The Journal of Supercomputing
https://doi.org/10.1007/s11227-021-04135-2

Containerized deployment of micro-services in fog devices: a reinforcement learning-based approach

Shubha Brata Nath[1] · Subhrendu Chattopadhyay[2] · Raja Karmakar[3] · Sourav Kanti Addya[4] · Sandip Chakraborty[1] · Soumya K Ghosh[1]

Accepted: 8 October 2021

Abstract

The real power of fog computing comes when deployed under a smart environment, where the raw data sensed by the Internet of Things (IoT) devices should not cross the data boundary to preserve the privacy of the environment, yet a fast computation and the processing of the data is required. Devices like home network gateway, WiFi access points or core network switches can work as a fog device in such scenarios as its computing resources can be leveraged by the applications for data processing. However, these devices have their primary workload (like packet forwarding in a router/switch) that is time-varying and often generates spikes in the resource demand when bandwidth-hungry end-user applications, are started. In this paper, we propose *pick–test–choose*, a dynamic micro-service deployment and execution model that considers such time-varying primary workloads and workload spikes in the fog nodes. The proposed mechanism uses a reinforcement learning mechanism, *Bayesian optimization*, to decide the target fog node for an application micro-service based on its prior observation of the system's states. We implement PTC in a testbed setup and evaluate its performance. We observe that PTC performs better than four other baseline models for micro-service offloading in a fog computing framework. In the experiment with an optical character recognition service, the proposed PTC gives average response time in the range of 9.71 sec–50 sec, which is better than Foglets (24.21 sec–80.35 sec), first-fit (16.74 sec–88 sec), best-fit (11.48 sec–57.39 sec) and mobility-based method (12 sec–53 sec). A further scalability study with an emulated setup over Amazon EC2 further confirms the superiority of PTC over other baselines.

Keywords Fog computing · Container deployment · Bayesian optimization · Internet of things · Reinforcement learning

✉ Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

Extended author information available on the last page of the article

Published online: 27 October 2021

1 Introduction

The concept of fog computing [9, 18] is gaining popularity in recent times due to the availability of computationally powerful network devices, like routers, WiFi access points, core switches, etc. Various recent works have focused on developing innovative applications around it, ranging from smart cities applications [12], big data processing [25], pervasive health care [1, 19], etc. In a typical fog computing architecture, the intermediate network devices' excess computing resources are used for end-user application processing purposes. Thus, it provides low network latency along with added privacy to the data. However, the response time becomes higher when the end devices like the Internet of Things (IoT) devices do not have sufficient computing resources, and hence, the computation needs to be offloaded to a secondary device. Although a large number of research works [2, 7, 10, 17, 24, 27, 31, 33, 34] have been devoted to design application offloading mechanisms for fog computing, they primarily consider the workload placement as a one-time process and fail to capture the dynamicity of the system.

In the fog computing paradigm, the application workloads are divided into micro-services that can be executed in parallel [4]. These micro-services [15, 29, 30] are placed on the fog nodes (the routers, access points and core network switches) through some sandboxing mechanism, like the virtual machines (VMs) or the containers. Such a placement mechanism needs to consider two different aspects—(1) there should be excess computing resources available at the fog nodes, and (2) the execution of the micro-service should not interfere with the primary workload of the fog nodes. Although the existing approaches, as mentioned above, consider these two factors, they fail to properly characterize the nature of the fog nodes' primary workload. Such primary workloads are typically dynamic. Consider the following example. In a smart home environment, let the IoT devices and the corresponding applications use the home gateway as a fog node. Thus, the IoT data processing tasks are offloaded on the network gateway. Now, multiple smart home appliances are connected to that gateway, and the gateway needs to route the data packets to respective subnets, which is its primary workload. Now, this gateway's primary workload observes a spike whenever high-bandwidth applications, such as a smart television, get started. During that time, the gateway might not have sufficient computation resources to execute the fog micro-services. If we still force the micro-services to get completed on that fog node, the response time will possibly get increased, resulting in a negative performance impact.

The primary advantage of fog computing comes from the fact that it provides a quicker response than the cloud as the computation happens near the device. However, unless we properly place and schedule the micro-services over the fog nodes by considering their primary workloads, the advantages might get nullified and it might result in a very high response time. The primary requirements for developing such an execution architecture are as follows. (1) The placement algorithm should not be a one-time algorithm; it should continuously monitor the primary workload and excess resources in the fog nodes and dynamically update the target node where the micro-service can execute. (2) Depending on the dynamic

execution decision, micro-services should be able to seamlessly migrate from one fog node to another to complete its execution. (3) The migration should be fast, having negligible overhead. (4) The decision of placement and migration should be very fast so that it does not contribute significantly to the overall response time. The existing mechanisms fail to capture all of these four aspects together.

We propose *pick–test–choose* (PTC) framework which can be considered as a containerized micro-service placement mechanism for the fog nodes considering their primary workloads and their impact on the micro-service execution. PTC continuously monitors the fog nodes' excess resources and the required computation environment for the micro-service workloads to complete their executions. Accordingly, it models the problem as a constrained optimization such that the response time always remains within a predefined boundary. As solving such a constrained optimization is computationally expensive, we use a reinforcement learning technology to dynamically map the micro-service execution to the fog nodes' computation platforms. To satisfy the requirement of having a quick decision, we use *Bayesian optimization* [26] (BO) that can dynamically and quickly decide the target solution. It also considers the measurement noises that may come while calculating the resource availability (of the fog nodes) and the resource requirements (of the micro-services). We have implemented PTC over a prototype testbed for thorough performance evaluation and comparison with baselines; along with that, the performance has also been evaluated in Amazon Elastic Compute Cloud (Amazon EC2) for scalability analysis. The experimental results show that PTC can minimize the response time while effectively using fog nodes' excess resources. We further analyze PTC's performance in the context of a natural language processing (NLP) application as a use case, which has been evaluated over the testbed. We observe that the proposed framework can significantly reduce the response time of the NLP application's micro-service executions.

An initial version of this work has been published in [21]. We have made a significant extension of the initial framework in this current version. (1) We have extended the theoretical framework to formulate the dynamic optimization problem based on resource availability at the fog nodes and the fog micro-services' resource demands. We have shown how a BO-based solution model can reasonably fit in the proposed optimization framework. (2) We have extended the evaluation with a detailed analysis of PTC's performance under various scenarios. We have further analyzed a use case over the PTC framework. In summary, the salient contributions of this paper are as follows.

1. We have formulated an optimization problem considering the fog micro-services' dynamic demands along with the dynamic resource availability at the fog nodes. We have shown formally that the proposed optimization is $\mathcal{NP}$-hard.
2. In order to solve the proposed optimization, we utilized a Bayesian optimization-based framework while considering the dynamicity and measurement noises that may come during the runtime of such a system.
3. We have evaluated the proposed framework thoroughly with an optical character recognition application. The proposed PTC framework is also evaluated with a

natural language processing application as a use case. These evaluations have shown the superiority of PTC compared to other baselines.
4. We have also tested the proposed PTC framework for scalability in Amazon EC2. We have observed from the experiments in Amazon EC2 that the proposed PTC framework is scalable while ensuring minimization of response time.

The rest of the paper is organized as follows. An extensive study of the existing literature is provided in Sect. 2. The proposed system architecture and design goals are discussed in Sect. 3. The micro-service placement is modeled as an optimization, which is discussed in Sect. 4. Based on the hardness of the proposed optimization, we provide a reinforcement learning-based solution of the proposed optimization, as discussed in Sect. 5. The performance evaluation of the proposed framework is provided in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Related work

Various works have focused on the micro-service deployment problem in fog computing. The authors in [28] present a metaheuristic approach to minimize the application response time in edge computing for micro-service-based applications. In [24], the authors have developed a programming infrastructure for geo-distributed applications, which launches the application modules and performs the migration of these modules between fog devices as containers. These approaches primarily choose the nearest node for application deployment and therefore do not consider that the nearest node might be loaded with clients' requests. In [7], Elgamal *et al.* have proposed a dynamic programming-based algorithm to partition operations in IoT applications. The operations are placed across edge and cloud resources to minimize the completion time of the end-to-end operations. VM-based micro-service placement and migration for mobile users have been discussed in [10] where the number of applications to be placed in fog devices has been maximized while minimizing the latency cost. In [11], the authors have studied the base station association, task distribution and VM placement for cost efficient fog-based medical cyber-physical systems. The authors have minimized the communication cost along with the VM deployment cost. Li et al.[16] have proposed a resource scheduling approach in edge computing-based smart manufacturing, ensuring latency constraints. However, all of these works have considered static workload at the fog devices; therefore, the placement algorithms are modeled mostly as a one-time optimization problem solved through some efficient heuristics.

A few works in the literature have focused on application placement at the fog devices based on dynamically monitoring of resources. Yigitoglu et al.[34] have proposed a containerized application placement strategy based on the current measurement of the fog workload. They have considered a hierarchical organization of fog devices based on their resource availability and placed the application at a suitable hierarchy. Taneja et al.[31] have presented a module mapping algorithm for efficient resource utilization by placing the application modules in fog–cloud infrastructure. However, these works still use one-time scheduling of the fog workloads depending

on the immediate availability of the computing resources at the fog nodes. Although such approaches can show good performance for short term workloads, the performance severely suffers when the execution time is a bit longer. The authors in [27] have proposed an orchestration system where the micro-services are placed in fog devices if the resources are available; otherwise, the micro-service would be placed in the cloud node. The authors have also analyzed if there is a need to offload the micro-service in the cloud or queue it to one of the busy fog devices. In [35], a container-based task scheduling and reallocation mechanism are designed to optimize the number of concurrent tasks in fog computing-based smart manufacturing. These approaches are mostly based on periodic measurements and re-execution of the full optimization, which is a significant overhead for the system. In [23], the authors have developed an orchestration tool based on Kubernetes. They have extended it with self-adaptation and network-aware placement capabilities. The authors have proposed a model-based reinforcement learning solution to solve the elasticity problem. The work of [13] has presented a distributed placement policy that optimizes energy consumption at fog nodes and communication costs. They have modeled the service placement problem as a combinatorial auction market. The authors in [22] have given a reinforcement learning-based approach that manages the containers' horizontal and vertical elasticity. In the next step, container placement is performed by solving an integer linear programming problem or using a network-aware heuristic. Nevertheless, these works do not consider the dynamic migration of fog workloads across different fog nodes depending on the nodes' primary workload. Consequently, the micro-services execution performance suffers when there is a spike in the fog nodes' primary workload. In [32], the authors have proposed an dynamic programming-based offline micro-service coordination algorithm. Also, they have proposed a reinforcement learning-based online micro-service coordination algorithm for dynamic micro-service deployment. Table 1 presents the comparison of these works.

Our proposed approach attempts to address the limitations of the works mentioned above as PTC can provide micro-service isolation by using lightweight virtualization, i.e., containers. Again, PTC considers the dynamic workload of the fog computing environment by doing BO-based online learning to decide the best micro-service placement and a migration strategy. Therefore, in contrast to the existing works, we consider a fully dynamic system where the time-varying behavior of both the fog workloads and the computing nodes has been taken care of.

3 System architecture and design goals

The overall objective of the proposed PTC orchestration framework is to support the following four requirements for micro-service placement over a fog architecture—(a) development of a lightweight technology suitable for micro-service deployment over fog devices, (b) support service isolation for micro-service execution over fog devices, (c) design of an online micro-service placement mechanism catering to the dynamic workload and resource availability characteristics, and (d) seamless migration of micro-services to maintain application QoS of both the primary workload

Table 1 Comparison with existing approaches

Work	Containers	Micro-service	Migration	Bayesian optimization	Scalability
Stevant et al., 2018 [28]	✓	✓	✗	✗	✓
Saurez et al., 2016 [24]	✓	✓	✓	✗	✗
Elgamal et al., 2018 [7]	✓	✓	✗	✗	✓
Gonccalves et al., 2018 [10]	✗	✗	✓	✗	✗
Gu et al., 2015 [11]	✗	✗	✓	✗	✓
Li et al., 2019 [16]	✗	✓	✗	✗	✗
Yigitoglu et al., 2017 [34]	✓	✓	✗	✗	✓
Taneja et al., 2017 [31]	✗	✓	✗	✗	✓
Souza et al., 2018 [27]	✗	✓	✓	✗	✗
Yin et al., 2018 [35]	✓	✗	✓	✗	✗
Rossi et al., 2020 [23]	✓	✗	✗	✗	✓
Kayal et al., 2019 [13]	✗	✓	✗	✗	✓
Rossi et al., 2019 [22]	✓	✗	✗	✗	✓
Wang et al., 2019 [32]	✗	✓	✓	✗	✓
PTC	✓	✓	✓	✓	✓

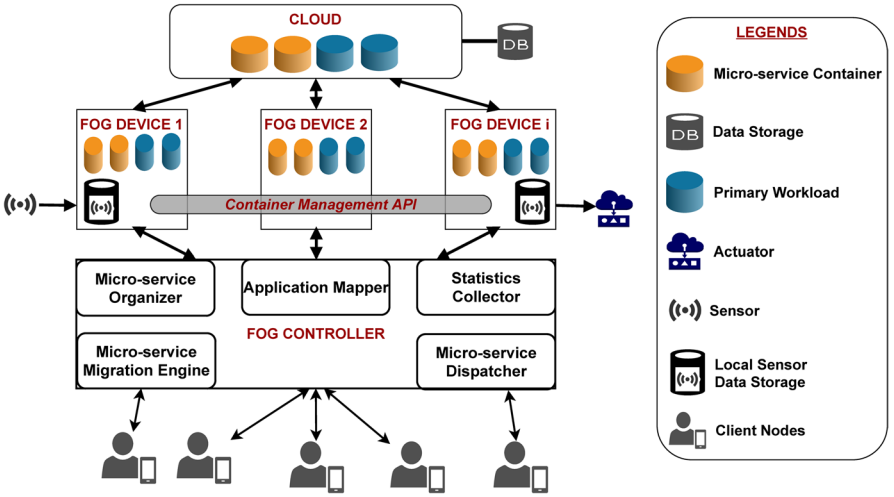

Fig. 1 PTC framework

and the micro-service workload. We discuss the architecture of the PTC framework (shown in Fig. 1) in this section.

Client Node The service request is generated by the client nodes. For example, a client might request the fog service to generate a consolidated report based on the sensing data captured by IoT devices. To do so, the client sends `REST` queries to the fog controller device.

Fog Controller Device The fog devices are connected to the fog controller device. This edge server receives the workload requests in terms of containerized micro-services and schedules them over the fog nodes. The different components of the *Fog Controller Device* are as follows. The `application mapper` maps the request–response to an application from where the requests originate. The `micro-service organizer` module divides the application into micro-services following the existing approaches such as ENTICE [14]. Also, the organizer module performs containerization of the micro-services. The `micro-service dispatcher` module finds the fog nodes to place the micro-services. The `statistics collector module` helps the dispatcher module to identify a suitable placement. The statistics collector module periodically monitors the available resources of the fog nodes. The micro-services are sent to the fog nodes by a `micro-service migration engine`, once the dispatcher module finds the placement. If the fog nodes become overloaded with their primary workloads, the micro-service dispatcher and the migration engines regenerate the placement.

Fog Device The containerized workloads are run in the fog devices. In Fig. 1, the blue-colored containers are the fog nodes' primary workloads. The micro-service containers utilize the remaining resources of the fog devices. We represent the micro-service containers with orange-colored containers (Fig. 1). The micro-service containers also need to communicate with the sensors and the actuators. The `local sensor data storage` module helps in this interaction. To seamlessly migrate the micro-service containers, we use a distributed `container management` API. The container management API is built using *"remote procedure call"* (`RPC`).

Cloud: The fog devices are connected to the cloud. Fog devices are always the first choice for placing a service. However, the proposed framework allows placement of a service in a cloud server if no fog server is available at a particular time instant.

In this framework, we combine two different approaches to design the overall execution model of PTC.

(a) We use containers as the sandboxing environment that supports the workload isolation along with the seamless migration of workloads across fog nodes.
(b) We design an online optimization strategy to decide the target fog node where a containerized micro-service can be placed. The optimization module facilitates both the initial micro-service deployment as well as the migration requirements. We use the standard application container-based approach [5] for executing the micro-services in the fog nodes and for migrating the micro-services based on the condition.

However, the challenge here is to design the dynamic micro-service placement in fog computing depending on the time-varying primary workload of the fog devices. The placement mechanism needs to be online, as well as it needs to cater to the dynamic workload characteristics of the system.

Next, we formally model the placement mechanism as an optimization problem. The details are given in Sect. 4.

4 Dynamic micro-service placement over fog nodes—an optimization formulation

This section presents the mathematical modeling of the fog micro-service placement problem and its theoretical analysis. The proposed execution model of the micro-services considers the following facts.

1. The fog devices have their primary workloads, which are time-varying, and thus can observe a spike in the resource demand. The priority is given to the primary workloads. Hence, the fog micro-service's execution might suffer if the primary workload withholds the resources used by the fog micro-services earlier.
2. To cope with such problems, PTC enables seamless migration of unfinished micro-services from one fog device to another. Such migrations are facilitated by container-based sandboxing. We consider that the migrations are live [20], indicating that execution states of the micro-services can be saved and later resumed on a different fog device.
3. The fog controller continuously monitors the available resource statistics on the fog devices and periodically decides the target fog device for a micro-service execution.

Considering the above mentioned facts, we present the system model for designing the optimization function for dynamic micro-service placement and execution over fog devices.

4.1 System model

We model the entire network, involving the end devices (sensors, actuators, edge devices), fog nodes (in-network devices like WiFi access point, gateway, router, core switches), fog controller, etc., as a weighted undirected communication graph. Table 2 shows the notations used. Let $G = (V,E,W)$ be an undirected and weighted communication graph. Here, the set of physical nodes is V, the set of physical communication links is E, and the set of edge weights is W. The sets of sensors, actuators and fog devices are denoted as $S = (S_i \in (1, \ldots, s))$, $\Lambda = (\Lambda_i \in (1, \ldots, \lambda))$ and $F = (F_i \in (1, \ldots, n))$, respectively. Here, the total number of sensors, actuators and fog devices are s, λ and n, respectively. Therefore, $V = \{S \cup \Lambda \cup F\}$. $e_{i,j} \in E$ represents the physical communication link between $v_i \in V$ and $v_j \in V$. In this case, the weight of each edge signifies the latency of the links.

We consider that the time is divided into l slots, where each slot is of length ι. Therefore, we define the system time vector $T = (\tau_t : t \in (1, \ldots, l))$. We also assume that each event either starts at the beginning of the slot or ends at the end of the slot. Therefore, all the time calculations done in this paper are in terms of number of slots. Let the weight of the shortest path between v_{i1} and v_{i2} be $D(v_{i1}, v_{i2})$, which signifies the communication latency between two nodes in the communication graph in terms of number of slots. Let k edge applications

Table 2 Notation table

Notation	Significance
G	An undirected and weighted communication graph
V	Set of physical nodes
E	Set of physical communication links
W	Set of edge weights
S	Set of sensors
Λ	Set of actuators
F	Set of fog nodes
s	Total number of sensors
λ	Total number of actuators
n	Total number of fog devices
$e_{i,j}$	The physical communication link between node v_i and node v_j
l	Number of time slots
ι	Length of a time slot
T	System time vector
$D(v_{i1}, v_{i2})$	Weight of the shortest path between node v_{i1} and node v_{i2}
k	Number of edge applications
$\mathbf{\Psi}$	Set of applications
$p_{x,y}$	y^{th} micro-service of a service P_x
h_x	Total number of atomic micro-services of a service P_x
$\mathbf{R}(F_i)$	The resource vector available in a fog node F_i
$\mathbf{\Gamma}(p_{x,y})$	The resource vector required by micro-service $p_{x,y}$
$\gamma_{x,y}^{q}$	The amount of resource type q required to execute $p_{x,y}$
f	The total different types of resources
r_i^q	The total available quantity of resource type q at F_i
A	Micro-service allocation matrix
$O(A, x, y)$	The occupancy of a fog node's resources by $p_{x,y}$ at a time instance
$Seq_{exe}(x, y)$	Fog device execution sequence by $p_{x,y}$
$[I]_{1\times n}$	The row vector of the fog device indices
T_{CPU}	Processing time
δ_i	Million instructions per second executed by fog node F_i
$\xi_{x,y}$	Million instructions required by $p_{x,y}$
$M_{x,y}$	Sub-vector of $Seq_{exe}(x, y)$
T_{MGR}	Migration time
$\Delta_{x,y}$	Constant delay factor for migration of $p_{x,y}$
ψ	The demand vector of sensors
α	The demand vector of actuators
$\psi_{x,y}$	The sensors required by $p_{x,y}$
α_x	The actuators required by $p_{x,y}$
$[I]_{1\times s}$	The row vector of the sensor indices
$[I]_{1\times \lambda}$	The row vector of the actuator indices
T_{DF}	The data fetch time
T_{ACT}	The actuation time

Table 2 (continued)

Notation	Significance
$\Omega(A, i, t)$	The executing micro-service vector at fog device F_i at time slot t
$\Theta(A, i, q, \Gamma, t)$	The amount of occupied resource of type q at time slot t
$T_{Resp}^{max}(A)$	The maximum response time taken by the applications
$\mathcal{R}(A)$	The resource availability at a particular time instant t
$\mathcal{D}_d$	The set of prior observations after d iterations
$\mu(\circ)$	The mean function
$K(\circ, \circ)$	The covariance kernel function
Φ	The standard normal cumulative distribution function
ϕ	The standard normal density function
EI	Acquisition function
ζ	Noise

are present in the system, such that $\mathbf{\Psi} = \left(P_x : x \in (1, \ldots, k)\right)$. $\mathbf{\Psi}$ is the set of applications. These applications need to execute their micro-services over the fog devices. An application can be decomposed into multiple micro-services, as mentioned earlier. We define $p_{x,y}$ as the y^{th} micro-service of P_x. Hence, $P_x = \left(p_{x,y} : y \in (1, \ldots, h_x)\right)$, where h_x is the total number of atomic micro-services of P_x. We consider that each micro-service can execute independently.

We denote $\mathbf{R}(F_i)$ as the resource vector available in a fog node F_i. This resource vector is time-varying and keeps on changing depending on the primary workload of the fog device. The resource vector required by micro-service $p_{x,y}$ is denoted by $\mathbf{\Gamma}(p_{x,y})$. We assume that the initial resource vector required by $p_{x,y}$ does not change over time, however depending on the partial completion of a micro-service's execution, the residual resource requirement might change. $\mathbf{\Gamma}(p_{x,y}) = (\gamma_{x,y}^q \in \mathbb{R} : q \in (1, \ldots, f))$, where each component $\gamma_{x,y}^q$ signifies the amount of resource type (central processing unit, memory, network bandwidth, etc.) q required to execute $p_{x,y}$ and f is the total different types of resources. Similarly, $\mathbf{R}(F_i) = (r_i^q \in \mathbb{R} : q \in (1, \ldots, f))$, where r_i^q is the total available quantity of resource type q at F_i. Each fog device F_i can host more than one micro-services depending on its available resources.

We define a micro-service allocation matrix $[A] = (A_{x,y,i,t} \in (0, 1) : (x \in (1, \ldots, k), y \in (1, \ldots, h_{max}), i \in (1, \ldots, n), t \in (1, \ldots, l))$ that gives the mapping between micro-services and fog nodes at each time stamp, where $h_{max} = \max_x(h_x)$, and each element $A_{x,y,i,t} = 1$ if $p_{x,y}$ is assigned to F_i at time slot t, otherwise $A_{x,y,i,t} = 0$. The task of the fog controller is to generate this occupancy metric dynamically based on its observation of the available resources at the fog devices. The allocation matrix changes over time, and such changes typically trigger a service migration from one fog device to another. Consequently, the occupancy ($O(A, x, y)$) of a fog node's resources by $p_{x,y}$ at a time instance is defined as follows.

$$\left[O(A,x,y)\right]_{n\times l} = \left(A_{x,y,i,t} \in (0,1) \,:\, i \in (1\ldots n), t \in (1\ldots l)\right) \tag{1}$$

Due to the dynamic workload characteristics of the fog devices, we consider that a micro-service may complete its execution over multiple fog nodes. Therefore, the time to calculate the total response time for an application has the following components—(a) processing time, (b) migration time and (c) data fetch and actuating time. We compute these three components as follows.

4.2 Computation of the processing time

As we mentioned earlier, a micro-service may get placed over a set of fog devices sequentially to complete its execution. Let the fog device execution sequence by $p_{x,y}$ be $Seq_{exe}(x,y) = [I]_{1\times n} \times [O(A,x,y)]_{n\times l}$, where $[I]_{1\times n} = [1,2,\ldots,n]$ represents the row vector of the fog device indices. We give the processing time (T_{CPU}) required by $p_{x,y}$ in order of number of time slots as follows. Here, F_i is capable of executing δ_i million instructions per second (MIPS), and $p_{x,y}$ requires $\xi_{x,y}$ million instructions.

$$T_{CPU}(x,y,A) = \sum_{i \in Seq_{exe}(x,y)} \frac{\xi_{x,y}}{\delta_i t} \tag{2}$$

4.3 Computation of the migration time

During the execution, a micro-service may require migration from one fog device to another, which incurs additional migration time. Let the sub-vector of $Seq_{exe}(x,y)$ be $M_{x,y} = (M^b_{x,y} = Seq_{exe}(x,y)_b \,:\, Seq_{exe}(x,y)_b \neq Seq_{exe}(x,y)_{b+1})$, where b is a particular time slot. Here, the fog node migration vector by $p_{x,y}$ is $M_{x,y}$. Therefore, the migration time (T_{MGR}) is computed as follows.

$$T_{MGR}(x,y,A) = \sum_{M^b_{x,y}} (\Delta_{x,y} \times D(M^b_{x,y}, M^{(b+1)}_{x,y})) \tag{3}$$

Here, $\Delta_{x,y}$ is a constant delay factor for migration of $p_{x,y}$, which depends upon the amount of data to be transferred from one fog device to another. Also, the weight of the shortest path between v_i and v_j is $D(v_i, v_j)$. $D(v_i, v_j)$ is signifying the communication latency between two nodes in terms of number of slots.

4.4 Computation of the data fetch and actuating time

Apart from migration, communication overhead plays a significant role in initial data fetching from sensors and triggering actuators. Let the demand vector of sensors and actuators are $\psi = (\psi_{x,y,i} \,:\, x \in (1,\ldots k), y \in (1,\ldots h_{max}), i \in (1,\ldots s))$ and $\alpha = (\alpha_{x,i} \,:\, x \in (1,\ldots k), i \in (1,\ldots \lambda))$, respectively. Here, $\psi_{x,y,i} = 1$ if $p_{x,y}$ requires S_i, otherwise it is 0. Similarly, $\alpha_{x,i} = 1$ if P_x requires Λ_i and it is 0 for all other cases. Let $\psi_{x,y} = [I]_{1\times s} \times (\psi_{x,y,i} \,:\, i \in (1,\ldots s))$ and $\alpha_x = [I]_{1\times\lambda} \times (\alpha_{x,i} \,:\, i \in (1,\ldots \lambda))$

are the sensors and actuators required by $p_{x,y}$, respectively. The row vector of the sensor indices is $[I]_{1\times s} = [1, 2, \ldots, s]$ and the row vector of the actuator indices is $[I]_{1\times\lambda} = [1, 2, \ldots, \lambda]$. The data fetch time ($T_{DF}$) of $p_{x,y}$ is given below.

$$T_{DF}(x, y, A) = \max_{i\in\psi_{x,y}}(D(M^1_{x,y}, i)) \tag{4}$$

The actuation time (T_{ACT}) is given below.

$$T_{ACT}(x, y, A) = \max_{i\in\alpha_x}(D(M^{|M_{x,y}|}_{x,y}, i)) \tag{5}$$

We define the executing micro-service vector ($\Omega(A, i, t)$) at fog device F_i at time slot t as follows.

$$\Omega(A, x, y, i, t) = \left(A_{x,y,i,t} \in (0, 1) : x \in (1 \ldots k), y \in (1 \ldots h_{max})\right) \tag{6}$$

The amount of occupied resource of type q at time slot t is given as follows.

$$\Theta(A, i, q, \mathbf{\Gamma}, t) = \left(\sum_{x,y}\left(\Omega(A, x, y, i, t) \times \gamma^q_{x,y}\right)\right) \tag{7}$$

According to the capacity property, cumulative occupied resources by micro-services executing on a single fog device must not surpass the total available resource at that fog device. A valid allocation matrix must satisfy the following capacity property given in Equation (8).

$$\underset{i,q,t}{\forall}\ \Theta(A, i, q, \mathbf{\Gamma}, t) \leq r^q_i \tag{8}$$

If the capacity property is satisfied by the allocation matrix, then the total response time required by $\mathbf{\Psi}_x$ can be calculated using Equations (2) to (5). Therefore, we calculate the total response time required by $\mathbf{\Psi}_x$ using Equations (2) to (5), as follows.

$$T_{Resp}(A, x) = \max_{y\in(1\ldots h_{max})}\begin{pmatrix} T_{DF}(x, y, A) + T_{CPU}(x, y, A) \\ +T_{MGR}(x, y, A) + T_{ACT}(x, y, A) \end{pmatrix} \tag{9}$$

4.5 Problem definition

We represent the formal definition of the micro-service placement problem as follows. Given the communication graph G and available resources ($\mathbf{R}(F_i)$), the micro-service placement problem finds an allocation schedule (A) for each micro-service ($p_{x,y}$) with required instructions ($\xi_{x,y}$) and resources ($\mathbf{\Gamma}(p_{x,y})$) such that the maximum response time taken by the applications ($T^{max}_{Resp}(A)$) is minimized. Therefore, mathematically, the problem can be represented as an optimization problem given by Equation (10). Therefore, we have

$$\begin{aligned} \underset{A}{\text{minimize}} \quad & T^{max}_{Resp}(A) \\ \text{subject to:} \quad & \\ & \mathcal{R}(A) \geq \mathbf{Z} \end{aligned} \tag{10}$$

where $\mathcal{R}(A)$ is the resource availability at a particular time instant t and $\mathbf{Z} = \left(z_{i,q,t} = 0 : i = (0, \ldots, n), q = (0, \ldots, f), t = (0, \ldots, l)\right)$.

In *"Minimax facility location problem"* (MFLP) [6], the cost to transfer the items to the demand site is optimized. In our micro-service allocation problem, we are interested to minimize the cost, i.e., the response time. We now show that the *"Minimax facility location problem"* (MFLP) [6] is polynomial time reducible to micro-service allocation problem (Equation (10)). Thus, the micro-service allocation is $\mathcal{NP}$-hard.

Theorem 4.1 *"Minimax facility location problem"* (*MFLP*) [6] *is polynomial time reducible to micro-service allocation problem given in Equation* (10).

Proof Let $\{L_i : i \in (1, \ldots, n)\}$ be the set of locations where a warehouse can be placed and $\{C_j : j \in (1, \ldots, m)\}$ be the set of demand sites that must be serviced. Suppose $c_{i,j}$ be the cost of delivery of items from L_i to C_j. In that case, the MFLP finds the subset of locations where the warehouses should be opened such that the maximum cost to transfer the items from facilities to the demand site is minimized. This is a known $\mathcal{NP}$-hard problem. To prove the $\mathcal{NP}$-hardness of our micro-service placement, we encode the instance mentioned above of MFLP.

The encoding of the instance mentioned above of MFLP can be done in the following way. In our case, the system consists of m applications, each having single micro-services (μ_j). Each of the micro-services (μ_j) needs to be placed in exactly one of the fog devices. Let $\{L_i : i \in (1, \ldots, n)\}$ be the set of fog devices and $\{C_j : j \in (1, \ldots, m)\}$ be the set of micro-services to be placed. $c_{i,j}$ is the cost (in terms of response time) of placing the j^{th} micro-service (C_j) in i^{th} fog device (L_i). The reduction can be made in polynomial time. This completes the polynomial time encoding of MFLP to our micro-service placement. The used fog devices represent the location where the warehouses can be set up from the solution allocation matrix. However, as the MFLP is a known $\mathcal{NP}$-hard problem; therefore, we can claim that our proposed micro-service placement problem is $\mathcal{NP}$-hard. □

5 Solution approach: Bayesian optimization for micro-service placement

As the optimization problem is *NP*-hard, we design an approximate solution of the proposed optimization, which is fast and provides quick decision about the placement depending on the system's dynamicity. We use Bayesian optimization (BO), a reinforcement learning framework that provides the solution of an optimization based on the historical observation and posterior distribution of the solution vector.

BO performs well in the scenarios when conducting a single experiment takes a higher time.

We define the micro-service allocation matrix as a solution configuration of BO. BO optimizes the objective function based on the prior observations and posterior distribution by conducting iterative experiments over the solution configurations. In our case, conducting an experiment is equivalent to test the performance of a given allocation matrix which is costly in terms of time taken to perform a test. Surprisingly, BO performs well in such cases where performing one single experiment takes a higher time.

To formulate the BO framework, we assume that the utility function $T^{max}_{Resp}(A)$ follows a normal distribution. BO executes initial experiments based on the prior belief function. After sufficient number of experiments, BO modifies the prior belief function based on the posterior distributions. BO uses an *"acquisition function"* (*EI*) to pick the configurations for the iterations. This leads to a near-optimal solution. Let $\mathcal{D}_d = \{(a_1, T^{max}_{Resp}(a_1)), \ldots, (a_d, T^{max}_{Resp}(a_d))\}$ be the set of prior observations after d iterations. We denote $p(T^{max}_{Resp}) = \mathcal{N}(\mu, K)$. Here, the mean function is $\mu(\circ)$, and the covariance kernel function is $K(\circ, \circ)$. We define the mean and covariance kernel function as follows.

$$\mu(a_u) = \mathbb{E}(T^{max}_{Resp}(a_u)) \tag{11}$$

$$K(a_u, a_v) = \mathbb{E}\Big(\Big(T^{max}_{Resp}(a_u) - \mu(a_u)\Big)\Big(T^{max}_{Resp}(a_v) - \mu(a_v)\Big)\Big) \tag{12}$$

Let us denote Φ and ϕ as the standard normal cumulative distribution function and the standard normal density function, respectively. We define $U_{min} = \underset{a \in \mathcal{D}_d}{min}(T^{max}_{Resp}(a))$, $\pi = \frac{U_{min} - \mu(a_d)}{\sigma(a_{d-1}, a_d)}$, and $\sigma(a_{d-1}, a_d) = \sqrt{K(a_{d-1}, a_d)}$. We choose the acquisition function (Equation (13)) which is recommended in [3].

$$EI(A|\mathcal{D}_d) = \begin{cases} 0 & \text{if:} \sigma(a_{d-1}, a_d) = 0 \\ \Big(\big(U_{min} - \mu(a_d)\big)\Phi(\pi)\Big) + \big(\phi(\pi)\sigma(a_{d-1}, a_d)\big) & \text{Otherwise} \end{cases} \tag{13}$$

However, $EI(A|\mathcal{D}_d)$ works well in case of unconstrained optimization. We take the procedure suggested by Gardner et al.[8] to satisfy our constrained optimization. By following their approach, we assume that $\mathcal{R} \Leftarrow \mathcal{A} \Rightarrow$ follows Bernoulli process, and *EI* can be modified to Equation (14).

$$EI^c(A|\mathcal{D}_d) = P(\mathcal{R}(A))EI(A|\mathcal{D}_d) \tag{14}$$

There can be observation noise in our setup due to a rise in loads in the fog devices, network delay, etc. Therefore, the proposed BO algorithm must handle noise. We assume that the noise (ζ) is a normally distributed random variable with zero mean, i.e., $\zeta = \mathcal{N}(0, \sigma_\zeta)$.

Algorithm 1: PTC

Input: Resource avalibility of the fog nodes, Resource requirement of the applications
Output: Allocation matrix for which the maximum response time taken by the applications is minimized

1 **Function** *Main()*:
  /* Function for Bayesian Optimization */
2 $\mathcal{D}_0 \leftarrow \emptyset$;
3 **for** $(d \leftarrow 1; HasConverged(\mathcal{D}_{(d-1)}); d++)$ **do**
4 $a_{nxt} \leftarrow \emptyset$;
5 **while** $Length(a_{nxt}) < n_{conf}$ **do**
6 Choose an allocation matrix a_i such that $a_i \notin \{\mathcal{D}_{(d-1)}\}$;
7 **if** *IsFeasible*(a_i) **then**
8 $a_{nxt} \leftarrow \{a_{nxt} \cup a_i\}$;
9 **else**
10 **go to** 5;
11 a_d = Configuration for which the aquisition function is minimum;
12 $\mathcal{D}_d = \left\{\mathcal{D}_{(d-1)} \cup \left(a_d, T^{max}_{Resp}(a_d) + \zeta\right)\right\}$;
13 **return** allocation matrix from $\mathcal{D}_d$ for which the maximum response time taken by the applications is minimized;

1 **Function** *IsFeasible*(a_κ):
  /* Checks feasibility of the allocation matrix */
2 **for** *all applications* $P_x \in \Psi$ **do**
3 **for** $p_{x,y} \in P_x$ **do**
4 $c = \{1, 2, \ldots f\}$;
5 **if** $\nexists_{i \in c} : r^q_i > \gamma^q_{x,y}$ **then**
6 **return** *False*;
7 **else**
8 Remove i from c;
9 **return** *True*;

PTC is described in Algorithm 1. In the Main() function, $\mathcal{D}_0$ is initialized to empty set in Line 2. Lines 3-12 iteratively finds the allocation or configuration for which the aquisition function is minimum. a_{nxt} is initialized to empty set in Line 4. Lines 5-10 generates $(n_{conf} - 1)$ numbers of allocation matrices (a_i). Each of the generated allocation matrices is checked for feasibility. If the allocation matrix is feasible, then that configuration is tested by applying it to the system. Once the testing process is over, the objective function is evaluated, and the next configuration (a_d) is determined based on the minimization of the acquisition function. Line 11 finds the configuration (a_d) for which the aquisition function is minimum. In Line 12, this new configuration (a_d) is added to the set of prior observations set, i.e., $\mathcal{D}_d$ set. This process is repeated until the system converges. PTC algorithm runs iteratively and it stops after reaching convergence. In the IsFeasible(a_κ) function, Line 2 runs for all applications. Line 3 runs for all micro-services of an application. In Line 4, c is the set of all resource types. Line 5-8 checks if any fog node is able to satisfy all the resource requirements of a micro-service. In each iteration, the PTC algorithm checks resource constraint for every micro-service. Therefore, the

computational complexity of PTC is $\mathcal{O}(k * h_x)$. Here, k is the number of edge services and h_x is the total number of micro-services of a service.

6 Performance evaluation

We have implemented the PTC framework in a testbed and a public cloud (Amazon EC2). In the testbed, we analyze the performance of PTC in a realistic environment under a constrained setup. To analyze the system scalability of PTC, we perform the experiments on a large scale over Amazon EC2.

6.1 Testbed setup

In the testbed, the fog nodes are implemented using `Raspberry Pi 3 model b` (https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (Access: 2021/10/13 08:51:59)) devices, which works like a network gateway. The fog nodes are connected over a local network and provide packet forwarding services to the connected devices. The devices have been configured with `Ubuntu 16.04.2 long-term support (LTS)` (https://www.ubuntu.com/) operating system. `Docker` (https://www.docker.com/) containers are used to provide the sandboxing environment of the micro-services running over the fog nodes. Docker provides a platform for lightweight containers by application layer virtualization. The PTC framework is connected to a private cloud available in our institute.

Figure 2 shows the entire topology of the testbed architecture, which emulates the framework components, as shown in Fig. 1. We give the description of the components used in testbed experiments in Table 3. An ethernet switch has been used to connect various networking components to the local area network (LAN) in the testbed. We have used two switches to connect the fog devices and the client nodes to the ethernet switch. These switches interconnect various fog devices, the fog controller and the client node. A virtual machine running in the private institute cloud is used to host micro-services using Docker over a cloud environment. The fog controller device has `Ubuntu 16.04.2 LTS` (https://www.ubuntu.com/) as the operating system. Client applications are executed on another workstation having a similar configuration to that of the fog controller device. However, the client node only uses a custom TCP socket program to request the fog applications.

6.2 Experimental methodology

Table 4 provides the details of the values taken during the experiments in the testbed. The delays have been set up using the Unix `tc` utility (http://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html). We use `stress-ng` (http://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html) for emulation of memory load due to the primary workload in the fog devices. For memory status monitoring of fog devices and the fog controller, we use Linux `free utility` (http://manpages.ubuntu.com/manpages/bionic/man1/free.1.html). The CPU usage is measured

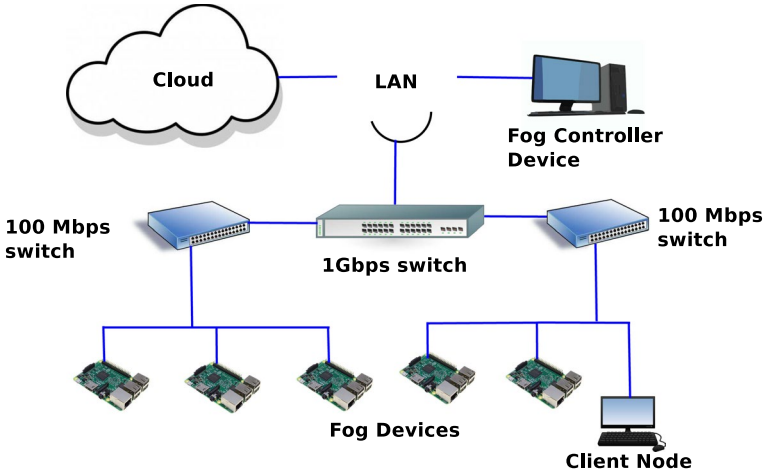

Fig. 2 Testbed topology for the experiments

Table 3 Devices used in testbed experiments

Component	Description
Fog devices	`Raspberry Pi 3 model b` devices (a quad-core 1.2 GHz Broadcom BCM2837 64-bit central processing unit (CPU) and 1 GB random access memory (RAM))
Ethernet wwitch	Netgear GS608 8-Port 1 Gigabit ethernet switch
Desktop wwitches	TP-Link TL-SF1008D 8-Port 10/100Mbps switches
Cloud VM	A cloud VM (4 GB RAM with 2 CPUs) running in the private institute cloud
Fog controller device	Quad-core 3.2 GHz Intel Core i5-4570 processor with 4 GB of RAM
Client node	Quad-core 3.2 GHz Intel Core i5-4570 processor with 4 GB of RAM

using the `iostat` utility (https://man7.org/linux/man-pages/man1/iostat.1.html). The network status is obtained by the `bmon` utility (http://manpages.ubuntu.com/manpages/bionic/man8/bmon.8.html). We use `scp` (http://manpages.ubuntu.com/manpages/trusty/man1/scp.1.html) to transfer files from the fog devices to the fog controller. `Scikit-Optimize (skopt)` (https://scikit-optimize.github.io/) python library is used for the implementation of the BO algorithm. We consider `optical character recognition (OCR)` as a service for testing; for this, we have run `tesseract-OCR web service` (https://hub.docker.com/r/guitarmind/tesseract-web-service/) as a docker container. Also, we have tested PTC with an NLP application, as discussed in Sect. 6.8.

6.3 Competing heuristics

We consider *Foglets* [24], *first-fit* [27], *best-fit* [27] and *mobility-based* [10] mechanisms as the baseline methods to analyze the performance of PTC. In Foglets, the

Table 4 Values of different parameters used in testbed

Parameter	Value
Number of fog devices	5
Number of application types	1
Number of micro-services per service or application	5
Network delay: RTT for fog device–fog device	10–18 ms (if not closest)/0.5 ms (if closest)
Network delay: RTT for fog device–cloud	200 ms
Network delay: RTT for controller device–cloud	200 ms
RAM load in fog devices	20 MB (low loaded)/100 MB (high loaded)
RAM needed for docker container, i.e., micro-service	40 MB
Number of parallel applications or number of scenarios	1 to 8
Number of runs per scenario	3-4

authors have proposed a programming infrastructure for geo-distributed applications, which uses a discovery and deployment protocol to find the fog computing devices with sufficient resources to host an application component. The first-fit mechanism selects the available fog devices for deployment and processing of an image. On the contrary, the best-fit method sorts the processing requests in ascending order according to their demands. In the mobility-based mechanism, the authors have developed a placement to minimize the applications' latency.

6.4 Application response time

We show the performance of PTC in terms of response time in Fig. 3(a). It is observed that PTC is faster than the baselines. The distribution of average response time for PTC converges within 50 seconds. At the same time, the distribution of average response time for baselines converges between 53 seconds and 88 seconds. It is found that PTC converges quickly to an optimal or near-optimal value (within 30 iterations). Table 5 presents a performance comparison of PTC with respect to the baseline approaches as shown in Fig. 3(a).

The PTC algorithm uses BO-based reinforcement learning. As BO can reach the desired solution in fewer iterations, the response time of the application reduces. BO finds the selected points in the search space with relatively few function evaluations. This optimization technique induces adaptive learning over the PTC framework and therefore makes PTC intelligent in dynamically selecting the target fog devices for micro-service placements. The BO-based adaptive reinforcement learning framework helps PTC optimize the utility function with prior observations and posterior distribution, such that PTC can reduce the overall response time. Consequently, the proposed framework learns the dynamic fog environment and their primary

Table 5 Performance of PTC in terms of average response time

Approach	Average response time
Foglets [24]	24.21–80.35 sec
First-fit [27]	16.74–88 sec
Best-fit [27]	11.48–57.39 sec
Mobility-based [10]	12–53 sec
PTC	9.71–50 sec

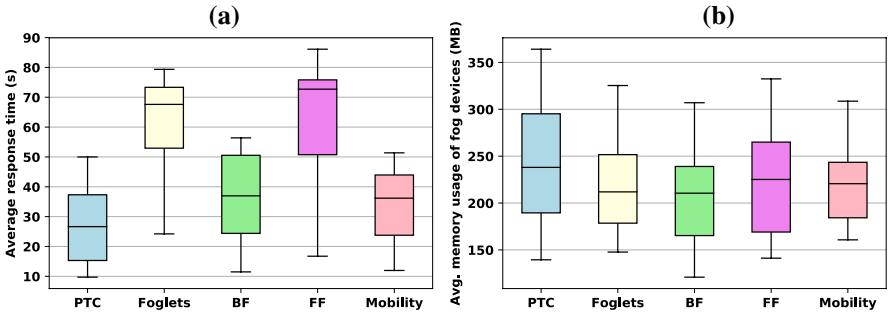

Fig. 3 **a** Distribution of average response time and **b** Distribution of average memory usage of the fog devices

workloads over time and can decide accordingly to support the applications' low response time.

On the contrary, the Foglets mechanism places the application components in fog devices that are closest. Therefore, it cannot find a suitable allocation matrix as the nearest fog nodes' computing resources might be blocked with their primary workloads. In the first-fit mechanism, the available fog devices which have sufficient resources are chosen for the placement. However, this method also does not consider the dynamic primary workload. Thus, in this case, the response time is more than PTC. The best-fit algorithm places the micro-services according to their resource demand. The best-fit response time is also more than PTC as the best-fit algorithm takes more time to deploy the micro-services in fog devices.

On the other hand, mobility-based placement only checks if the applications' latency is minimized while allocating them in the fog devices. This approach tries to maximize the number of tasks deployed in the fog landscape, ensuring the latency constraint. Thus, the average response time is higher in the mobility-based scheme than PTC, particularly when the primary workload observes a spike in the resource demand.

6.5 Resource usage of the fog devices

The memory consumption of the fog nodes is depicted in the Fig. 3b. It may be observed that the PTC consumes more amount of memory than the baseline

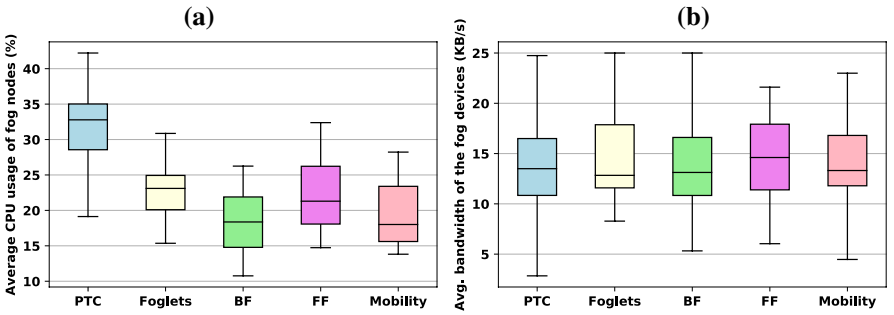

Fig. 4 **a** Distribution of average CPU usage of the fog devices and **b** Distribution of average network bandwidth usage of the fog devices

algorithms. The density of memory usage distribution is higher in PTC (in the range of 139 MB–366 MB) than the baselines. Further, we show the average CPU usage distribution of the fog nodes in Fig. 4a. The CPU usage is also higher in PTC (in the range of 18.43%–44%) than the CPU usage of the baseline algorithms. In a similar line, the bandwidth usage is also higher in PTC (in the range of 3 KB/s–24.74 KB/s) in comparison with other baselines, as shown in the Fig. 4b.

As these statistics are collected over the fog nodes, it indicates the average resource consumption by the micro-services placed on that fog nodes. The figures suggest that PTC can provide better resource utilization to the running micro-services compared to other baselines. We observe that the baseline mechanisms fail to place the micro-services in the most optimized fog devices having better availability of excess resources over time. They only consider the instantaneous measurements and, therefore, do not consider the fog devices' time-varying primary workload characteristics. Consequently, the running micro-services get better computing platform in PTC, resulting in a low response time, as we have seen earlier.

6.6 Resource usage of the fog controller

We show the resource consumption of the fog controller node in Fig. 5(a). As the proposed PTC algorithm runs for multiple iterations, the PTC algorithm's memory consumption is more than the baselines. The controller's average memory usage is higher (in the range of 756 MB–922 MB) in PTC than the baselines. The average CPU consumption of the fog controller device is shown in Fig. 5(b). It is also higher in PTC (in the range of 26%–74%). The average bandwidth usage of PTC is shown in Fig. 6(a). It is also higher (in the range of 11 KB/s–87.28 KB/s) in PTC compared to the baselines. In summary, we observe that the trade-off is in the fog controller's resource usage, where the PTC controller needs more resources than other baselines. However, considering the improvement observed in the application response time, this trade-off is acceptable as the average increase in the resource usage is not very significant, although it is higher than the baselines.

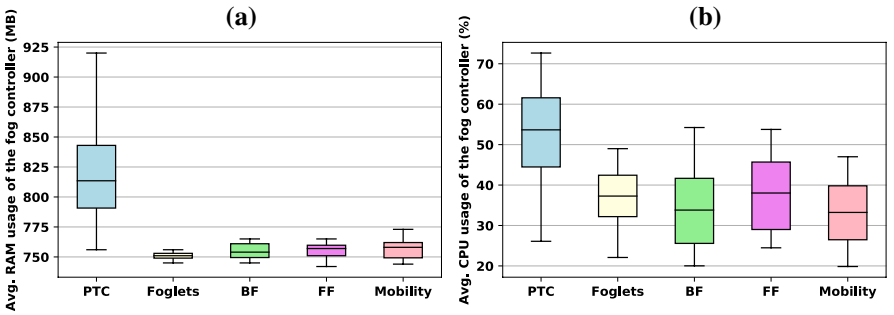

Fig. 5 **a** Distribution of average memory usage of the fog controller device and **b** Distribution of average CPU usage of the fog controller device

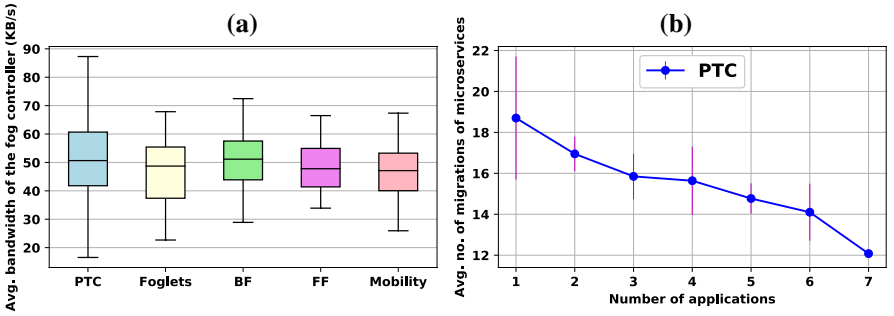

Fig. 6 **a** Distribution of average network bandwidth usage of the fog controller device and (b) Avg. number of migrations of the micro-services

6.7 Average number of micro-service migrations

The proposed PTC algorithm migrates the containerized micro-services to a new fog node if the current fog node becomes loaded. The migration ensures the minimization of service response time. The average number of migrations of the micro-services in Fig. 6b. It is observed that though the total number of migrations increases, the average number of migrations decreases with the increase in the number of applications. It indicates the system's stability at high load.

6.8 Case study with an NLP (Natural Language Processing) application

We have tested PTC with an NLP application—text summarization toolbox[1]. Such an application is important because it supports many practical situations. The evaluation is performed in our fog computing testbed with the configurations given in

[1] https://glowingpython.blogspot.com/2014/09/text-summarization-with-nltk.html (Access: 2021/10/13 08:51:59).

Table 6 Performance of PTC in terms of average response time for the NLP application

Approach	Average response time
Foglets [24]	59.12–131 sec
First-fit [27]	58.86–142.2 sec
Best-fit [27]	49.59–144 sec
Mobility-based [10]	49.37–143.29 sec
PTC	36.75–107 sec

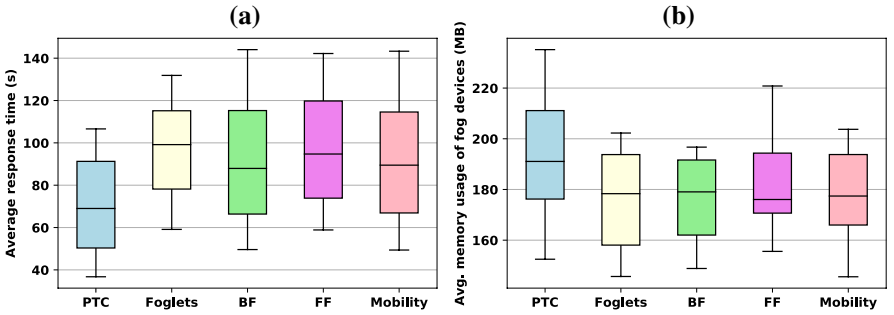

Fig. 7 **a** Distribution of average response time and **b** Distribution of average memory usage of the fog devices

Table 4. The testbed setup given in Sect. 6.1 has been used for the case study. It is observed that the average response time is less for PTC than the baselines. The distribution of the response time in PTC converges within 107 seconds (Fig. 7a), whereas distributions of the response time for the baseline algorithms converge between 131 seconds and 144 seconds. Table 6 shows a performance comparison of PTC with respect to the baseline approaches for the NLP application as shown in Fig. 7a.

The average memory usage is more in PTC, as we have seen earlier. It is in the range of 152 MB–235 MB (shown in Fig. 7b), whereas the average memory distribution of baselines converges within 220 MB for first-fit, within 202 MB for Foglets, within 203 MB for mobility-based and within 196 MB for best-fit, respectively. The CPU usage of the fog devices is higher in PTC in the case of the NLP application. It is in the range of 6%–19% (Fig. 8a) for PTC. The CPU usage distribution of baselines converges within 13% for first-fit, within 14% for best-fit and Foglets methods. The distribution of CPU usage converges within 15% for mobility-based algorithm. The distribution of bandwidth usage of the fog devices is also marginally higher in PTC than the baseline algorithms. For PTC, it is in the range of 5 KB/s–17 KB/s (Fig. 8b). On the contrary, the distribution of network bandwidth usage converges within 18 KB/s for the best-fit mechanism, within 17 KB/s for Foglets and within 16 KB/s for the first-fit and the mobility-based algorithm.

Similar to the previous observation, the average memory usage distribution of the controller is more in PTC compared to other baselines, which is in the range of 756 MB–930 MB (Fig. 9a). The fog controller's average memory usage distribution

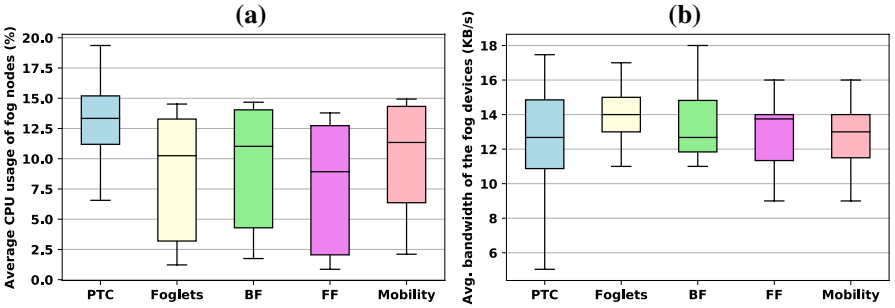

Fig. 8 **a** Distribution of average CPU usage of the fog devices and **b** Distribution of average network bandwidth usage of the fog devices

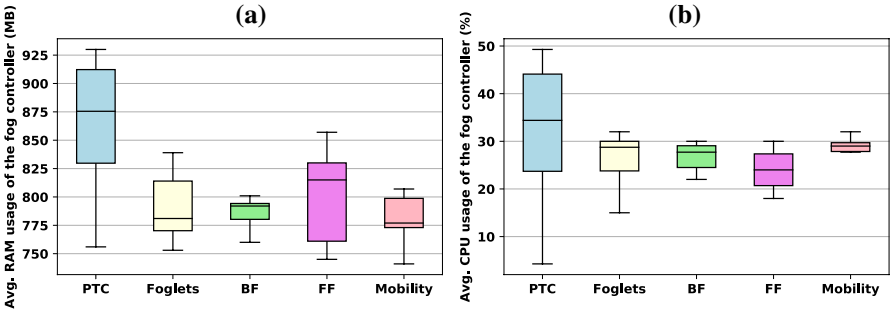

Fig. 9 **a** Distribution of average memory usage of the fog controller device and **b** Distribution of average CPU usage of the fog controller device

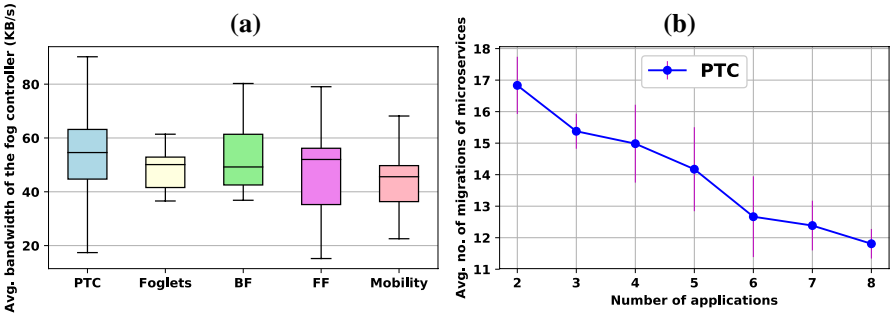

Fig. 10 **a** Distribution of average network bandwidth usage of the fog controller device and **b** Avg. number of migrations of the micro-services

converges within 839 MB for Foglets, within 800 MB for best-fit, within 857 MB for first-fit and within 807 MB for the mobility-based algorithm. The CPU usage of the controller is in the range of 4–49% for PTC. The CPU usage is shown in Fig. 9b. The distribution of baselines converges within 35% for mobility-based algorithm and 32% Foglets, respectively. It converges within 30% for best-fit and first-fit

Table 7 Performance of PTC in terms of average response time for scalability analysis

Approach	Average response time
Foglets [24]	175–650.82 sec
First-fit [27]	211.66–755.34 sec
Best-fit [27]	196–775 sec
Mobility-based [10]	214–634 sec
PTC	106.6–w318 sec

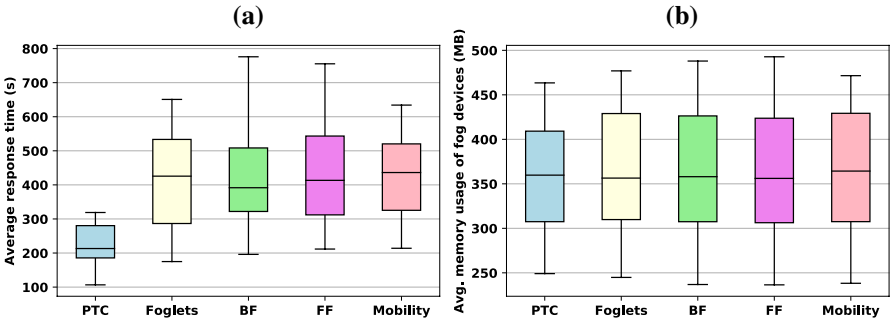

Fig. 11 **a** Distribution of average response time and **b** Distribution of average memory usage of the fog devices

algorithms. The distribution of the average network bandwidth usage of the PTC controller is marginally higher than other baselines. It is in the range of 17 KB/s–98 KB/s (Fig. 10a) for PTC. The distributions of baselines converge within 80 KB/s for best-fit, within 79 KB/s for first-fit, within 61 KB/s for Foglets and within 68 KB/s for the mobility-based mechanism.

The average number of migrations of the micro-services for the NLP application is shown in Fig. 10b. In this case, we also observe that though the total number of migrations increases, the average number of migrations drops with the increase in the number of applications, indicating the system's stability.

6.9 Scalability analysis in Amazon EC2

To test the scalability of the system, we have experimented with Amazon EC2. We have taken `tesseract-OCR web service` as a docker container for testing. We have taken 45 virtual machines (VM) in Amazon EC2. These VMs are considered as the fog devices for the experiments. The VMs have a 2.5 GHz Intel Xeon Family Processor with 1 GB RAM. The number of containers is increased from 80 to 360. The fog controller device has 1 GB of RAM. We show the average response time in 11a. We find that our proposed PTC framework can decrease the services' average response time than the baselines. The distribution of average response time for PTC converges within 318 seconds (Fig. 11a). Distributions of the baselines converge between 634 seconds and 775 seconds. On

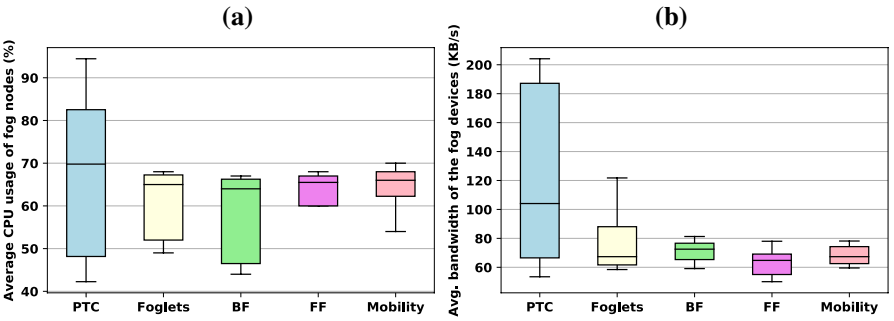

Fig. 12 **a** Distribution of average CPU usage of the fog devices and **b** Distribution of average network bandwidth usage of the fog devices

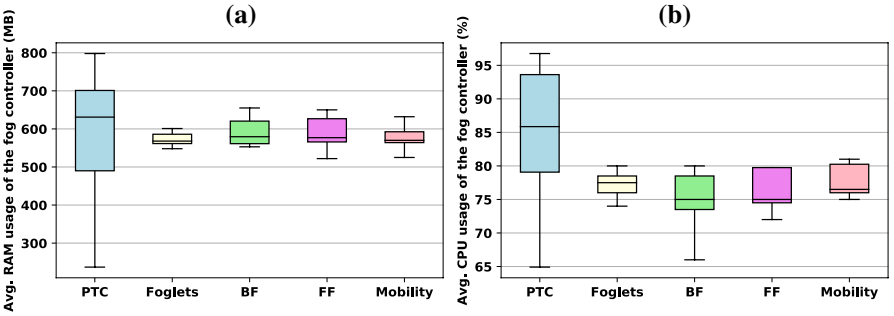

Fig. 13 **a** Distribution of average memory usage of the fog controller device and **b** Distribution of average CPU usage of the fog controller device

average, PTC performs better than the baselines in Amazon EC2 experiments. Thus, our PTC framework is scalable. Table 7 shows a performance comparison of PTC with respect to the baseline approaches for the scalability analysis as shown in Fig. 11a.

It may be observed that the average memory usage of the fog devices is in the range of 250 MB–463 MB in PTC in Amazon EC2 (shown in Fig. 11b). The average memory usage distribution for baseline algorithms converges within 492 MB for first-fit, within 476 MB for Foglets, within 471 MB for mobility-based and within 487 MB for the best-fit algorithm, respectively. The CPU usage of the fog devices is higher in PTC in Amazon AWS experiments. It is in the range of 42%–94% (Fig. 12a). The distribution for baselines converges within 68% for first-fit and Foglets, within 67% for best-fit and within 70% for the mobility-based algorithm, respectively. The distribution of bandwidth usage of the fog devices is marginally higher in PTC than other baselines when experimented with over Amazon EC2. It is in the range of 53 KB/s–204 KB/s (Fig. 12b) for PTC. On the contrary, such distributions converge within 81 KB/s for the best-fit mechanism, within 121 KB/s for Foglets, within 77 KB/s for first-fit and within 78 KB/s for the mobility-based algorithm. The overhead in terms of the fog controller's

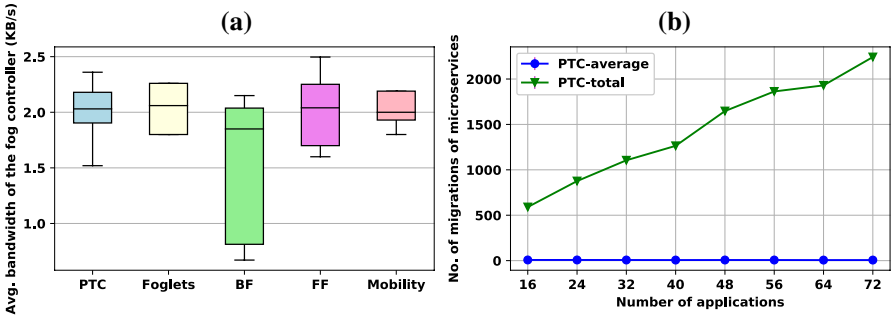

Fig. 14 **a** Distribution of average network bandwidth usage of the fog controller device and **b** Number of migrations of the micro-services

memory usage is shown in Fig. 13a. We have observed that the controller's average memory usage distribution is more in PTC in the Amazon EC2 experiments. It is in the range of 237 MB–798 MB (Fig. 13(a)). On the contrary, the fog controller's average memory usage distribution converges within 650 MB for first-fit, within 601 MB for Foglets, within 655 MB for best-fit and within 649 MB for the mobility-based algorithm.

Similar to our previous observations, the CPU usage of the controller is higher in PTC than other baselines and it is in the range of 65–97% (Fig. 13(b)). The distribution for baselines converges within 81% for mobility-based algorithm and 80% Foglets, respectively. It converges within 80% for best-fit and first-fit algorithm. The distribution of the average bandwidth distribution of the controller for PTC is in the range of 1.52 KB/s–2.36 KB/s (Fig. 14(a)). In contrary, the distribution for baselines converges within 2.15 KB/s for best-fit, within 2.49 KB/s for first-fit, within 2.26 KB/s for Foglets and within 2.19 KB/s for the mobility-based mechanism.

The number of migrations of the micro-services for the Amazon EC2 experiments is shown in Fig. 14(b). In these experiments, we observe that the total number of migrations increases with the number of applications. Also, the average number of micro-service migrations is in the range of 6.03 to 7.37.

7 Conclusion

With the widespread deployment of IoT-based services, the concept of fog computing can provide a useful solution for low-latency privacy-ensured data processing by utilizing the in-network processing capability of various devices. Although many existing works have focused on scheduling and deployment of application micro-services over fog nodes, the dynamicity of the fog devices' primary workloads is still a concern. In this paper, we have proposed PTC, a reinforcement learning-induced framework for continually monitoring the fog devices' primary workloads and accordingly dynamically deciding the execution platform for the application micro-services. PTC solves a hard time-varying optimization problem with Bayesian optimization, which dynamically finds a solution based on prior observations.

We have implemented PTC over a small-scale testbed setup and evaluated its performance over a large-scale emulation with Amazon EC2 clouds. The experiments confirm that PTC can reduce the application response time with better utilization of the fog device's excess resources, with the cost of a moderately complicated mechanism, which needs an additional edge server for execution.

We believe that the proposed framework can introduce dynamic execution of IoT data processing workloads by utilizing the fog devices' in-network computing facilities. Therefore, such a framework can be useful in various application scenarios, like smart home, smart manufacturing environments, etc. As future work, we plan to deploy the proposed framework with a complete IoT data collection and processing framework to build up an entire system by utilizing its performance. In future, we would like to perform evaluation of the proposed PTC framework with different IoT applications having different data processing needs.

References

1. Ahmad M, Amin MB, Hussain S, Kang BH, Cheong T, Lee S (2016) Health fog: a novel framework for health and wellness applications. J Supercomput 72(10):3677–3695
2. Ahmed A, Pierre G (2018) Docker container deployment in fog computing infrastructures. In: 2018 IEEE International Conference on Edge Computing (EDGE), pp. 1–8. IEEE
3. Alipourfard O, Liu HH, Chen J, Venkataraman S, Yu M, Zhang M (2017) Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 469–482
4. Alturki B, Reiff-Marganiec S, Perera C, De S (2019) Exploring the effectiveness of service decomposition in fog computing architecture for the internet of things. IEEE Transactions on Sustainable Computing
5. Bernstein D (2014) Containers and cloud: from lxc to docker to kubernetes. IEEE Cloud Comput 1(3):81–84
6. Drezner Z, Wesolowsky GO (1983) Minimax and maximin facility location problems on a sphere. Naval Res Logistics Quart 30(2):305–312
7. Elgamal T, Sandur A, Nguyen P, Nahrstedt K, Agha G (2018) Droplet: distributed operator placement for iot applications spanning edge and cloud resources. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 1–8. IEEE
8. Gardner JR, Kusner MJ, Xu ZE, Weinberger KQ, Cunningham JP (2014) Bayesian optimization with inequality constraints. ICML 2014:937–945
9. Goethals T, De Turck F, Volckaert B (2020) Near real-time optimization of fog service placement for responsive edge computing. J Cloud Comput 9(1):1–17
10. Gonçalves D, Velasquez K, Curado M, Bittencourt L, Madeira E (2018) Proactive virtual machine migration in fog environments. In: 2018 IEEE Symposium on Computers and Communications (ISCC), pp. 00742–00745. IEEE
11. 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
12. Javadzadeh G, Rahmani AM (2020) Fog computing applications in smart cities: a systematic survey. Wireless Netw 26(2):1433–1457
13. Kayal P, Liebeherr J (2019) Distributed service placement in fog computing: an iterative combinatorial auction approach. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 2145–2156. IEEE
14. Kecskemeti G, Marosi AC, Kertesz A (2016) The entice approach to decompose monolithic services into microservices. In: 2016 International Conference on High Performance Computing & Simulation (HPCS), pp. 591–596. IEEE
15. Li DC, Huang CT, Tseng CW, Chou LD (2021) Fuzzy-based microservice resource management platform for edge computing in the internet of things. Sensors 21(11):3800

16. Li X, Wan J, Dai HN, Imran M, Xia M, Celesti A (2019) A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing. IEEE Trans Industr Inf 15(7):4225–4234
17. Liao S, Wu J, Mumtaz S, Li J, Morello R, Guizani M (2020) Cognitive balance for fog computing resource in internet of things: an edge learning approach. IEEE Trans Mobile Comput
18. Mukherjee M, Shu L, Wang D (2018) Survey of fog computing: fundamental, network applications, and research challenges. IEEE Commun Surv Tutorials 20(3):1826–1857
19. Mutlag AA, Abd Ghani MK, Arunkumar Na, Mohammed MA, Mohd O (2019) Enabling technologies for fog computing in healthcare iot systems. Future Gener Comput Syst **90**, 62–78
20. Nadgowda S, Suneja S, Bila N, Isci C (2017) Voyager: Complete container state migration. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 2137–2142. IEEE
21. Nath SB, Chattopadhyay S, Karmakar R, Addya SK, Chakraborty S, Ghosh SK (2019) Ptc: Pick-test-choose to place containerized micro-services in iot. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE
22. Rossi F, Cardellini V, Presti FL (2019) Elastic deployment of software containers in geo-distributed computing environments. In: 2019 IEEE Symposium on Computers and Communications (ISCC), pp. 1–7. IEEE
23. Rossi F, Cardellini V, Presti FL, Nardelli M (2020) Geo-distributed efficient deployment of containers with kubernetes. Comput Commun 159:161–174
24. Saurez E, Hong K, Lillethun D, Ramachandran U, Ottenwälder B (2016)Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, pp. 258–269
25. Singh SP, Nayyar A, Kumar R, Sharma A (2019) Fog computing: from architecture to edge computing and big data processing. J Supercomput 75(4):2070–2105
26. Snoek J, Larochelle H, Adams RP (2012) Practical bayesian optimization of machine learning algorithms. Adv Neural Inf Proc Syst 25:1
27. Souza VB, Masip-Bruin X, Marín-Tordera E, Sànchez-López S, Garcia J, Ren GJ, Jukan A, Ferrer AJ (2018) Towards a proper service placement in combined fog-to-cloud (f2c) architectures. Futur Gener Comput Syst 87:1–15
28. Stévant B, Pazat JL, Blanc A (2018) Optimizing the performance of a microservice-based application deployed on user-provided devices. In: 2018 17th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 133–140. IEEE
29. Taherizadeh S, Apostolou D, Verginadis Y, Grobelnik M, Mentzas G (2021) A semantic model for interchangeable microservices in cloud continuum computing. Information 12(1):40
30. Taherizadeh S, Stankovski V, Grobelnik M (2018) A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers. Sensors 18(9):2938
31. Taneja M, Davy A (2017) Resource aware placement of iot application modules in fog-cloud computing paradigm. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1222–1228. IEEE
32. Wang S, Guo Y, Zhang N, Yang P, Zhou A, Shen XS (2019) Delay-aware microservice coordination in mobile edge computing: a reinforcement learning approach. IEEE Trans Mobile Comput
33. Wang W, Zhao Y, Tornatore M, Gupta A, Zhang J, Mukherjee B (2017) Virtual machine placement and workload assignment for mobile edge computing. In: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), pp. 1–6. IEEE
34. Yigitoglu E, Mohamed M, Liu L, Ludwig H (2017) Foggy: a framework for continuous automated iot application deployment in fog computing. In: 2017 IEEE International Conference on AI & Mobile Services (AIMS), pp. 38–45. IEEE
35. Yin L, Luo J, Luo H (2018) Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. IEEE Trans Industr Inf 14(10):4712–4721

Authors and Affiliations

Shubha Brata Nath[1] · Subhrendu Chattopadhyay[2] · Raja Karmakar[3] · Sourav Kanti Addya[4] · Sandip Chakraborty[1] · Soumya K Ghosh[1]

Shubha Brata Nath
nath.shubha@gmail.com

Subhrendu Chattopadhyay
subhrendu@iitg.ac.in

Raja Karmakar
rkarmakar.tict@gmail.com

Sourav Kanti Addya
kanti.sourav@gmail.com

Soumya K Ghosh
skg@cse.iitkgp.ac.in

1 Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, 721302 Kharagpur, India

2 Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, 781039 Guwahati, India

3 Techno International New Town, New Town 700156, India

4 Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal 575025, India