PTC: Pick-Test-Choose to Place Containerized Micro-services in IoT

Shubha Brata Nath*, Subhrendu Chattopadhyay†, Raja Karmakar‡,
Sourav Kanti Addya§, Sandip Chakraborty¶ and Soumya K Ghosh‖
*§¶‖Indian Institute of Technology Kharagpur, India † Indian Institute of Technology Guwahati, India
‡Techno International New Town, India
Email: *nath.shubha@gmail.com, †subhrendu@iitg.ac.in, ‡rkarmakar.tict@gmail.com,
§kanti.sourav@gmail.com, ¶sandipc@cse.iitkgp.ac.in, ‖skg@cse.iitkgp.ac.in

***Abstract*—In the presence of the Internet of Things (IoT) devices, the end-users require a response within a short amount of time which the cloud computing alone cannot provide. Fog computing plays an important role in the presence of IoT devices in order to meet such delay requirements. Though beneficial in these latency-sensitive scenarios, the fog has several implementation challenges. In order to solve the problem of micro-service placement in the fog devices, we propose a framework with the objective of achieving low response time. This problem has been formulated as an optimization problem to improve the response time by considering the time-varying resource availability of the fog devices as constraints. We propose an orchestration framework named *Pick-Test-Choose* (PTC) to solve the problem. PTC uses *Bayesian Optimization based iterative reinforcement learning* algorithm to find out a micro-service allocation based on the current workload of the fog devices. PTC employs containers for service isolation and migration of the micro-services. The proposed architecture is implemented over an in-house testbed as well as in iFogSim simulator. The experimental results show that the proposed framework performs better in terms of response time compared to various other baselines.**

***Keywords*-Fog Computing; Internet of Things; Micro-service Placement; Container Placement; Reinforcement Learning**

I. Introduction

Usage of cloud infrastructure is a popular choice for the deployment of large scale distributed applications. However, the cloud is not a suitable platform for many Internet of Things (IoT) applications. Short bursty flows generated by the IoT applications increase the response time when each of the packets is processed in the cloud. To overcome this limitation, Cisco proposed *"Fog computing"* [1], which employs *"in-network processing"* to deliver user desired quality of service (QoS) for time critical IoT applications. Fog hosts small-scale cloud to support *"in-network processing"* by using the residual resources of the network equipment like the routers. Fog devices are resource constrained; therefore, instead of using monolithic *"service oriented architecture"* (SOA), the fog applications or services must adopt *"micro-service architecture"* [2], where each application is developed in the form of a bunch of loosely coupled lightweight services. Although the placement of micro-services over various fog devices is as important as the service placement in case of cloud computing, it is not very straightforward and differs from the traditional service offloading mechanisms [3], [4]. The primary challenges are as follows. (i) As the fog uses in-network processing, all the fog devices have their primary workloads (e.g., routing for a network router), and the fog resource availability varies over time. Further, the secondary workload from the micro-services should not interfere with the primary workload of the fog devices. (ii) Multiple micro-services can be placed simultaneously at the fog devices; therefore, the architecture requires micro-service isolation to ensure resource provisioning. (iii) Based on the temporal nature of the primary workload, the architecture must support seamless migration of micro-services from one fog device to another to ensure application QoS. (iv) The deployment framework and the micro-service migration framework must be lightweight so that it can cater to low latency IoT applications by deciding micro-service placement quickly. (v) The highly dynamic nature of the in-network processing architecture increases monitoring overhead, and maintaining consistency of the monitored statistics is difficult. The monitoring inconsistency leads to the performance degradation of the overall system. Ensuring these five requirements simultaneously is non-trivial over a fog environment.

A few works in the recent literature have addressed the problem of micro-service placement in the fog devices. In [5], the authors have addressed the application placement problem by placing the virtual machines (VM) in the edge or fog devices. However, the use of VM requires high resource consumption; to avoid this overhead, [6] have proposed a container (lightweight virtualization technology) driven framework to speed-up application deployment procedure. [7] and [8] have proposed the container placement and module mapping algorithm respectively in a hierarchical fog environment. DROPLET [9] provides a computation partitioning and offloading procedure. On the other hand, a micro-service offloading framework by exploiting traditional memory allocation strategies have been proposed in [10]. In Foglets [11], the authors have proposed a greedy service placement strategy for a hierarchical fog infrastructure in the presence of mobile users to minimize the access delay. For this type of mobility capable fog systems, Mobility-based [12] have provided a VM placement and migration framework to maximize the number of applications placed in fog while reducing the overall application latency. However, most of these existing works are primarily targeted towards a dedicated fog infrastructure where the fog device does not have any primary workload. So, the existing works are incapable of handling the temporal primary workload demands. Therefore, the design of a QoS ensuring micro-service deployment framework with resource constraints is non-trivial in case of fog deployment.

In this paper, we develop a lightweight orchestration framework named Pick-Test-Choose (PTC) for dynamic online micro-service placement and resource management over a fog computing infrastructure. PTC (§II) solves the above five requirements over a dynamic time-constrained environment by using containers [13]. Use of container ensures – (i) service isolation and (ii) lightweight migration of micro-services from one fog device to another as required. To solve the dynamic placement problem of micro-services over fog devices based on resource availability and workload characteristics, we formulate the placement problem as an optimization problem based on integer linear programming (ILP) (§III). However, ILP is NP-hard even with a static placement, and therefore, it is difficult to find an optimal placement when the temporal workload pattern is difficult to estimate. We consider this time-varying effect of the primary workload is the inherent environmental factor and can be learned by using *"environmental learning"* [14] primitive.

For this purpose, we use a reinforcement learning based algorithm where the framework can monitor and predict the nature of resource availability and workload characteristics over time (environmental factors), and it can dynamically decide on selecting a suitable fog device for executing a micro-service. The proposed algorithm can also readjust the placement based on the change in the environmental factors. Consequently, we use Bayesian Optimization [15], a lightweight, dynamic, iterative, and online reinforcement learning framework as a design choice for solving the micro-service deployment problem (§III-E). The proposed framework has been implemented and tested over a fog networking prototype testbed. We observe that the proposed framework can reduce the average application response time than the baselines. Also, the proposed framework provides better functionalities in terms of micro-service isolation and lightweight execution. To test the scalability of the framework, we have implemented the proposed framework over *iFogSim* [16] simulator. The experimental results have shown that our framework is effective for optimization of system response time meeting the resource constraints.

II. System Architecture

The system architecture, as shown in Figure 1, has the following components.

Client Node: Client nodes are end-user nodes. The task of a client node includes the generation of a fog service request.

Fog Controller Device: The fog controller device is responsible for managing the computation offloading in the fog devices. A fog controller is logically connected with the fog devices. Fog controller identifies the client requests and maps the request to a suitable application through `application mapper`. This application is further divided into micro-services by the `micro-service organizer` module, following the existing approaches such as ENTICE [17]. The organizer module is also responsible for the containerization of the micro-services to overcome the micro-service isolation issue. Once the micro-services are identified, it is the task of the `micro-service dispatcher` to identify the fog devices which can host the micro-services. To identify a feasible micro-service placement, the dispatcher module takes help of the `statistics collector module`. The major task of the statistics collector is to periodically monitor the available resources of the connected fog devices. The proposed framework is capable of finding out the placement of micro-services even in the presence of noises in the measurement. Once the placement schedule is generated by the dispatcher module, the micro-services are delegated to the fog devices by a `micro-service migration engine`. During the execution of the micro-services, the fog devices might get overloaded by its primary workloads. In such cases, the micro-service dispatcher and the migration engine handle the situations by the regeneration of placement schedule and migrating micro-services according to the schedule.

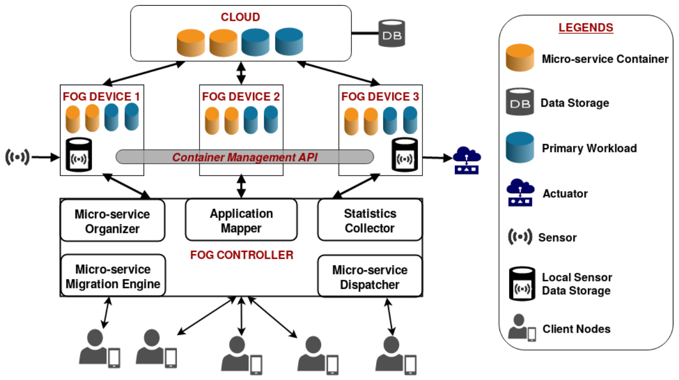

Fig. 1. In-network processing architecture with PTC framework

Fog Device: Fog devices are responsible for the actual computation and storage related to the applications provided by the fog-cloud. Fog devices use container technologies for resource isolation and management of the micro-services executing in them. General purpose fog devices are mainly equipment like networking hardware, smart meters, gateways, etc. Therefore, the fog devices have some primary workloads which cannot be affected at any cost. These primary workloads consume resources based on environmental demands. In Figure 1, we represent these primary workloads with blue colored containers. The residual capacity of the fog devices can be utilized by the fog micro-service containers, delegated by the controller management modules. Orange colored containers in the Figure 1 represent such containers. The micro-services running in the containers might require interaction with sensors and actuators. Our framework contains a dedicated `local sensor data storage` module to ease this process. During the execution of the micro-services, a container may require migration from one fog device to another to satisfy resource demand of the primary workload. To support the migration, we use a distributed `container management` application programming interface (API).

Cloud: Although the primary objective of fog is to place micro-services as near as possible to the data sources to reduce communication overhead, sometimes it is not possible to cater to all the micro-service demands inside the fog. Therefore, the proposed framework keeps the provision of using the cloud to avoid such overloaded scenarios.

III. Micro-service Placement as an Optimization

We consider an undirected and weighted communication graph $G = (V, E, W)$. Here, V represents the set of physical nodes, E is the set of physical communication links, and W is the set of edge weights. We denote $S = (S_i \in (1, \ldots, s))$, $\Lambda = (\Lambda_i \in (1, \ldots, \lambda))$, and $F = (F_i \in (1, \ldots, n))$ as the sets of sensors, actuators, and fog devices respectively. Here, s, λ, and n are the total number of sensors, actuators, and fog devices 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 delay of the links. $D(i1, i2)$ defines the weight of the shortest path between v_{i1} and v_{i2}, which signifies the communication delay between the nodes. We consider that the time is divided into l time slots where each slot is of length ι. Therefore, we define system time vector $T = (\tau_t : t \in (1, \ldots, l))$.

Let us consider that there are k applications present in the system such that $\vec{\Psi} = (P_x : x \in (1, \ldots, k))$. $\vec{\Psi}$ is the set of applications. We consider that an application P_x can further be divided into h_x atomic micro-services, and each micro-service can execute independently. We define $p_{x,y}$ as the y^{th} micro-service of P_x. Hence, $P_x = (p_{x,y} : y \in (1, \ldots, h_x))$. We denote the resource vector available in F_i as $\vec{R}(F_i)$, and $\vec{\Gamma}(p_{x,y})$ denotes the resource vector required by micro-service $p_{x,y}$. We assume that the resource vector required by $p_{x,y}$ does not change over time. $\vec{\Gamma}(p_{x,y}) = (\gamma^q_{x,y} \in \mathbb{R} : q \in (1, \ldots, f))$, where each component $\gamma^q_{x,y}$ 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 type of resources. Similarly, $\vec{R}(F_i) = (r^q_i \in \mathbb{R} : q \in (1, \ldots, f))$, where r^q_i is the total available quantity of resource type q at F_i. Each fog device F_i can host several micro-services based on the available resources. A micro-service allocation matrix $[A] = (A_{x,y,i,t} \in (0, 1) : (x \in (1, \ldots, k), y \in (1, \ldots, h_{max}), i \in (1, \ldots, n), t \in (1, \ldots, l))$ provides the mapping between micro-services and fog devices at each time quantum, 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 t, otherwise $A_{x,y,i,t} = 0$. Let $O(A, x, y)$ signify the occupancy of fog device resources by $p_{x,y}$ at each time instance as defined in Eq. (1).

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

The total response time for an application has the following components.

A. Estimation of Processing Time

Each micro-service cannot occupy more than one fog device at the same time. Therefore, the row vector $Seq_{exe}(x, y) = [I]_{1 \times n} \times [O(A, x, y)]_{n \times l}$ provides the fog device execution sequence by $p_{x,y}$, where $[I]_{1 \times n} = [1, 2, \ldots, n]$ represents row vector of the fog device indices. If F_i is capable of executing δ_i million instructions per second (MIPS), and $p_{x,y}$ requires $\xi_{x,y}$ million instructions, the processing time (T_{CPU}) required by $p_{x,y}$ can be expressed as follows.

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

B. Estimation of Migration Time

Let $M_{x,y} = (M^b_{x,y} = Seq_{exe}(x, y)_b : Seq_{exe}(x, y)_b \neq Seq_{exe}(x, y)_{b+1})$ represent the sub-vector of $Seq_{exe}(x, y)$ where b is a particular time slot. $M_{x,y}$ represents the fog device migration vector by $p_{x,y}$. Therefore, the total time required for migration (T_{MGR}) is

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

Here, $\Delta_{x,y}$ denotes a constant delay factor for migration of $p_{x,y}$, which depends upon the amount of data to be communicated from one fog device to another, and $D(v_i, v_j)$ represents the weight of the shortest path between v_i and v_j.

C. Estimation of Data Fetch and Actuation Time

Let $\psi = (\psi_{x,y,i} : x \in (1, \ldots k), y \in (1, \ldots h_{max}), i \in (1, \ldots s))$ and $\alpha = (\alpha_{x,i} : x \in (1, \ldots k), i \in (1, \ldots \lambda))$ represent the demand vector of sensors and actuators respectively by the micro-services. Here, $\psi_{x,y,i} = 1$ if $p_{x,y}$ requires S_i, otherwise 0. Similarly, $\alpha_{x,i} = 1$ if P_x requires Λ_i and $\alpha_{x,i} = 0$ for all other cases. We denote sensors and actuators required by $p_{x,y}$ as $\psi_{x,y} = [I]_{1 \times s} \times (\psi_{x,y,i} : i \in (1, \ldots s))$ and $\alpha_x = [I]_{1 \times \lambda} \times (\alpha_{x,i} : i \in (1, \ldots \lambda))$ respectively. $[I]_{1 \times s} = [1, 2, \ldots, s]$ represents the row vector of the sensor indices, and $[I]_{1 \times \lambda} = [1, 2, \ldots, \lambda]$ represents the row vector of the actuator indices. We define the data fetch time (T_{DF}) of $p_{x,y}$ as follows.

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

We define the time required to send a reply to actuators (T_{ACT}) as follows.

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

Let $\Omega(A, i, t)$ signify the executing micro-service vector at fog device F_i at time t, and it can be defined as follows.

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

From Eq. (6), at time t, we can calculate the amount of occupied resource type q as follows.

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

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 capacity property as given in Eq. (8).

$$\underset{i,q,t}{\forall}\,\Theta(A,i,q,\vec{\Gamma},t) \leq r_i^q \tag{8}$$

Therefore, the total response time required by $\vec{\Psi}_x$ can be calculated by using Eqs. (2) to (5), as follows.

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

D. Problem Definition

Given the communication graph G and available resources ($\vec{R}(F_i)$), 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 ($\vec{\Gamma}(p_{x,y})$) such that the maximum response time taken by the applications is minimized. Therefore, we have

$$\begin{aligned} &\underset{A}{\text{minimize}} \qquad T_{Resp}^{max}(A) \\ &\text{subject to:} \\ &\qquad\qquad \mathcal{R}(A) \geq \vec{Z} \end{aligned} \tag{10}$$

where $T_{Resp}^{max}(A) = \max_x (T_{Resp}(A,x))$, $\mathcal{R}(A)_{i,q,t} = \vec{R}(F_i) - \Theta(A,i,q,\vec{\Gamma},t)_{i,q,t}$, and $\vec{Z} = (z_{i,q,t} = 0 : i = (0,\ldots,n), q = (0,\ldots,f), t = (0,\ldots,l))$.

Based on the formal definition of the problem, we can prove that the *"Minimax Facility location problem"* (MFLP) [18] is polynomial time reducible to micro-service allocation problem given in Eq. (10). Thus, the micro-service allocation is $\mathcal{NP}$-hard. However, we exclude this proof due to the space constraints of this paper.

E. Solution Approach: Micro-service Placement using Bayesian Optimization

As the problem is $\mathcal{NP}$-hard and difficult to implement due to high monitoring overhead, we propose a reinforcement learning framework which requires very little monitoring and can perform in the presence of noise. For this purpose, we design *"Bayesian optimization"* (BO) based mechanism, PTC, to place micro-services over the fog devices. 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. BO performs well in such cases where performing one single experiment takes higher time.

Without loss of generality, we assume that the utility function $T_{Resp}^{max}(A)$ follows a normal distribution. Based on this prior belief function, BO executes initial experiments. After sufficient amount of preliminary experiments, BO updates the prior belief function based on the posterior distributions. To update the prior distribution, BO uses *"acquisition function"* (EI) which intelligently notifies BO to choose the configurations for subsequent experiments. This leads the framework towards optimum configuration. Let $\mathcal{D}_d = \{(a_1, T_{Resp}^{max}(a_1)), \ldots, (a_d, T_{Resp}^{max}(a_d))\}$ be the set of prior observations after d iterations. We denote $p(T_{Resp}^{max}) = \mathcal{N}(\mu, K)$. Here, the mean function is $\mu(\circ)$, and the co-variance kernel function is $K(\circ,\circ)$. These are defined as follows.

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

$$K(a_u,a_v) = \mathbb{E}\left(\left(T_{Resp}^{max}(a_u) - \mu(a_u)\right)\left(T_{Resp}^{max}(a_v) - \mu(a_v)\right)\right) \tag{12}$$

Let us denote Φ and ϕ as the standard normal cumulative distribution function and the standard normal density function respectively. We define $U_{min} = \underset{a\in\mathcal{D}_d}{min}(T_{Resp}^{max}(a))$, $\pi = \frac{U_{min}-\mu(a_d)}{\sigma(a_{d-1},a_d)}$, and $\sigma(a_{d-1},a_d)=\sqrt{K(a_{d-1},a_d)}$. We choose our acquisition function as described in Eq. (13) as recommended in [19].

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

However, $EI(A|\mathcal{D}_d)$ works well in case of unconstrained optimization. Therefore, to satisfy our constrained optimization, we adopt the procedure suggested by Gardner et al. [20]. By following their footsteps, we assume that $\mathcal{R}(\mathcal{A})$ follows Bernoulli process, and EI can be modified to Eq. (14).

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

In our set-up, there can be observation noise due to various factors such as a rise in loads in the fog devices, network delays due to channel states, the rise in reporting delays due to various circumstance, etc. Therefore, the proposed BO algorithm must withstand observation noise. It is a standard practice to assume observation noise (ζ) as a normally distributed random variable with zero mean i.e., $\zeta = \mathcal{N}(0, \sigma_\zeta)$.

IV. Evaluation

The proposed framework is implemented both in a testbed and over a simulation environment. The testbed gives results under a realistic setup, whereas the simulation has helped us to understand the scalability of PTC orchestration framework.

A. Implementation Details

The testbed is implemented using `Raspberry Pi 3 model b` (https://www.raspberrypi.org/products/raspberry-pi-3-model-b/) single-board computers as the fog devices, connected over a local network. Raspberry Pi 3 model b is equipped with quad core 1.2 GHz Broadcom BCM2837 64 bit central processing unit (CPU) and 1 GB random access memory (RAM). Therefore, these devices have low processing and storage capability. Our deployed fog

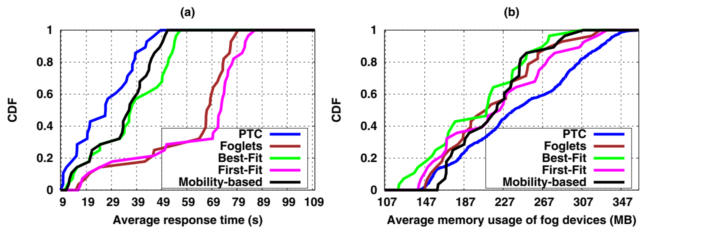

Fig. 2. (a) CDF of average response time and (b) CDF of average memory usage of the fog devices

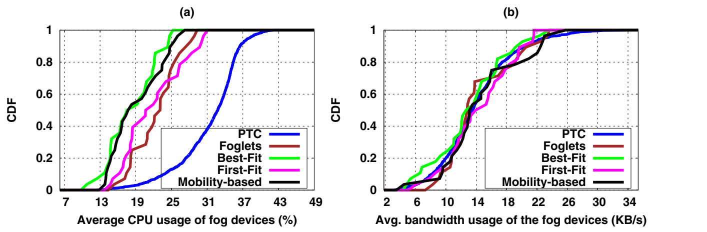

Fig. 3. (a) CDF of average CPU usage of the fog devices and (b) CDF of average network bandwidth usage of the fog devices

devices use `Ubuntu 16.04.2 long-term support (LTS)` (https://www.ubuntu.com/) operating system. In the fog devices, we use `Docker` (https://www.docker.com/) to achieve micro-service isolation. Docker provides a platform for lightweight containers by application layer virtualization. The framework is also connected with a private cloud environment available in our institute.

B. Results

Figure 2(a) illustrates the performance of PTC in terms of average response time of the applications. We observe that the average response time is significantly low in PTC compared to other schemes; therefore, the cumulative distribution function (CDF) of PTC converges within 50 seconds. On the other hand, CDFs of other competing mechanisms converge between 53 seconds and 88 seconds. Through deep dive into the performance logs, we observe that BO in PTC converges quickly to an optimal or near-optimal allocation vector (within 30 iterations). BO has a tendency to find the desired points in the search space with relatively few function evaluations. With the help of this optimization technique, adaptive learning is induced in PTC, and thus, PTC becomes an intelligent scheme. In this regard, BO-based adaptive learning mechanism helps PTC to optimize the utility function with the help of prior observations and posterior distribution such that the overall response time can be reduced. As a result, the framework learns the fog environment and can take action accordingly to provide a minimum average response time for running the applications.

The CDF of the average memory usage of the fog devices is shown in Figure 2(b). The PTC has more average memory usage than the baseline schemes. The density of such distribution is higher (in the range of 139 MB – 366 MB) in PTC than that of the baselines. Whereas, the baselines converge within 334 MB for first-fit, within 327 MB for foglets, within 310 MB for mobility-based, and within 308 MB for best-fit respectively. The CDF of average CPU usage of the fog devices is shown in Figure 3(a). It is higher in PTC, and it is in the range of 12% – 44%. The baselines converge within 27% for best-fit, within 33% for first-fit, within 31% for foglets, and within 29% for mobility-based algorithm respectively. The CDF of bandwidth usage of the fog devices is shown in Figure 3(b). It is higher in PTC. It is in the range of 3 KB/s – 36 KB/s. Whereas, such distributions converge within 25 KB/s for the

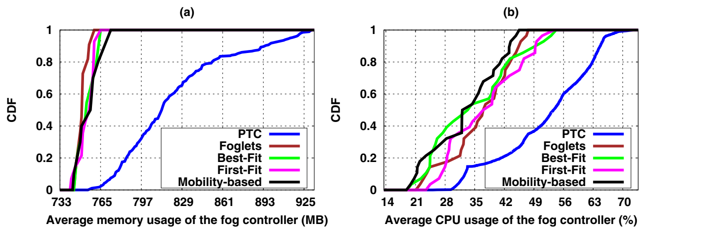

Fig. 4. (a) CDF of average memory usage of the fog controller device and (b) CDF of average CPU usage of the fog controller device

best-fit and foglets mechanisms. It converges within 23 KB/s for the first-fit mechanism and within 27 KB/s for the mobility-based algorithm.

Next, we look into the average resource usage by the fog controller which is responsible for running the BO based placement algorithm. The CDF of the average memory usage by the fog controller is shown in Figure 4(a). PTC has more average memory usage for the controller device due to the overhead of multiple iterations. The density of average memory usage distribution of controller is higher (in the range of 756 MB – 933 MB) in PTC than that of the baselines. The average memory usage distribution of the fog controller converges within 760 MB for foglets, within 765 MB for best-fit, within 766 MB for first-fit, and within 774 MB for mobility-based algorithm. The CDF of average CPU usage of the controller device is shown in Figure 4(b). It is higher in PTC, and it is in the range of 26% – 74%. The baselines converge within 56% for best-fit, within 55% for first-fit, within 49% for foglets, and within 47% for mobility-based algorithm respectively. The CDF of the average bandwidth distribution for PTC controller is shown in Figure 5(a). It is higher (in the range of 11 KB/s – 291 KB/s). The baselines converge within 74 KB/s for best-fit, within 68 KB/s for first-fit, within 69 KB/s for foglets, and within 69 KB/s for the mobility-based mechanism.

PTC uses container migration for balancing the load across the fog devices with an objective towards minimizing the average response time of the application. We have shown the average number of migrations of the micro-services in Figure 5(b). PTC migrates the micro-services in a new fog device only when there is a better fog device available in terms of resource availability. Interestingly, we observe that though the total number of migrations increases, the average number

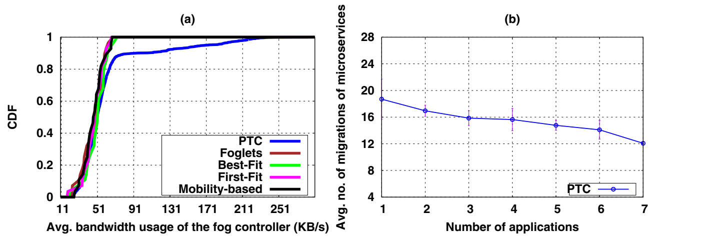

Fig. 5. (a) CDF of average network bandwidth usage of the fog controller device and (b) Avg. number of migrations of the micro-services

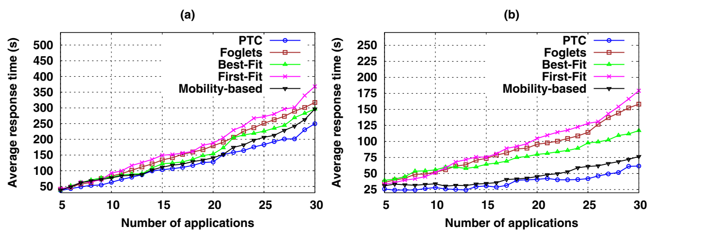

Fig. 6. Average response time: (a) number of fog devices=20, and (b) number of fog devices=40

of migrations decreases with the increase in the number of applications.

We have also implemented PTC in iFogSim [16] in order to check the scalability of the proposed framework. The simulation is performed with an increased numbers of fog devices.

Figure 6 shows the average response times under different number of fog devices. In order to optimally place the micro-services in fog devices, we need an online learning which can select the available fog devices dynamically by considering the load distribution as well as the resource availability of the devices. In addition, an optimization technique is required to place the micro-services in fog devices in real-time and migrate these services in proper fog devices such that the average response time of the applications gets reduced. In this regard, BO is an iterative reinforcement learning approach which tries to optimize its utility function based on the knowledge gained through the execution of the algorithm. This adaptive learning based optimization helps PTC to utilize the available fog devices as much as possible, such that the maximum number of applications can be placed parallelly. As a result, the average response time is reduced significantly in PTC, as the number of available fog devices increases.

V. Conclusion

In this paper, we have proposed a BO-based iterative reinforcement learning mechanism for containerized micro-service placement in IoT considering the time-varying resource availability based on the fog device workloads. Based on the primary workload of the fog devices, there is a need to select the correct set of devices for micro-service placement in order to minimize the response time of the applications. In order to achieve this requirement, we have formulated the micro-service assignment problem as an optimization problem which is solved using a Bayesian Optimization based iterative reinforcement learning algorithm. We have implemented PTC in an in-house testbed setup as well as in *iFogSim* simulator, and it is observed that PTC can minimize the response time of the system. In the future, we plan to evaluate the performance of PTC under different applications.

References

[1] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big data and internet of things: A roadmap for smart environments*. Springer, 2014, pp. 169–186.

[2] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap," in *Proc. of the 13th IEEE SCC*. IEEE, 2016, pp. 813–818.

[3] G. Orsini, D. Bade, and W. Lamersdorf, "Computing at the mobile edge: Designing elastic android applications for computation offloading," in *Proc. of the 8th IFIP WMNC*. IFIP, 2015, pp. 112–119.

[4] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278–289, 2018.

[5] W. Wang, Y. Zhao, M. Tornatore, A. Gupta, J. Zhang, and B. Mukherjee, "Virtual machine placement and workload assignment for mobile edge computing," in *Proc. of the 6th IEEE CloudNet*. IEEE, 2017, pp. 1–6.

[6] A. Ahmed and G. Pierre, "Docker Container Deployment in Fog Computing Infrastructures," in *Proc. of the IEEE EDGE*. IEEE, July 2018, pp. 1–8.

[7] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, "Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing," in *Proc. of the 6th IEEE AIMS*. IEEE, 2017, pp. 38–45.

[8] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm," in *Proc. of the 15th IFIP/IEEE IM*. IFIP/IEEE, 2017, pp. 1222–1228.

[9] T. Elgamal, A. Sandur, P. Nguyen, K. Nahrstedt, and G. Agha, "DROPLET: Distributed Operator Placement for IoT Applications Spanning Edge and Cloud Resources," in *Proc. of the 11th IEEE CLOUD*. IEEE, July 2018, pp. 1–8.

[10] V. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, J. Garcia, G.-J. Ren, A. Jukan, and A. J. Ferrer, "Towards a proper service placement in combined Fog-to-Cloud (F2C) architectures," *Future Generation Computer Systems*, vol. 87, pp. 1–15, 2018.

[11] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proc. of the 10th ACM DEBS*. ACM, 2016, pp. 258–269.

[12] D. Gonçalves, K. Velasquez, M. Curado, L. Bittencourt, and E. Madeira, "Proactive virtual machine migration in fog environments," in *Proc. of the 23rd IEEE ISCC*. IEEE, 2018, pp. 00 742–00 745.

[13] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[14] S. Gough and W. Scott, *Sustainable Development and Learning: Framing the Issues*. Routledge, 2003.

[15] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[16] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments," *Weily Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[17] G. Kecskemeti, A. C. Marosi, and A. Kertesz, "The ENTICE approach to decompose monolithic services into microservices," in *Proc. of the 14th IEEE HPCS*. IEEE, 2016, pp. 591–596.

[18] Z. Drezner and G. O. Wesolowsky, "Minimax and maximin facility location problems on a sphere," *Naval Research Logistics Quarterly*, vol. 30, no. 2, pp. 305–312, 1983.

[19] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *Proc. of the 14th USENIX NSDI*. USENIX, 2017, pp. 469–482.

[20] J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian Optimization with Inequality Constraints," in *Proc. of the 31st ICML*, 2014, pp. 937–945.