.P., Fazia, M., Villari, M. and Ranjan, R. (2016) 'Internet of things and edge cloud computing roadmap for manufacturing', *IEEE Cloud Computing*, Vol. 3, No. 4, pp.66–73.

Giang, N.K. Blackstock, M., Lea, R. and Leung, V.C. (2015) 'Developing IoT applications in the fog: a distributed dataflow approach', *2015 5th International Conference on the Internet of Things (IOT)*, pp.155–162, IEEE.

Gupta, H., Dastjerdi, A.V., Ghosh, S.K. and 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: Practice and Experience*, Vol. 47, No. 9, pp.1275–1296.

Huang, T., Li, T., Dong, Q., Zhao, K., Ma, W. and Yang, Y. (2017) 'Communication-aware task scheduling algorithm for heterogeneous computing', *International Journal of High Performance Computing and Networking*, Vol. 10, Nos. 4–5, pp.298–309.

Huang, X., Ganapathy, S. and Wolf, T. (2009) 'Evaluating algorithms for composable service placement in computer networks', *IEEE International Conference on Communications 2009, ICC'09*, pp.1–6, IEEE.

Jaiswal, K., Sobhanayak, S., Mohanta, B.K. and Jena, D. (2017) 'IoT-cloud based framework for patient's data collection in smart healthcare system using Raspberry-Pi', *2017 International Conference on Electrical and Computing Technologies and Applications (ICECTA)*, November, pp.1–4.

Johnston, S.J. and Cox, S.J. (2017) *The Raspberry Pi: A Technology Disrupter, and the Enabler of Dreams*.

Kubler, S., Holmström, J., Främling, K. and Turkama, P. (2016) 'Technological theory of cloud manufacturing', *Service Orientation in Holonic and Multi-Agent Manufacturing*, pp.267–276, Springer.

Liu, Y., Li, K., Tang, Z. and Keqin, L. (2015) 'Energy aware list-based scheduling for parallel applications in cloud', *Internationl Journal of Embedded Systems*.

Schulte, S., Hoenisch, P., Hochreiner, C., Dustdar, S., Klusch, M. and Schuller, D. (2014) 'Towards process support for cloud manufacturing', *2014 IEEE 18th International Enterprise Distributed Object Computing Conference (EDOC)*, pp.142–149, IEEE.

Singh, R. (2018) 'Hybrid genetic, variable neighbourhood search and particle swarm optimisation-based job scheduling for cloud computing', *International Journal of Computational Science and Engineering*, Vol. 17, No. 2, pp.184–191.

Skarlat, O., Borkowski, M. and Schulte, S. (2016) 'Towards a methodology and instrumentation toolset for cloud manufacturing', *2016 1st International Workshop on Cyber-Physical Production Systems (CPPS)*, pp.1–4, IEEE.

Sun, Y., Teng, L., Yin, S. and Li, H. (2019) 'A new wolf colony search algorithm based on search strategy for solving travelling salesman problem', *International Journal of Computational Science and Engineering*, Vol. 18, No. 1, pp.1–11.

Tian, S. and Yang, H. (2019) 'Load-balanced overlay recovery scheme with delay minimisation', *International Journal of High Performance Computing and Networking*, Vol. 13, No. 1, pp.119–128.

Toczé, K. and Nadjm-Tehrani, S. (2018) *A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing*, arXiv preprint arXiv:1801.05610.

Wang, W., Dong, M., Ota, K., Wu, J., Li, J. and Li, G. (2019) 'Cdlb: a cross-domain load balancing mechanism for software defined networks in cloud data centre', *International Journal of Computational Science and Engineering*, Vol. 18, No. 1, pp.44–53.

Wu, D., Greer, M.J., Rosen, D.W. and Schaefer, D. (2013) 'Cloud manufacturing: strategic vision and state-of-the-art', *Journal of Manufacturing Systems*, Vol. 32, No. 4, pp.564–579.

Xiong, Y., Chen, Y., Jiang, K. and Tang, Y. (2017) 'An energy-aware task consolidation algorithm for cloud computing data centre', *International Journal of High Performance Computing and Networking*, Vol. 10, Nos. 4–5, pp.352–358.

Xu, X. (2012) 'From cloud computing to cloud manufacturing', *Robotics and Computer-Integrated Manufacturing*, Vol. 28, No. 1, pp.75–86.

Zhang, H. and Wang, K. (2018) 'Research of dynamic load balancing based on stimulated annealing algorithm', *International Journal of Embedded Systems*, Vol. 10, No. 3, pp.188–195.

Zhang, Q., Cheng, L. and Boutaba, R. (2010) 'Cloud computing: state-of-the-art and research challenges', *Journal of Internet Services and Applications*, Vol. 1, No. 1, pp.7–18.