COSCO: Container Orchestration Using Co-Simulation and Gradient Based Optimization for Fog Computing Environments

Shreshth Tuli, Shivananda R. Poojara, Member, IEEE, Satish N. Srirama, Senior Member, IEEE, Giuliano Casale, and Nicholas R. Jennings

Abstract—Intelligent task placement and management of tasks in large-scale fog platforms is challenging due to the highly volatile nature of modern workload applications and sensitive user requirements of low energy consumption and response time. Container orchestration platforms have emerged to alleviate this problem with prior art either using heuristics to quickly reach scheduling decisions or AI driven methods like reinforcement learning and evolutionary approaches to adapt to dynamic scenarios. The former often fail to quickly adapt in highly dynamic environments, whereas the latter have run-times that are slow enough to negatively impact response time. Therefore, there is a need for scheduling policies that are both reactive to work efficiently in volatile environments and have low scheduling overheads. To achieve this, we propose a Gradient Based Optimization Strategy using Back-propagation of gradients with respect to Input (GOBI). Further, we leverage the accuracy of predictive digital-twin models and simulation capabilities by developing a Coupled Simulation and Container Orchestration Framework (COSCO). Using this, we create a hybrid simulation driven decision approach, GOBI*, to optimize Quality of Service (QoS) parameters. Co-simulation and the back-propagation approaches allow these methods to adapt quickly in volatile environments. Experiments conducted using real-world data on fog applications using the GOBI and GOBI* methods, show a significant improvement in terms of energy consumption, response time, Service Level Objective and scheduling time by up to 15, 40, 4, and 82 percent respectively when compared to the state-of-the-art algorithms.

Index Terms—Fog computing, coupled simulation, container orchestration, back-propagation to input, QoS optimization

1 INTRODUCTION

Fog computing is an emerging paradigm in distributed systems, which encompasses all intermediate devices between the Internet of Things (IoT) layer (geo-distributed sensors and actuators) and the cloud layer (remote cloud platforms). It can reduce latency of compute, network and storage services by placing them closer to end users leading to a multitude of benefits. However, fog environments pose several challenges when integrating with real-world applications. For instance, many applications, especially in healthcare, robotics and smart-cities demand ultra low response times specifically for applications sensitive to Service Level Objectives (SLO) [1]. Other applications involving energy scavenging edge devices and renewable resources need supreme energy efficiency in task execution [2]. The challenge of reaching low response times and energy consumption is further complicated by modern-day application workloads being highly dynamic [3] and host machines having non-stationary resource capabilities [4].

To provide quick and energy efficient solutions, many prior works focus on developing intelligent policies to schedule compute tasks on fog hosts [5], [6], [7]. These methods have been dominated by heuristic techniques [7], [8], [9], [10]. Such approaches have low scheduling times and work well for general cases, but due to steady-state or stationarity assumptions, they provide poor performance in non-stationary heterogeneous environments with dynamic workloads [9]. To address this, some prior approaches use more intelligent and adaptive schemes based on evolutionary methods and reinforcement learning. These methods adapt to changing scenarios and offer promising avenues for dynamic optimization [11]. However, they too are unable to efficiently manage volatile fog environments because of their poor modelling accuracies and low scalability [3], [5]. For accurate and scalable modelling of the fog environment, there have been many works that use deep learning based local search or learning models with neural networks which approximate an objective function such as energy consumption or response time [3], [5], [12]. As these neural networks approximate objective functions of optimization problem, they are often referred to as “neural approximators” [13], [14]. Specifically, many recent state-of-the-art techniques primarily use optimization methods like genetic algorithms (GA) [12] and policy-gradient learning [3].
to optimize QoS parameters, thanks to their generality. However, gradient-free methods like GA are slow to converge to optima due to undirected search schemes [15]. Further, policy-gradient learning takes time to adapt to sudden changes in the environment and shares the same problem of high scheduling overheads. Such high scheduling times limit the extent of the possible improvement of latency and subsequently SLO violations. This is not suitable for highly volatile environments where host and workload characteristics may suddenly and erratically change. Thus, there is a need for an approach which not only can adapt quickly in volatile environments but also have low scheduling overheads to efficiently handle modern workload demands.

To solve this problem, a natural choice is to use directed schemes like A* search or gradient-based optimization strategies. Even though such strategies have been shown to converge much faster than gradient-free approaches, prior work does not use them due to the highly non-linear search spaces in real-world problems which can cause such methods to get stuck in local optima [16]. Moreover, prior works fail to leverage recent advances like root-mean-square propagation, and momentum and annealed gradient descent with restarts that help prevent the local optima problem [17], [18], [19]. Given this, we believe that by leveraging the strength of neural networks to accurately approximate QoS parameters [20], we can apply gradient-based algorithms in tandem with advances that reduce the likelihood of such methods getting stuck at local optima. Prior work also establishes that neural approximators are able to precisely model the gradients of the objective function with respect to input using back-propagation allowing us to use them in gradient-based methods for quick convergence [21]. When taken together, this suite of approaches should provide a quick and efficient optimization method. In particular, we formulate a gradient-based optimization algorithm (GOBI) which calculates the gradients of neural networks with respect to input for optimizing QoS parameters using advanced gradient-based optimization strategies. We establish by experiments that our approach provides a faster and more scalable optimization strategy for fog scheduling compared to the state-of-the-art methods.

However, simply using gradient-based optimization is not sufficient as data driven neural models can sometimes saturate [22]. This is when feeding more data to the neural model does not improve the performance. Further optimization of QoS in such cases is hard and more intelligent schemes are required. Coupled-simulation (also referred as co-simulation or symbiotic simulation) and execution of tasks have been shown to be a promising approach for quickly obtaining an estimate of QoS parameters in the near future [23], [24], [25]. Specifically, coupled-simulation allows us to run a simulator in the background with the scheduling algorithms to facilitate decision making. However, prior works use this to aid search methods and not to generate more data to facilitate decision making of an AI model. The latter requires development of a new interface between the scheduler and simulator. Hence, we develop COSCO: Coupled Simulation and Container Orchestration Framework to leverage simulated results to yield better QoS. The COSCO framework is the first to allow a single or multi-step simulation of container migration decisions in fog environments. It enables a scheduler to get an estimate of the QoS parameters at the end of a future interval for better prediction and subsequently optimum schedules. Container migration refers to the process of moving an application between different physical or virtual hosts and resuming computation on the target host. This allows us to run the GOBI approach, simulate the schedule (with a predicted workload model) and provide the objective values to another neural approximator which can then better approximate the objective function and lead to improved performance. We call this novel optimization loop GOBI* (Fig. 1). The interactive training dynamic between GOBI and GOBI* aides the latter to converge quickly and adapt to volatile environments (more details in Section 7).

In summary, the key contributions of this paper are:

- We present a novel framework, COSCO, which is the first to allow coupled simulation and container orchestration in fog environments.
- We propose a gradient based-back-propagation approach (GOBI) for fast and scalable scheduling and show that it outperforms the state-of-the-art schedulers.
- We also propose an extended approach (GOBI*) that leverages simulation results from COSCO running GOBI’s decision, to provide improved predictions and scheduling decisions with lower scheduling overheads.

Validating both GOBI and GOBI* on physical setups with real-world benchmark data shows that, GOBI gives lower performance than GOBI*. However, GOBI is more suitable for resource constrained fog brokers, because of its low computational requirements. GOBI*, on the other hand, is more suitable for setups with critical QoS requirements and powerful fog brokers.

The rest of the paper is organized as follows. Related work is overviewed in Section 2. Section 3 provides a motivating example for the problem. Section 4 describes the system model and formulates the problem specifications. Section 5 details GOBI and Section 6 introduces the COSCO framework. Section 7 presents the GOBI* approach, integrating GOBI with COSCO’s co-simulation feature. A performance evaluation of the proposed methods is shown in Section 8. Finally, Section 9 concludes.

2 RELATED WORK

We now analyze prior work in more detail (see Table 1 for an overview).

Heuristic Methods. Several studies [9], [26], [27] have shown how heuristic based approaches perform well in a variety of scenarios for optimization of diverse QoS
parameters. Methods like LR-MMT schedules workloads dynamically based on local regression (LR) to predict which hosts might get overloaded in the near future and select containers/tasks which can be migrated in the minimum migration time (MMT). Both LR and MMT heuristics are used for overload detection and task selection, respectively [9], [26]. Similarly, MAD-MC calculates the median-attribute deviation (MAD) of the CPU utilization of the hosts and select containers based on the maximum correlation (MC) with other tasks [9]. However, such heuristic based approaches fail to model stochastic environments with dynamic workloads [3]. This is corroborated by our results in Section 8. To overcome this, GOBI and GOBI* are able to adapt to dynamic scenarios by constantly learning the mapping of scheduling decisions with objective values. This makes them both robust to diverse, heterogeneous and dynamic scenarios.

**Evolutionary Models.** Other prior works have shown that evolutionary based methods, and generally gradient-free approaches, perform well in dynamic scenarios [6], [28], [29], [30]. Evolutionary approaches like genetic algorithms (GA) lie in the domain of gradient-free optimization methods. The GA method schedules workloads using a neural model to approximate the objective value (as in GOBI) and a genetic-algorithm to reach the optimal decision [6]. However, gradient-free methods have a number of drawbacks. They take much longer to converge [16] and are not as scalable [15] as gradient-based methods. Moreover, non-local jumps can lead to significant change in the scheduling decision which can lead to a high number of migrations (as seen in Figs. 8e and 9d).

**MaxWeight Based Schedulers.** Over the years, MaxWeight scheduling has gained popularity for its theoretical guarantees and ability to reduce resource contention [31], [32]. However, MaxWeight policies can exhibit poor delay performance, instability in dynamic workloads and spatial inefficiency [37], [38], [39]. We use the pessimistic-optimistic online dispatch approach, POND by Liu et al. [31] which is a variant of the MaxWeight approach [40]. POND uses constrained online dispatch with unknown arrival and reward distributions. It uses virtual queues for each host to track violation counts and an Upper-Confidence Bound (UCB) policy [41] to update the expected rewards of each allocation from the reward signal. The final decision is made using the MaxWeight approach with the weights as the expected reward values. To minimize objective value in our setup, we provide the objective score as negative reward. Due to the inability of MaxWeight approaches to adapt to volatile scenarios, their wait times are high.

**Reinforcement Learning Models.** Recently, reinforcement learning based methods have shown themselves to be robust and versatile to diverse workload characteristics and complex edge setups [3], [5], [34], [35]. Such methods use a Markovian assumption of state which is the scheduling decision at each interval. Based on new observations of reward signal (negative objective score in our setting) they explore or exploit their knowledge of the state-space to converge to an optimal decision. A recent method, DQLCM, models the container migration problem as a multi-dimensional Markov Decision Process (MDP) and uses a deep-reinforcement learning strategy, namely deep Q-Learning to schedule workloads in a heterogeneous fog computing environment [35]. The state-of-the-art method, A3C, schedules workloads using a policy gradient based reinforcement learning strategy which tries to optimize an actor-critic pair of agents [3]. This approach uses Residual Recurrent Neural Networks to predict the expected reward for each action i.e., scheduling decision and tries to optimize the cumulative reward signal. However, such methods are still slow to adapt to real-world application scenarios as discussed in Section 8. This leads to higher wait times and subsequently high response times and SLO violations. Moreover, these methods do not scale well [42].

Finally, coupled or symbiotic simulation and model based control have long been used in the modelling and optimization of distributed systems [25], [43], [44]. Many prior works have used hybrid simulation models to optimize decision making in dynamic systems. To achieve this, they monitor, analyze, plan and execute decisions using previous knowledge-base corpora (MAPE-k) [1]. However, such works use this to facilitate search methods and not to generate additional data to aid the decision making of an AI model. COSCO is developed to leverage a seamless interface between the orchestration framework and simulation engine to have a interactive dynamic between AI models to optimize QoS.

For our experiments, we use LR-MMT, MAD-MC, POND, GA, DQLCM and A3C as baselines to compare against the proposed GOBI and GOBI* models. We try to cover the best approach(es) in each category by careful literature review. The different categories have been selected for their complementary benefits. Heuristic based approaches are fast but not as accurate as other methods. Learning methods are adaptive but slow to adapt or output a scheduling decision.

### 3 Motivating Example
As demonstrated by many prior works [3], [6], [9], for fog datacenters, the scheduling time increases exponentially with the number of fog devices. This can have a significant impact on response time and, correspondingly, on Service
Level Objectives (SLOs) corresponding to the task response time. To test this, we perform a series of experiments on the iFogSim simulator [45]. We test the popular heuristic based approach LR-MMT [9] and the evolutionary approach using genetic algorithm (GA) [6] with 50 simulated host machines (details in Section 8.1). The former represents the broad range of heuristic based approaches that can quickly generate a scheduling decision but lack the ability to adapt. The latter have the ability to quickly adapt but have high scheduling overheads. As in [6], we provide the GA the utilization metrics of all hosts and tasks and aim to minimize the average response time.

We assume a central scheduler which periodically allocates new tasks and migrates active tasks if required to reduce SLO violations. We assume that the container allocation time is negligible and the total response time of a fog task is the sum of the scheduling and execution times. We consider volatile workloads that are generated from traces of real-world applications running in a fog environment [3] (using the Bitbrain dataset, details in Section 8). For the approaches considered, the scheduling time can range from 100 to 150 seconds with SLO violation rate of up to 24 percent. However, if we could bring down the scheduling times to 10 seconds with the same decisions as GA (referred to as GA2), the SLO violation rate goes down to ~2%. This shows how scheduling time is a key factor that limits the extent to which SLO can be optimized (Fig. 2), thus highlighting the potential for an approach which can respond quickly in volatile scenarios and has low scheduling overheads, leveraging the fast and scalable execution times of AI methods.

4 SYSTEM MODEL AND PROBLEM FORMULATION

4.1 System Model

We consider a standard distributed and heterogeneous fog computing environment as shown in Fig. 3. We assume a single fog datacenter with geographically distributed edge and cloud layers as computational nodes. We consider network latency effects for interactions between compute devices in different layers (edge and cloud) and between compute devices and fog broker. We ignore communication latencies among devices in the same layer. Tasks are container instances with inputs being generated by sensors and other IoT devices and results received by actuators which physically manifest the intended actions. These devices constitute the IoT layer and send and receive all data from the fog gateway devices. All management of tasks and hosts is done by a Fog Broker in the management layer. This includes task scheduling, data management, resource monitoring and container orchestration. The work described in this paper is concerned with improving the scheduler in the

Fig. 2. Motivating example.

Fog Broker by providing an interface between the container orchestration and resource monitoring services with a simulator. Our work proposes discrete-time controllers, that are commonly adopted in the literature [3], [5], [43]. The communication between end-users and the fog broker is facilitated by gateway devices. The compute nodes in the fog resource layer, which we refer to as “hosts”, can have diverse computational capabilities. The hosts at the edge of the network are resource-constrained and have low communication latency with the broker and gateway devices. On the other hand, cloud hosts are several hops from the users and have much higher communication latency and are computationally more powerful. We assume that there are a fixed number of host machines in the fog resource layer and denote them as \( H = \{ h_0, h_1, \ldots, h_{N-1} \} \). We denote the collection of time-series utilization metrics, which include CPU, RAM, Disk and Network Bandwidth usage of host \( h_i \) as \( U(h_i) \). The collection of maximum capacities of CPU, RAM, Disk and Network Bandwidth with the average communication latency of host \( h_i \) is denoted as \( C(h_i) \).

4.2 Workload Model

We now present our workload model, as summarized in Fig. 4. We divide a bounded timeline into equal sized scheduling intervals with equal duration \( \Delta \). The \( t \)th interval is denoted by \( I_t \), and starts at \( s(I_t) \), so that \( s(I_0) = 0 \) and \( s(I_t) = s(I_{t-1}) + \Delta, t > 0 \). In interval \( I_{t-1} \), \( N_t \) new tasks are created by the IoT devices. The gateway devices then send batches of these new tasks \( N_t \) with their SLO requirements to the Fog Broker. The SLO violations are measured corresponding to the response time metrics of task executions as a fraction of them that exceeds the stipulated deadlines. The set of active tasks in the interval \( I_t \) is denoted as \( A_t = \{ a_{i_0}, a_{j_1}, \ldots, a_{j_N} \} \) and consists of \( |A_t| \) tasks. Here \( a_{j_N} \) represents an identifier for the \( j \)th task in \( A_t \). The Fog Broker schedules the new tasks to the compute nodes and decides which of the active tasks need to be migrated. At the end of interval \( I_{t-1} \), the set of completed tasks is denoted as \( L_t \), hence the tasks that carry forward to the next interval is the set \( A_{t-1} \setminus L_t \). Thus, the broker takes a decision \( D^t = D(Y_t) \),
where \( Y_i \) is the union of the new \((N_i)\), active \((A_{i-1}\setminus L_i)\) and waiting tasks \((W_{i-1})\). This decision is a set of allocations and migrations in the form of ordered-pairs of tasks and hosts (assuming no locality constraints). At the first scheduling interval, as there are no active or waiting tasks, the model only takes allocation decision for new tasks \((N_0)\). Here, \( D \) denotes the scheduler such that \( D : Y_i \mapsto Y_i \times H \). Only if the allocation is feasible, i.e., depending on whether the target host can accommodate the scheduled task or not, is the migration/allocation decision executed. Initially, the waiting queue and the set of active and leaving actions are empty. The executed decision is denoted by \( D \) and satisfies the following properties

\[
\hat{D}(A_i) \subseteq D(Y_i), \quad \forall (a_i', h_i) \in \hat{D}(A_i), \quad U(a_i') + U(h_i') \leq C(h_i).
\]

We use the notation \( a_i' \in \hat{D}(A_i) \) to mean that this task was allocated/migrated. The set of new and waiting tasks \( \hat{N}_i \subseteq \hat{D}(A_i) \), where \( \hat{N}_i \subseteq N_i \) and \( W_{i-1} \subseteq W_{i-1} \), denotes those tasks which could be allocated successfully. Those new tasks, i.e., \( N_i \setminus \hat{N}_i \) are added to get the new wait queue \( W_i \). Similarly, active tasks are denoted as \( A_{i-1} \subseteq A_{i-1} \). Hence, for every interval \( I_t \),

\[
A_t \leftarrow \hat{N}_i \cup W_{i-1} \cup A_{i-1} \setminus L_i
\]

\[
W_i \leftarrow (W_{i-1} \setminus \hat{W}_{i-1}) \cup (N_i \setminus \hat{N}_i), \quad W_{-1} = A_{-1} = L_{-1} = \emptyset.
\]

We also denote the utilization metrics of an active task \( a_i' \in A_i \) as \( U(a_i') \) for the interval \( I_t \). We denote a simulator as \( S : \hat{D}(A_i) \times \{U(a_i')|a_i' \in A_i\} \mapsto \{U(h_i')|h_i \in H\} \times P_i \) which takes scheduling decision and container utilization characteristics to give host characteristics and values of QoS parameters. Here \( P_i \in P \) is a set of QoS parameters such as energy consumption, response times, SLO violations, etc. Similarly, execution on a physical fog framework is denoted by \( F \). Hence, execution of interval \( I_t \) on a physical setup is denoted as

\[
\{(U(h_i')|h_i \in H), P_t\} \equiv F(\hat{D}(A_i), \{U(a_i')|a_i \in A_i\}).
\]

All symbols and their meanings are stated in Table 2.

### 4.3 Problem Formulation

After execution of tasks \( A_i \) in interval \( I_t \), the objective value to minimize is denoted as \( O(P_t) \). \( O(P_t) \) could be a scalar value which combines multiple QoS parameters.

5. **The GOBI Scheduler**

As discussed in Section 1, we now present the approach based on Gradient Based Optimization using back-propagation to Input (GOBI). This constitutes the Scheduler module of the Fog Broker described in Fig. 3. Here, we consider that we optimize an objective function \( O(P_t) \) at every interval \( I_t \) via taking an optimal action in the form of scheduling decision \( D \). This \( D \) is then used by the framework/simulator to execute tasks as described in optimization program (7).

5.1 Objective Function

We will now present how our optimization scheme can be used for taking appropriate scheduling decisions in a fog environment. To optimize the QoS parameters, we consider an objective function that focuses on energy consumption and response time, two of the most crucial metrics for fog environments [46], [47]. For interval \( I_t \),

\[
O(P_t) = \alpha \cdot AEC_t + \beta \cdot ART_t.
\]

Here \( AEC \) and \( ART \) (\( \in P_t \)) are defined as follows.

(1) **Average Energy Consumption (AEC)** is defined for any interval as the energy consumption of the infrastructure (which includes all edge and cloud
hosts) normalized by the maximum power of the hosts, i.e.,

$$AEC_t = \frac{\sum_{h \in H} \int_{t_{i-1}(t)}^{t_{i}(t)} Power_{h}(t) \, dt}{|A_t| \sum_{h \in H} Power_{h_{\max}} \times (t_{i+1} - t_{i})} \quad (9)$$

where $Power_{h}(t)$ is the power function of host $h_i$ at instant $t$, and $Power_{h_{\max}}$ is maximum possible power of $h_i$.

(2) **Average Response Time** (ART)$^1$ is defined for an interval $I_t$ as the average response time for all leaving tasks ($L_t$), normalized by maximum response time until the current interval, as shown below

$$ART_t = \frac{\sum_{j \in L_t} Response\ Time(j)}{|L_t| \max_{1 \leq j \leq |L_t|} Response\ Time(j)} \quad (10)$$

### 5.2 Input Parameters

In concrete implementations, we can assume a finite maximum number of active tasks, with the upper bound denoted as $M$. Thus at any interval, $|A_{i}| \leq M$. Moreover, we consider the utilization metrics of Instructions per second (IPS), RAM, Disk and Bandwidth consumption which form a feature vector of size $F$. Then, we express task utilization metrics $\{\{a_{i}^{-1}\}, a_{j}^{-1} \in A_{t-1}\}$ as an $M \times F$ matrix denoted as $\phi(A_{t-1})$, where the first $A_{t-1}$ rows are the feature vectors of tasks in $A_{t-1}$ in the order of increasing creation intervals (breaking ties uniformly at random), and the rest of the rows are 0. We also form feature vectors of hosts with IPS, RAM, Disk and Bandwidth utilization with capacities and communication latency of each host, each of size $F'$. Thus, at each interval $I_t$ for $|H|$ hosts, we form a $|H| \times F'$ matrix $\phi(H_{t-1})$ using host utilization metrics of interval $I_{t-1}$.

Finally, as we have $N_t$, $W_{t-1}$, $A_{t-1}$ and $L_t$ at the start of $I_t$, the decision matrix $\phi(D)$ is an $M \times |H|$ matrix with the first $|N_t| \cup W_{t-1} \cup A_{t-1} \cup L_t$ rows being one-hot vector of allocation of tasks to one of the $H$ hosts. The remaining rows being 0.

### 5.3 Model Training

We now describe how a neural model can be trained to approximate $O(P_t)$ using the input parameters $\{\phi(A_{t-1}), \phi(H_{t-1}), \phi(D)\}$. Consider a continuous function $f(x; \theta)$ as a neural approximator of $O(P_t)$ with the $\theta$ vector denoting the underpinning neural network parameters and $x$ as a continuous or discrete variable with a bounded domain. Here, $x$ is the collection of utilization metrics of tasks and hosts with a scheduling decision. The parameters $\theta$ are learnt using the dataset $\Lambda = \{\phi(A_{t-1}), \phi(H_{t-1}), \phi(D), O(P_t)\}$, such that a given loss function $L$ is minimized for this dataset. We form this dataset by running a random scheduler and saving the traces generated (more details about the setup in Section 8). The loss $L$ quantifies the dissimilarity between the predicted output and the ground truth. We use Mean Square Error (MSE) as the loss function as done in prior work [3]. Hence,

$$L(f(x; \theta), y) = \frac{1}{T} \sum_{t=0}^{T} (y - f(x; \theta))^2, \text{ where } (x, y) \in \Lambda.$$ 

1. Both AEC and ART are unit-less metrics and lie between 0 and 1.

Thus, for datapoints $(x, y) \in \Lambda$, where $y$ is the value of $O(P_t)$ for the corresponding $x$, we have $\theta = \arg\min_{\theta} \sum_{(x, y) \in \Lambda} [L(f(x; \theta), y)]$. To do this, we calculate the gradient of the loss function with respect to $\theta$ as $\nabla_{\theta} L$ and use back-propagation to learn the network parameters. By the universal approximation theorem [20], we can assume that we already have parameters $\theta$ after network training, such that this neural network approximates a generic function to an arbitrary degree of precision (considering a sufficiently large parameter set $\theta$). A key part of this ability of neural networks rests on randomly initializing a parameterized function and changing the large number of parameters based on the loss function.

Now, with this, we need to find the appropriate decision matrix $\phi(D)$, such that $f(x; \theta)$ is minimized. We formulate the resulting optimization problem as

$$\text{minimize } f(x; \theta)$$

subject to each element of $\phi(D)$ is bounded $\forall t, Eqs. (1)–(6)$.

This is a reformulation of the program (7) using the neural approximator of $O(P_t)$ as the objective function. Note that, because there is a bounded space of inputs, there must be an optimal solution, i.e., $\exists \phi(D)$ such that

$$f(\{\phi(A_{t-1}), \phi(H_{t-1}), \phi(D)\}; \theta) \leq f(\{\phi(A_{t-1}), \phi(H_{t-1}), \phi(D)\}; \theta), \forall \text{ feasible } \phi(D).$$

We solve this optimization problem using gradient-based methods, for which we need $\nabla_x f(x; \theta)$. This is uncommon for neural network training since prior work normally modifies the weights using input and ground truth values. Thus the gradients can be calculated without having to update the weights. Now, for a typical feed-forward artificial neural network, $f(x; \theta)$ is a composition of linear layers and non-linear activation functions. Just like the back-propagation approach, we can find gradients with respect to input $x$. For simplicity, we consider non-affine functions like $\tanh()$ and $\text{softplus}()$ as activations in our models, with no effect to the generality of the neural approximator. For instance, gradients for a single linear layer with $\tanh()$ non-linearity is given in Proposition 1.$^2$

**Proposition 1 (Finding gradients with respect to input).**

For a linear layer with $\tanh()$ non-linearity that is defined as follows:

$$f(x; W, b) = \tanh(W \cdot x + b),$$

the derivative of this layer with respect to $x$ is given as

$$\nabla_x f(x; W, b) = W^T \times (1 - \tanh^2(W \cdot x + b)).$$

**Proof.** It is well known that $\nabla_x \tanh(x) = 1 - \tanh^2(x)$, therefore,

2. Derivations for other activation functions with a single linear layer are given in Supplementary Section 2, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2021.3087349.
Using the chain-rule, we can find the derivative of an arbitrary neural approximator \( f \) as \( \frac{d}{dx} f(x; \theta) \) is a composition of multiple such layers. Once we have the gradients, we can initialize an arbitrary input \( x \) and use gradient descent to minimize \( f(x; \theta) \) as

\[
x_{n+1} = x_n - \gamma \cdot \nabla_x f(x_n; \theta),
\]

where \( \gamma \) is the learning rate and \( n \) is the iteration count. We apply Eq. (12) until convergence. The calculation of \( \nabla_x f(x_n; \theta) \) in the above equation is carried out in a similar fashion as back-propagation for model parameters. We can use momentum based methods like Adam [17] or annealed schedulers like cosine annealing with warm restarts [19] to prevent the optimization loop getting stuck in local optima. As we show in our experiments, the combination of gradient calculation with respect to inputs and advances like momentum and annealing allow faster optimization avoiding local minima, giving a more robust and efficient approach.

**Algorithm 1. The GOBI scheduler**

Require:
- Pre-trained function approximator \( f(x; \theta) \)
- Dataset used for training \( \Lambda \)
- Convergence threshold \( \epsilon \)
- Iteration limit \( \sigma \)
- Learning rate \( \gamma \)

Initial random decision \( \mathcal{D} \)

1. **procedure** **MINIMIZE** \( (\mathcal{D}, f, z) \)
2. Initialize decision matrix \( \phi(\mathcal{D}); i = 0 \)
3. do
4. \( x \leftarrow [z, \phi(\mathcal{D})] \) \( \triangleright \) Concatenation
5. \( \delta \leftarrow \nabla_x f(x; \theta) \) \( \triangleright \) Partial gradient
6. \( \phi(\mathcal{D}) \leftarrow \phi(\mathcal{D}) - \gamma \cdot \delta \) \( \triangleright \) Decision update
7. \( i \leftarrow i + 1 \)
8. while \( |\delta| > \epsilon \) and \( i \leq \sigma \)
9. Convert matrix \( \phi(\mathcal{D}) \) to scheduling decision \( \mathcal{D}^* \)
10. return \( \mathcal{D}^* \)
11. end procedure
12. **procedure** **GOBI**(scheduling interval \( I_t \))
13. if \( t = 0 \)
14. Initialize random decision \( \mathcal{D} \)
15. else
16. \( \mathcal{D} \leftarrow \mathcal{D}^* \) \( \triangleright \) Output for the previous interval
17. Get \( \phi(H_{t-1}) \)
18. \( \mathcal{D}^* \leftarrow \text{MINIMIZE}(\mathcal{D}, f(\phi(A_{t-1}), \phi(H_{t-1}))) \)
19. Fine-tune \( f \) with loss = \( \text{MSE}(\mathcal{O}(P_{t-1}), f([\phi(A_{t-2}), \phi(H_{t-2}), \phi(D^{t-1})]; \theta)) \)
20. return \( \mathcal{D}^* \)

5.4 Scheduling

After training the model \( f \), at start of each interval \( I_t \), we optimize \( \phi(\mathcal{D}) \) by the following rule (\( \gamma \) is the learning rate) until the absolute value of the gradient is more than the convergence threshold \( \epsilon \), i.e.,

\[
\phi(D)_{n+1} \leftarrow \phi(D)_n - \gamma \cdot \nabla \phi(D)f(x_n; \theta).
\]

In dynamic scenarios, as the approximated function for the objective \( \mathcal{O}(P_t) \) changes with time, we continuously fine-tune the neural approximator as per the loss function \( \text{MSE}(\mathcal{O}(P_{t-1}), f(\phi(A_{t-2}), \phi(H_{t-2}), \phi(D^{t-1})]; \theta)) \) (line 20 in Algorithm 1). Equation (13) then leads to a decision matrix with a lower objective value (line 6 in Algorithm 1). Hence, gradient-based iteration with continuous fine-tuning of the neural approximator allows GOBI to adapt quickly in dynamic scenarios. When the above equation converges to \( \phi(D^*) \), the rows represent the likelihood of allocation to each host. We take the arg max over each row to determine the host to which each task should be allocated as \( D^* \). This then becomes the final scheduling decision of the GOBI approach. The returned scheduling decision is then run on a simulated or physical platform (as in Eq. (6)) and tasks sets are updated as per Eqs. (3), (4), and (5). To find the executed decision \( \mathcal{D} \), we sort all tasks in \( D^* \) in descending order of their wait times (breaking ties uniformly at random) and then try to allocate them to the intended host in \( D^* \). If the allocation is not possible due to utilization constraints, we ignore such migrations and do not add them to \( \mathcal{D} \). At run-time, we fine-tune the trained neural model \( f \), by newly generated data for the model to adapt to new settings.

6 THE COSCO FRAMEWORK

Motivated from prior coupled-simulators [44], we now present the COSCO framework as introduced in Section 1. A basic architecture is shown in Fig. 5. Implementation details of the COSCO framework are given in Supplementary Section 1, available online. The hosts might correspond to simulated hosts or physical compute nodes with Instructions per second (IPS), RAM, Disk and Bandwidth capacities with power consumption models with CPU utilization. The workloads are either time-series models of IPS, RAM, Disk, Bandwidth requirements in the case of simulation or are actual application programs in the case of physical experiments. At the beginning of each scheduling interval, \( N_t \) workloads are created. The framework also maintains a waiting queue for workloads that cannot be allocated in the previous interval (\( W_{t-1} \)).

6.1 Simulator

The Simulator uses a Scheduler (GOBI for instance), which uses the utilization metrics and QoS parameters to take an allocation decision for new workloads or migration decisions for active workloads. At the start of a scheduling
interval $I_t$, the simulator destroys workloads completed in $I_{t-1}$, denoted as $L_t$. It also gets workloads $N_t \cup W_{t-1}$. The Scheduler decides which host to allocate these tasks to and whether to migrate active tasks from $A_{t-1} \setminus L_t$. The Simulator object considers all $N_t \cup W_{t-1}$ tasks in the order of the interval at which they were created. If the target host is unable to accommodate a task, it is added to the wait queue. This is done for each utilization metric of active task $a^j_t$ in $U(a^j_t)$. As for new tasks, the utilization metrics are unknown, $u = \forall u \in U(n^j_t) \forall n^j_t \in N_t$. To prevent overflow of host resources, we compare for each new task $n^j_t$ the sum of the maximum possible utilization metrics of the corresponding application and target host against the host’s maximum capacity (more details in Supplementary Section 1), available online. Hence, at each interval, Eq. (5) holds. Instead of running the tasks on physical machines, we use trace driven discrete-event simulation.

### 6.2 Framework

The Framework instantiates the tasks as Docker containers. Docker is a container management platform as a service used to build, ship and run containers on physical or virtual environments. Moreover, the hosts are physical computational nodes in the same Virtual Local Area Network (VLAN) as the server machine on which the Scheduler program runs. The server communicates with each host via HTTP REST APIs [43]. At every host machine, a DockerClient service runs with a Flask HTTP web-service [48]. Flask is a web application REST client which allows interfacing between hosts and the server. To migrate a container, we use the Checkpoint/Restore In Userspace (CRU) [49] tool. For each migration decision $(a^j_t, h_t) \in \bar{D}(A_t)$, a checkpoint of $a^j_t$ is created which is equivalent to a snapshot of the container. Then, this snapshot is transferred (migrated) to the target host $h_t$, where the container is resumed (restored). The allocation and migrations are done in a non-blocking fashion, i.e. the execution of active containers is not halted when some tasks are being migrated.

### Algorithm 2. The GOBI* Scheduler

```plaintext
Require:
Pre-trained function approximator $f^*(x; \theta^*)$
Pre-trained LSTM model $LSTM(\{U(a^j_t) \forall t' < t\})$
Dataset used for training $A'$; Convergence threshold $\epsilon$
Learning rate $\eta$; Initial random decision $D$

1: \hspace{0.5cm} procedure GOBI (scheduling interval $I_t$)
2: \hspace{1cm} if ($t = 0$) then
3: \hspace{2cm} Initialize random decision $D$
4: \hspace{1cm} else
5: \hspace{2cm} $D \leftarrow D^*$ \hspace{0.5cm} // Output for the previous interval
6: \hspace{2cm} Get $\phi(A_{t-1}), \phi(H_{t-1})$
7: \hspace{2cm} $U(a^j_t)\forall a^j_t \in A_t \leftarrow LSTM(\{U(a^j_t) \forall t' < t\})$
8: \hspace{2cm} $D \leftarrow GOBI(I_t)$
9: \hspace{2cm} $S(D(A_t), \{U(a^j_t)\forall a^j_t \in A_t\})$
10: \hspace{2cm} $O(P_t) \leftarrow \alpha \cdot AEC_t + \beta \cdot ART_t$
11: \hspace{2cm} $D^* \leftarrow \text{MINIMIZE}(D, f, \{\phi(A_{t-1}), \phi(H_{t-1}), O(P_t)\})$
12: \hspace{2cm} Fine-tune $f^*$ with loss $L$
13: \hspace{2cm} $MSE(O(P_{t-1}), f(\{\phi(A_{t-2}), \phi(H_{t-2}), \phi(D^{t-1})\}; \theta)) +$
14: \hspace{2cm} $1 \cdot O(P_{t-1}) < O(P_{t-1}) \times MSE(D^{t-1}, D^{t-1})$
15: \hspace{2cm} return $D^*$
```

### 6.3 Model Interface

We now discuss the novelty of the COSCO framework. As all scheduling algorithms and simulations are run on a central fog broker [43], it can be easily run in fog environments which follow a master-slave topology. As discussed in Section 4, a simulator can execute tasks on simulated host machines to return QoS parameters $P$. To execute an interval $I_t$, the simulator needs utilization metrics $U(a^j_t), \forall a^j_t \in A_t$, and a scheduling decision $D^*$ to be executed on the simulator. Thus, at the beginning of the interval $I_t$, we have $L_t, N_t, W_{t-1}$ and $A_{t-1}$. After checking for the possibility of allocation/migration, we get $\bar{D}(A_t)$, $N_t$ and $W_{t-1}$. Now, using Eq. (3), we find $A_t$. Using given utilization metrics $U(a^j_t)$ and $\bar{D}(A_t)$, we execute $A_t$ tasks on the simulator to get $S(\bar{D}(A_t), \{U(a^j_t)\forall a^j_t \in A_t\}) = \{U(h^j_t)\forall h_t \in H_t, P_t\}$. This means that at the beginning of interval $I_t$, the COSCO framework allows simulation of the next scheduling interval (with an action of interest $D$ and predicted utilization metrics) to predict the values of the QoS parameters in the next interval $P_t$. This single step look-ahead simulation allows us to take better scheduling decisions in the GOBI* algorithm, as shown in the next section.

### 7 Extending GOBI to GOBI*

Now that we have a scheduling approach based on back-propagation of gradients, we extend it to incorporate simulated results as described in Section 1. To do this, we use the following components.

1. **GOBI Scheduler**: We assume that we have the GOBI scheduler which can give us a preferred action of interest $D$. Thus, at the start of a scheduling interval $I_t$, we get $D = GOBI(I_t)$ (line 8 in Algorithm 2). $D$ now denotes the decision of GOBI*.
2. **Utilization prediction models**: We train a utilization metric prediction model, such that using previous utilization metrics of the tasks, we get a predicted utilization metric set for the next interval. As is common in prior work [50], we use a Long-Short-Term-Memory (LSTM) neural network for this and train it using the same dataset that we used for training the GOBI neural approximator. Thus, at the start of interval $I_t$, using $D$ from GOBI and checking allocation possibility, we get $N_t$ and $W_{t-1}$. Using (3), we get $A_t$. Then, we predict $U(a^j_t), \forall a^j_t \in A_t$, using the LSTM model (line 7 in Algorithm 2). Hence, we get the $\phi(A_{t-1}), \phi(H_{t-1})$ matrices.
3. **Simulator**: Now that we have an action $D$ and predicted utilization metrics $U(a^j_t), \forall a^j_t \in A_t$, we can use the simulator to predict the QoS parameters at the end of $I_t$ as described in Section 6.3 (lines 9 and 10 in Algorithm 2).

Since at the start of the interval $I_t$, we do not have utilization models of all containers and subsequently hosts for $I_t$, our GOBI model is forced to predict QoS metrics at the end of $I_t$ using only utilization metrics of the previous interval, i.e., $I_{t-1}$. This can make GOBI’s predictions inaccurate in some cases. However, if we can predict with reasonable accuracy the utilization metrics in $I_t$ using the utilization values of the previous intervals, we can simulate and get an improved estimate of the QoS parameters during the next
period $I_t$. Adding these as inputs to another neural approximator for $O(\hat{P}_t)$ denoted as $f^*(x, \theta^*)$, we now have $x = \{\phi(A_{t-1}), \phi(H_{t-2}), O(\hat{P}_t), \phi(D^*)\}$.

Again, using a random scheduler and pre-trained GOBI model, we generate a dataset $\Lambda^*$ to train this new neural approximator $f^*$. At execution time, we freeze the neural model $f$ of GOBI and fine tune $f^*$ with the loss function. This loss function is the MSE of objective values. We add the MSE between the predicted action $D$ by GOBI* and $D$ by GOBI in case the estimated objective value of GOBI is lower than that of GOBI*. Thus,

$$L = MSE(O(P_{t-1}), f^*([\phi(A_{t-2}), \phi(H_{t-2}), \phi(D^0); \theta^*])) + 1(O(\hat{P}_{t-1}) < O(P_{t-1})) \times MSE(D^0, D^*).$$

(14)

Using this trained model, we can now provide schedules as shown in Algorithm 2 and Fig. 6. At interval $I_t$, the previous output of GOBI*, i.e., $D^*$ is given to the GOBI model as the initial decision. The GOBI model uses utilization metrics to output the action $D$. Using a single-step simulation, we obtain an estimate of the objective function for this decision. Then, GOBI* uses this estimate to predict the next action $D$. Based on whether GOBI*’s decision is better than GOBI or not, GOBI* is driven to the decision which has lower objective value. This interactive dynamic should allow GOBI* to make a more informed prediction of QoS metrics for $I_t$ and hence perform better than GOBI.

### 7.1 Convergence of GOBI and GOBI*

We outline two scenarios of an ideal and a more realistic case to describe the interactive dynamic between GOBI and GOBI* and how the latter optimizes the objective scores.

**Ideal Case.** Consider the case when the neural approximator of GOBI, $f$ can perfectly predict the objective value of the next interval $I_t$ as $O(\hat{P}_t)$. In this case, the predicted action $D$ is optimal. In such a setting, we expect GOBI* to converge such that it predicts the same action as $D$. Assuming an ideal simulator which exactly mimics the real-world, $O(\hat{P}_t) = O(P_t)$. Thus, any action other than $D$ is sub-optimal. Hence, to minimize the loss metric, GOBI* would directly forward $D$ and ignore all other information and converge to GOBI (which is optimal).

**Real Case.** While the limited explainability of neural networks prevent us from giving formal convergence properties, we supply of number of technical observations that justify the convergence we have seen in practice when applying the methods to real systems. Thus, considering the real-case, time-bound training only gives a sub-optimal approximator $f$ for GOBI. Hence, in general the predicted decision $D$ is not always optimal. Assuming GOBI* predicts a decision other than GOBI, GOBI* will sometimes get a sub-optimal action $D$ such that $O(\hat{P}_t) < O(P_t)$ and other times $O(\hat{P}_t) > O(P_t)$. The former is the case when the action predicted by GOBI, $D$, is better and the latter is the case when the GOBI* action, $D$, has a lower objective score. (1) In the case of $D$ being better than $\hat{D}$ (having lower objective score), in an attempt to minimize the first part of the loss,

$$L_1 = MSE(O(P_{t-1}), f^*([\phi(A_{t-2}), \phi(H_{t-2}), \phi(D^0); \theta^*]),$$

(15)

GOBI* would converge to avoid $\hat{D}$ (considering the ideal simulator that returns exact answers). Furthermore, the predicted action $D$ has a lower objective score, hence the optimization loop would tend to converge to this or better scheduling decisions. (2) In the case of $D$ being better than $\hat{D}$, which means that $O(\hat{P}_{t-1}) < O(P_{t-1}),$ GOBI* is given an incentive to predict $\hat{D}$. This is because the latter part of the loss makes the model converge such that it minimizes

$$L_2 = MSE(D^{0-1}, D^{t-1}).$$

(16)

This, with the former part of the loss, creates an interactive dynamic such that the GOBI* model can find scheduling decisions even better than the one predicted by the GOBI model. However, in cases when GOBI*’s prediction is worse, the GOBI’s prediction aids the model to quickly converge to a better decision (that of GOBI’s). Thus, in both cases, each model update based on the loss in Eq. (14), yields a scheduling decision with objective score same or lower than that given by GOBI. Assuming a sufficiently long training time, GOBI* would eventually converge to predict $D$ such that the objective value when the decision is $D$ is never greater than when it is $\hat{D}$. We observe in our experiments that in the real-case GOBI* always performs better than GOBI in terms of objective scores. The interactive training with simulations allows GOBI* to converge quickly and adapt to volatile environments.

The convergence plots for GOBI* are shown in Fig. 7. It demonstrates that as the model gets trained, the probability of the GOBI*’s actions being better than GOBI increases. This means that initially $L_2$ has a high contribution to the loss for model training which aids GOBI* to quickly converge to GOBI’s performance. This can be seen by the sudden drop in $L_1$ in Fig. 7a. This is also shown by Fig. 7b that GOBI*’s loss converges faster with the additional $L_2$ component.
8 PERFORMANCE EVALUATION

8.1 Experimental Setup
To test the efficacy of the proposed approaches and compare against the baseline methods, we perform experiments on both simulated and physical platforms. As COSCO has the same underlying models of task migration, workload generation and utilization metric values for both simulation and physical test-bench, we can test all models on both environments with the same underlying assumptions. As in prior work [51], we use \(\alpha = \beta = 0.5\) in (8) for our experiments. Moreover, we consider all tasks, to be allocated and migrated, in batches with each batch having feature vectors corresponding to up to \(M = |H|^2\) tasks. If number of tasks is less than \(M\), we pad the input matrix with zero vectors. This style of inference strategy is in line with prior work [3]. This is due to the limitation of non-recurrent neural networks to take a fixed size input for a single forward pass, hence requiring action inference for tasks being executed in batches.

**Physical Environment.** We use the Microsoft Azure cloud provisioning platform to create a test-bed of 10 VMs located in two geographically distant locations. The gateway devices are considered to be in the same LAN of the server which was hosted in London, United Kingdom. Of the 10 VMs, 6 were hosted in London and 4 in Virginia, United States. The host capacities \(C(h)\) are shown in Table 3 and Supplementary Table 1, available online. All machines use Intel Haswell 2.4 GHz E5-2673 v3 processor cores. The cost per hour (in US Dollar) is calculated based on the costs of similar configuration machines offered by Microsoft Azure at the datacenter in the South UK region.\(^5\) The power consumption models are taken from the SPEC benchmarks repository.\(^4\) We run all experiments for 100 scheduling intervals, with each interval being 300 seconds long, giving a total experiment time of 8 hours 20 minutes. We average over 5 runs and use diverse workload types to ensure statistical significance.

**Simulation Environment.** We consider 50 hosts machines as a scaled up version of the 10 machines from the last subsection. Here, each category has 5 times the instance count to give a total of 50 machines in a comparatively larger-scale fog environment as considered in prior art [5], [52]. As we cannot place simulated nodes in geographically distant locations, we model the latency and networking characteristics of these nodes in our simulator as per Table 3.

8.2 Model Training and Assumptions
For the GOBI and GOBI* algorithms we use standard feed-forward neural models with the following characteristics adapted from [3]. We use non-affine activation functions for our neural approximators to be differentiable for all input values.

- Input layer of size \(M \times F + |H| \times F' + M \times N\) for GOBI and \(M \times F + |H| \times F' + M \times N + 1\) for GOBI*. The non-linearity used here is softplus as in [3]. Note, \(|H|\) may vary from 10 in tests in the physical environment to 50 in the simulator.
- Fully connected layer of size 128 with softplus activation.
- Fully connected layer of size 64 with tanhshrink activation.
- Fully connected layer of size 1 with sigmoid activation.

To implement the proposed approach, we use PyTorch Autograd package [53] to calculate the gradients of the network output with respect to input keeping model parameters constant. We generate training data for the GOBI model \(f\) by running a random scheduler for 2,000 intervals on the simulator. We then run the random scheduler on the framework for 300 intervals and fine-tune the GOBI model on this new dataset. For GOBI*, we use a random scheduler and predictions of the GOBI model \(f\) to generate data of the form described in Section 5 of size 2,000 on simulator and fine-tune it on data corresponding to 300 intervals on the framework. The dataset was used to create the matrices \([\phi(A_{-1}), \phi(H_{-1}), \phi(D)]\), where each column of \(\phi(A_{-1})\), \(\phi(H_{-1})\) is normalized by the maximum and minimum utilization values, and \(\phi(D)\’s\) rows are one-hot decision vectors. To train a model, we use the AdamW optimizer [54] with learning rate \(10^{-5}\) and randomly sample 80 percent of data to get the training set and the rest as the cross-validation set.\(^6\) The convergence criterium used for model training is the loss of consecutive epochs summed over the last 10 epochs is less than \(10^{-2}\). For GOBI*, we also train predictors for utilization metrics. Specifically, we use a LSTM neural

---

3. Microsoft Azure pricing calculator for South UK https://azure.microsoft.com/en-gb/pricing/calculator/  
4. More details in Supplementary Section 4, available online.  
5. The definitions of these activation functions can be seen at the PyTorch web-page: https://pytorch.org/docs/stable/nn. 
6. All model training and experiments were performed on a system with configuration: Intel i7-10700K CPU, 64GB RAM, Nvidia GTX 1060 and Windows 10 Pro OS.
network [55] with a single LSTM cell to achieve this and train it using the same dataset used to train the GOBI model.

8.3 Workloads
To generate workloads for training the GOBI and GOBI* models and testing them against baseline methods, we use two workload characteristics Bitbrain traces and DeFog applications. These were chosen because of their non-stationarity, their highly volatile workloads and their similarity with many real-world applications.

(1) BitBrain traces. The dynamic workload is generated for cloudlets based on the real-world, open-source Bitbrain dataset [56]. This dataset consists of real traces of resource utilization metrics from 1,750 VMs running on BitBrain distributed datacenter. The workloads running on these servers are from a variety of industry applications including computational analytical programs used by major banks, credit operators and insurers [56] and are commonly used for benchmarking fog-cloud models [3], [57], [58]. The dataset consists of workload information for each time-stamp (separated by 5 minutes) including the number of requested CPU cores, CPU usage in terms of Million Instructions per Second (MIPS), RAM requested with Network (receive/transmit) and Disk (read/write) bandwidth characteristics. These different categories of workload data constitute the feature values of \( \phi(A_{t-1}) \) and \( \phi(H_{t-1}) \). As these workload traces correspond to applications running on real infrastructure, we use these to run our simulations and generate training data. At the start of each interval \( I_t \), the size of the new tasks \( N_t \) set follows a discrete Poisson distribution\(^8\) Poisson(\( \lambda \)), as per prior works [3], [59]. Here, \( \lambda = 1.2 \) jobs for \( |H| = 10 \) and \( \lambda = 5 \) jobs for \( |H| = 50 \).

(2) DeFog applications. DeFog [60] is a fog computing benchmark which consists of six real-time heterogeneous workloads such as Yolo, Pocketsphinx, Aeneas, FogLamp and iPocketMon. We use three specific heterogeneous applications of DeFog: Yolo (Memory, Bandwidth and Compute Intensive benchmark), PocketSphinx (Compute Intensive) and Aeneas (Bandwidth Intensive). Yolo runs with 1,500 user requests and each request uploads a unique image with different density and can run up to three intervals. PocketSphinx runs with 320 user requests and Aeneas with 1,500. Here too, at the start of each interval \( I_t \), the size of the new tasks \( N_t \) set follows a Poisson distribution Poisson(\( \lambda \)) with \( \lambda = 1.2 \) jobs. Out of the \( N_t \) tasks, the distribution of selection of Yolo/PocketSphinx/Aeneas follows the probabilities of \( (p_y, p_p, p_a) = (0.33) \). However, to test diverse workload characteristics, we also run using any one as 0.80 and other two as 0.10. We call these workload dominant runs. We perform all our experiments to compare with baselines using the DeFog benchmarks.

8.4 Baseline Models
We evaluate the performance of the proposed algorithms and compare them against the state-of-the-art scheduling approaches. The reasons for comparing against these baselines are described in Section 2. We consider two light-weight heuristic based schedulers LR-MMT and MAD-MC that have low scheduling times, but higher energy consumption, response times and SLO violations. We also consider a deep-learning based gradient-free optimization method GA, and two reinforcement-learning based schedulers DQLCM and A3C. Finally, we compare against a one max-weight based allocation method POND.

8.5 Evaluation Metrics
We use the following evaluation metrics to test the GOBI and GOBI* models as motivated from prior works [1], [3], [5], [9], [43]. We also use AEC and ART as in Section 5.

(1) SLO Violations which is given as

\[
\sum_t \sum_{\ell \in L_t} \mathbb{I}(\text{Response Time}(\ell_t) \leq \psi(\ell_t)) / \sum_t |L_t|,
\]

where \( \psi(\ell_t) \) is the 95th percentile response time for this application type (Yolo/PocketSphinx/Aeneas for DeFog and random/sequential for Bitbrain) on the state of the art baseline A3C. This definition of percentile-based SLO is defined for the response time metrics of completed tasks and is inspired from [61].

(2) Fairness which is given as the Jain’s Fairness Index

\[
\frac{(\sum_t \sum_{\ell \in L_t} \text{Response Time}(\ell_t))^2}{(\sum_t |L_t|) \times (\sum_t \sum_{\ell \in L_t} \text{Response Time}(\ell_t)^2)}.
\]

(3) Average Migration Time which is the average time for the migration of tasks over all intervals in the run

\[
\sum_t \sum_{(a, h) \in D(A_t)} \frac{\text{Migration Time}(a, h)}{|D(A_t)|}.
\]

(4) Scheduling Time which is the average time to reach the scheduling decision over all intervals in the run.

(5) Average Wait Time which is the average time for a container in the wait queue before it starts execution.

8.6 Results
In this section we provide comparative results, showing how GOBI and GOBI* perform against other baselines as described in Section 8.4. We compare the proposed approaches with respect to the evaluation metrics described in Section 8.5. Additional results are given in Supplementary Section 5, available online. The graphs in Fig. 8 show the results for runs with...
length of 100 scheduling intervals viz 8 hours 20 minutes using the DeFog workloads on 10 physical Azure machines. Fig. 9 shows similar trends in the result for 50 hosts in a simulated environment.

Fig. 8a shows the energy consumption of Azure host machines for each scheduling policy. Among the baseline approaches, A3C consumes the least average interval energy of 221.63 KW-hr. Running the GOBI approach on the Azure platform consumes 193.11 KW-hr, 12.86 percent lower than A3C. Further, GOBI* consumes the least energy 188.26 KW-hr (15.05 percent lower than that of A3C). The major reason for this is the low response time of each application, which leads to more tasks being completed within the same duration of 100 intervals. Thus, even with similar energy consumption across all policies, GOBI and GOBI* have a much higher task completion rate of 112-115 tasks in 100 intervals compared with only 107 tasks. Similarly, when we consider 50 host machines in a simulated platform (Fig. 9a), GOBI* still consumes 9.39 percent lower than A3C. The energy consumption with GOBI is very close to A3C.

Fig. 8b shows the average response time for each policy. Here, response time is the time between the creation of a task from an IoT sensor and up to the gateway receiving the response. Among the baselines models, POND has the lowest average response time of 266.53 seconds. Here, GOBI and GOBI* have 226.04 and 156.09 seconds (15.19-41.43 percent better than POND). For the three applications, the average response time for each policy is shown in Fig. 8f. Clearly, among the DeFog benchmark applications, Yolo takes the maximum time. The highest response time for Yolo among all policies is for GA as it prefers scheduling shorter jobs first i.e. Pocketsphinx and Aeneas to reduce response time. This is verified by the results which show that Aeneas has the lowest response time when the GA policy is used. Another disadvantage of this preference is that it leads to high wait times when running the GA policy. However, in larger scale experiments on 50 hosts in simulation (Fig. 9b), GA has much lower wait times leading to the best response times among the baseline algorithms. Here, compared to GA, GOBI and GOBI* have 17.98-20.87 percent lower average response times, respectively.
Fig. 8k shows the average waiting time (in intervals) for tasks running on the Azure framework with each policy. As prior works have established that RL approaches are slow to adapt in highly volatile environments [3], [62], the A3C and DQLCM approaches are unable to adapt when a host is running at capacity. This means that for a large number of scheduling intervals (47.72 percent), they predict an overloaded host. Hence, the task cannot be assigned in the same interval and has to wait until either it is assigned to another host, or the resources of this host are released. This is also reflected in the wait time per application (Fig. 8g). As established in prior work, reinforcement-learning approaches scale poorly and hence this waiting time gets more pronounced when running with 50 hosts (Fig. 9h). However, as GOBI and GOBI* are able to adapt quickly to the environment, because of the neural model update in each iteration, they do not face such problems even on larger scale experiments.

As seen in Fig. 8d, due to the high response times of the baseline approaches, they have a high SLO violation rate. For the experiments of physical hosts, as POND has the lowest response time, the SLO violation rate is also the lowest among the baselines (2.75 percent). GOBI has only 0.9 percent SLO violations and GOBI* has none. This is due to the back-propagation approach which minimizes the response time, as well as avoiding local optima using the adaptive moment estimation approach (Adam optimizer). For GOBI, the SLO violations are only for Yolo tasks (4.5 percent). In the simulation environment, due to the high wait and response times of the A3C, DQLCM and POND schedulers, their SLO violation rates are highest among all methods. Here, GOBI has only 1.1 percent SLO violations. Due to the high response times of A3C and DQLCM for 50 hosts, the SLO itself is much higher giving the GA approach with low response times only 0.2 percent SLO violations. GOBI* has no SLO violations in this case as well.

The main reason for low response times in GOBI and GOBI* is their low scheduling time. Even with single-step simulation, the back-propagation based optimization strategy with the AdamW optimizer converges faster than the gradient-free optimization methods like A3C, DQLCM GA and POND. This is apparent from the scheduling times seen in Figs. 8j and 8i (shaded region shows the 95 percent confidence interval). Here, MAD-MC has the lowest scheduling time, with GOBI next at 2.65 seconds and GOBI* at 5.88 seconds. Even with 50 hosts (Fig. 9g), GOBI and GOBI* have one of the lowest scheduling times across all policies (except MAD-MC).

Fig. 8k shows the average waiting time (in intervals) for tasks running on the Azure framework with each policy. As prior works have established that RL approaches are slow to adapt in highly volatile environments [3], [62], the A3C and DQLCM approaches are unable to adapt when a host is running at capacity. This means that for a large number of scheduling intervals (47.72 percent), they predict an overloaded host. Hence, the task cannot be assigned in the same interval and has to wait until either it is assigned to another host, or the resources of this host are released. This is also reflected in the wait time per application (Fig. 8g). As established in prior work, reinforcement-learning approaches scale poorly and hence this waiting time gets more pronounced when running with 50 hosts (Fig. 9h). However, as GOBI and GOBI* are able to adapt quickly to the environment, because of the neural model update in each iteration, they do not face such problems even on larger scale experiments.

As seen in Fig. 8d, due to the high response times of the baseline approaches, they have a high SLO violation rate. For the experiments of physical hosts, as POND has the lowest response time, the SLO violation rate is also the lowest among the baselines (2.75 percent). GOBI has only 0.9 percent SLO violations and GOBI* has none. This is due to the back-propagation approach which minimizes the response time, as well as avoiding local optima using the adaptive moment estimation approach (Adam optimizer). For GOBI, the SLO violations are only for Yolo tasks (4.5 percent). In the simulation environment, due to the high wait and response times of the A3C, DQLCM and POND schedulers, their SLO violation rates are highest among all methods. Here, GOBI has only 1.1 percent SLO violations. Due to the high response times of A3C and DQLCM for 50 hosts, the SLO itself is much higher giving the GA approach with low response times only 0.2 percent SLO violations. GOBI* has no SLO violations in this case as well.

Fig. 9. Comparison of GOBI and GOBI* against baselines on simulator with 50 hosts.

8.7 Comparing GOBI and GOBI*
Now that GOBI and GOBI* have been shown to out-perform prior works, we directly compare the two approaches. Table 4 shows GOBI* has a lower objective value (16.4 percent) and prediction error of objective value (14.2 percent) for the next interval. This clearly shows that GOBI* can optimize the objective value better than GOBI. This is because GOBI predicts the objective value for the interval $I_t$ using...
only the utilization metrics of interval $t_{i-1}$. However, GOBI* has more information as the LSTM models predict utilization metrics for $t_i$ and the GOBI model gives a tentative action, which is then used to simulate $t_i$ and get a reasonable estimate of the objective value for $t_i$. This estimate with the utilization metrics of $t_{i-1}$ aid GOBI* to have much closer predictions and hence better scheduling decisions. However, the scheduling time of GOBI* is more than twice that of GOBI. This makes the GOBI approach more appropriate for environments with resource constrained servers.

9 CONCLUSION AND FUTURE WORK

We have presented a coupled-simulation approach to leverage simulators to predict the QoS parameters and make better decisions in a heterogeneous fog setup. The presented COSCO framework allows deployment of a holistic platform that provides an easy to use interface for scheduling policies to access simulation capabilities. Moreover, we have presented two scheduling policies based on back-propagation of gradients with respect to input, namely GOBI and GOBI*. These approaches use neural approximators to model objective scores for the scheduling decisions. GOBI* uses GOBI, predictors of utilization characteristics and an underlying simulator to better predict the objective values leading to better decisions. Comparing GOBI and GOBI* against state-of-the-art schedulers using real-world fog applications, we see that our methods are able to reduce energy consumption, response time, SLO violations and scheduling time. Between GOBI and GOBI*, GOBI* gives better QoS values; however, GOBI is more suitable for resource constrained servers.

For the future, we propose to extend the COSCO framework to allow workflow models for serverless computing. Extending to serverless would allow us to perform fine-grained auto-scaling, increase productivity and improve flexibility and logistics [63]. For the back-propagation approach, we wish to extend our methods to consider layer types and activations like recurrent, convolution or residual with Rectified Linear Units (ReLU). This is because such non-differentiable functions are increasingly being used to approximate diverse objective functions. More advanced layer types would also allow us to model temporal and spatial characteristics of the environment.

SOFTWARE AVAILABILITY

The code is available at https://github.com/imperial-gore/COSCO. The Docker images are available at https://hub.docker.com/u/shreshthtuli. The training datasets and execution traces are available at https://doi.org/10.5281/zenodo.4897944, released under the CC BY 4.0 license.

AUTHOR CONTRIBUTIONS

S.T. designed and implemented the COSCO framework with GOBI, and GOBI* algorithms. S.P aided in software debugging. S.T. designed and conducted the experiments. S.T., S.P., G.C. and N.R.J. analyzed the experimental results. S.T., G.C. and N.R.J. wrote the paper. G.C., S.N.S. and N.R.J. supervised the project.

ACKNOWLEDGMENTS

The work of Shreshth Tuli was supported by the Imperial College London through the President’s Ph.D. Scholarship Scheme. The work of Shivananda Poojara was supported by the European Social Fund through IT Academy Program. The work of Giuliano Casale was supported in part by the EU’s Horizon 2020 Program under Grant 825040.

REFERENCES

[1] S. S. Gill et al., “Transformative effects of IoT, blockchain and artificial intelligence on cloud computing: Evolution, vision, trends and open challenges,” Internet Things, vol. 8, 2019, Art. no. 100118.
[2] A. Toor et al., “Energy efficient edge-of-things,” EURASIP J. Wirel. Commun. Netw., vol. 2019, no. 1, 2019, Art. no. 82.
[3] S. Tuli et al., “Dynamic Scheduling for stochastic edge-cloud computing environments using A3 learning and residual recurrent neural networks,” IEEE Trans. Mobile Comput., early access, 2020, doi: 10.1109/TMC.2020.3017079.
[4] N. Wang et al., “DYVERSE: Dynamic vertical scaling in multi-tenant edge environments,” Future Gener. Comput. Syst., vol. 108, pp. 598–612, 2020.
[5] D. Basu, X. Wang, Y. Hong, H. Chen and S. Bressan, “Learn-as-you-go with Megh: Efficient live migration of virtual machines,” IEEE Trans. Parallel Distrib. Syst., vol. 30, no. 8, pp. 1786–1801, Aug. 2019.
[6] K. Han et al., “Fog computing task scheduling strategy based on improved genetic algorithm,” Comput. Sci., vol. 4, 2018, Art. no. 22.
[7] X.-Q. Pham et al., “A cost-and performance-effective approach for task scheduling based on collaboration between cloud and fog computing,” Int. J. Distrib. Sensor Netw., vol. 13, no. 11, 2017, Art. no. 1550147717742073.
[8] O. Skarlat et al., “Optimized IoT service placement in the fog,” Serv. Oriented Comput. Appl., vol. 11, no. 4, pp. 427–443, 2017.
[9] A. Beloglazov and R. Buyya, “Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers,” Concurrency Comput.: Practice Experience, vol. 24, no. 13, pp. 1397–1420, 2012.
[10] T. Choudhari et al., “Prioritized task scheduling in fog computing,” in Proc. ACMSE Conf., 2018, pp. 1–8.
[11] G. Fox et al., “Learning everywhere: Pervasive machine learning for effective high-performance computation,” in Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops, 2019, pp. 422–429.
[12] S. Liu and N. Wang, “Collaborative optimization scheduling of cloud service resources based on improved genetic algorithm,” IEEE Access, vol. 8, pp. 150878–150890, 2020.
[13] V. M. Johnson and L. L. Rogers, “Accuracy of neural network approximators in simulation-optimization,” J. Water Resour. Plan. Manage., vol. 126, no. 2, pp. 48–56, 2000.
[14] G. Villarrubia et al., “Artificial neural networks used in optimization problems,” Neurocomputing, vol. 272, pp. 10–16, 2018.
[15] L. M. Rios and N. V. Sahinidis, “Derivative-free optimization: A review of algorithms and comparison of software implementations,” J. Glob. Optim., vol. 56, no. 3, pp. 1247–1293, 2013.
[16] L. Bogolubsky et al., “Learning supervised pagerank with gradient-based and gradient-free optimization methods,” in Proc. Neural Inf. Process. Syst., 2016, pp. 4914–4922.
[17] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014, arXiv:1412.6980.
[18] H. Pan and H. Jiang, “Annealed gradient descent for deep learning,” in Proc. 31st Conf. Uncertainty Artif. Intell., 2015, pp. 652–661.
[19] I. Louizos and F. Hutter, “SGD: Stochastic gradient descent with warm restarts,” 2016, arXiv:1608.03983.
[20] F. Scarselli and A. C. Tsol, “Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results,” Neural Netw., vol. 11, no. 1, pp. 15–37, 1998.
[21] K. Rehman et al., “A cloud-based development environment using Hadoop and hadoop for the co-emulation of a containerized electric vehicle fleet,” in Proc. IEEE/SICE Int. Symp. Syst. Integration (SII), 2019, pp. 47–54.
[22] B. S. Öngko et al., “Symbiotic simulation system: Hybrid systems model meets big data analytics,” in Proc. Winter Simul. Conf. (WSC), 2018, pp. 1358–1369.
[23] N. Nguyen-Thien and T. Tran-Cong, “Approximation of functions and their derivatives: A neural network implementation with MATLAB,” Appl. Math. Model., vol. 23, no. 9, pp. 687–704, 1999.
[24] G. R. Ghosal, D. Ghosal, A. Sim, A. V. Thakur, and K. Wu, “A comprehensive survey of some existing methods, and a new efficient algorithm,” arXiv:1608.03983.
[25] Z. Tang, X. Zhou, F. Zhang, W. Jia, and W. Zhao, “Migration scheduling algorithms,” in Proc. IEEE Symp. Ser. Comput. Intell., 2015, pp. 1423–1430.
[26] C. Gomes et al., “Co-simulation: A survey,” ACM Comput. Surv. (CSUR), vol. 51, no. 3, pp. 3–33, 2018.
[27] S. Krishnasamy et al., “Augmenting max-weight with explicit learning for serverless computing,” Future Gener. Comput. Syst., 2019, pp. 1350–1359.
[28] A. Samanta, Y. Li, and F. Esposito, “Battle of microservices: Empowering A* search algorithms with neural network techniques,” Future Gener. Comput. Syst., 2019, pp. 1350–1359.
[29] K. Rehman et al., “Approximation of functions and learning algorithms,” Future Gener. Comput. Syst., 2019, pp. 1350–1359.
[30] J. Bae, J. Lee, and S. Chong, “Beyond max-weight scheduling: A reinforcement learning-based approach,” in Proc. Int. Symp. Modeling Optim. Mobile, Ad Hoc, Wirrel. Netw. (WiOPT), 2019, pp. 1–8.
[31] P. van den Ven, S. Borst, and S. Sheer, “Instability of maxweight scheduling algorithms,” in Proc. IEEE INFOCOM, 2009, pp. 253–261.
[32] P. M. van den Ven et al., “Inefficiency of maxweight scheduling in spatial wireless networks,” Comput. Commun., vol. 36, no. 12, pp. 1350–1359, 2013.
[33] M. G. Markakis, E. Modiano, and J. N. Tsitsiklis, “Max-weight scheduling in queueing networks with heavy-tailed traffic,” IEEE Trans. Netw., vol. 22, no. 1, pp. 257–270, 2013.
[34] P. Auer et al., “Finite-time analysis of the multiarmed bandit problem,” Mach. learn., vol. 47, no. 2/3, pp. 235–256, 2002.
[35] S. Nandi et al., “Artificial neural-network-assisted stochastic process optimization strategies,” AIChE J., vol. 47, no. 1, pp. 126–141, 2001.
[36] S. Tuli et al., “FogBus: A blockchain-based lightweight framework for edge and fog computing,” J. Syst. Softw., vol. 154, pp. 22–36, 2019.
[37] S. Bosmans et al., “Testing IoT systems using a hybrid simulation based testing approach,” Computing, vol. 101, no. 7, pp. 857–872, 2018.
[38] H. Gupta et al., “iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments,” Softw.: Pract. Experience, vol. 47, no. 9, pp. 1275–1296, 2017.
[39] M. M. Mahmoud et al., “Towards energy-aware fog-enabled cloud of things for healthcare,” Comput. Elect. Eng., vol. 67, pp. 58–69, 2019.
[40] A. A. Mutlag et al., “Enabling technologies for fog computing in healthcare IoT systems,” Future Gener. Comput. Syst., vol. 90, pp. 62–78, 2019.
[41] M. Grinberg, Flash Web Development: Developing Web Applications with Python.Newton, MA, USA: O’Reilly Media, Inc., 2018.
[42] R. S. Venkatesh et al., “Fast in-memory CRIU for docker container management,” in Proc. Int. Conf. Cloud, Big Data, 2018, pp. 53–65.
[43] S. Gupta and D. A. Dinesh, “Resource usage prediction of cloud workloads using deep bidirectional long short term memory networks,” in Proc. IEEE Int. Conf. Adv. Netw. TeleCommun. Syst. (ANTS), 2017, pp. 1–6.
[44] S. Tuli et al., “HealthFog: An ensemble deep learning based smart healthcare system for automatic diagnosis of heart diseases in integrated IoT and Fog computing environments,” Future Gener. Comput. Syst., vol. 104, pp. 187–200, 2020.
[45] A. Ahmed and G. Pierre, “Docker container deployment in fog computing infrastructures,” in Proc. IEEE Int. Conf. Edge Comput. (EDGE), 2018, pp. 1–8.
[46] A. Paszke et al., “Automatic differentiation in pytorch,” in Proc. Adv. Conf. Neural Inf. Process. Syst., 2017, pp. 1–4.
[47] N. Saleh and M. Masahly, “A dynamic simulation environment for container-based cloud data centers using containerclouds,” in Proc. Ninth Int. Conf. Intell. Comput. Inf. Syst. (ICICIS), 2019, pp. 332–336.
[48] S. Hochreiter and J. Schmidhuber, “Long term short memory,” Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997.
[49] S. Shen, V. van Beek, and A. Iosup, “Statistical characterization of business-critical workloads hosted in cloud datacenters,” in Proc. IEEE/ACM Int. Symp. Cluster, Cloud, Grid Comput., 2015, pp. 465–474.
[50] J. Khamse-Ashari, I. Lambadaris, G. Kesidis, B. Urgaonkar, and Y. Zhao, “An efficient and fair multi-resource allocation mechanism for heterogeneous servers,” IEEE Trans. Parallel Distrib. Syst., vol. 29, no. 12, pp. 2686–2699, Dec. 2018.
[51] Z. Li et al., “Bayesian network-based virtual machines consolidation method,” Future Gener. Comput. Syst., vol. 69, pp. 75–87, 2017.
[52] T. He et al., “SLA-aware multiple migration planning and scheduling in SDN-NFV-enabled clouds,” J. Syst. Softw., vol. 176, 2021, Art. no. 110943.
[53] J. McChesney et al., “Defog: Fog computing benchmarks,” in Proc. ACM/IEEE Symp. Edge Comput., 2019, pp. 47–58.
[54] K. Boloor, R. Chirkova, Y. Vinioz, and T. Sala, “Dynamic request allocation and scheduling for context aware applications subject to a percentile response time SLA in a distributed cloud,” in Proc. IEEE Second Int. Conf. Cloud Comput. Technol. Sci., 2010, pp. 464–472.
[55] C. Findling et al., “Computational noise in reward-guided learning drives behavioral variability in volatile environments,” Nat. Neurosci., vol. 22, no. 7, pp. 709–717, 2019.
[56] G. Casale et al., “Radon: Rational decomposition and orchestration for serverless computing,” SICO Softw.-Intensive Cyber-Phys. Syst., vol. 35, no. 1, pp. 77–87, 2020.
Shreshth Tuli received the undergraduate degree from the Department of Computer Science and Engineering, Indian Institute of Technology Delhi, India. He is currently a President’s PhD scholar with the Department of Computing, Imperial College London, U.K. He was a visiting research fellow with CLOUDS Laboratory, School of Computing and Information Systems, University of Melbourne, Australia. His research interests include Internet of Things, fog computing, and deep learning. He was the recipient of the national level Kishore Vaigyanik Prtsahen Yojana scholarship holder from the Government of India for excellence in science and innovation.

Shivananda R. Poojara (Member, IEEE) is currently working toward the PhD degree with Mobile and Cloud Computing Laboratory, University of Tartu, Estonia. He is a junior research fellow with IT Academy Program. He has more than six years of teaching experience and was an intern with Nokia R&D Labs. His research interests include serverless computing, edge analytics, and fog and cloud computing. He is a member of IET, and ISTE.

Satish N. Srirama (Senior member, IEEE) received the PhD degree in computer science from RWTH Aachen University in 2008. He is currently an associate professor with the School of Computer and Information Sciences, University of Hyderabad, India. He is also a visiting professor and the honorary head of the Mobile & Cloud Lab, Institute of Computer Science, University of Tartu, Estonia. His current research interests include cloud computing, mobile cloud, IoT, fog computing, migrating scientific computing, and large scale data analytics to the cloud. He is the editor of Wiley Software: Practice and Experience Journal.

Giuliano Casale joined the Department of Computing, Imperial College London in 2010, where he is currently a reader. He was a research scientist and consultant with the capacity planning industry. He is a teacher and researcher in performance engineering and cloud computing topics on which he has authored or coauthored more than 100 refereed papers. He was on the technical program committee of more than 80 conferences and workshops and as a cochair for several conferences in performance and reliability engineering, such as the ACM SIGMETRICS/Performance and IEEE/IFIP DSN. His research work was the recipient of multiple awards, recently the Best Paper Award at ACM SIGMETRICS. He was on the editorial boards of IEEE Transactions on Network and Service Management and ACM Transactions on Modeling and Performance Evaluation of Computing Systems and the current chair of ACM SIGMETRICS.

Nicholas R. Jennings is the vice-provost for Research and Enterprise and the professor of artificial intelligence with Imperial College London. He is an internationally-recognised authority in the areas of AI, autonomous systems, cybersecurity and agent-based computing. He is a member of the U.K. government’s AI Council, the governing body of the Engineering and Physical Sciences Research Council, the Monaco Digital Advisory Council, and the chair of the Royal Academy of Engineering’s Policy Committee. Before joining Imperial, he was the U.K.’s first Regius professor of computer science (a post bestowed by the monarch to recognise exceptionally high quality research) and the U.K. Government’s first chief scientific advisor for national security.