Structure-aware reinforcement learning for node-overload protection in mobile edge computing

Anirudha Jitani*‡, Aditya Mahajan†‡, Zhongwen Zhu§, Hatem Abou-zeid¶,
Emmanuel Thepie Fapi§, and Hakimeh Purmehdi§

* School of Computer Science, McGill Univeristy, Montreal, Canada
† Electrical and Computer Engineering, McGill Univeristy, Montreal, Canada
‡ Montreal Institute of Learning Algorithms, Montreal, Canada
§ Global AI Accelerator, Ericsson, Montreal, Canada, ¶ Ericsson, Ottawa, Canada

***Abstract*—Mobile Edge Computing (MEC) refers to the concept of placing computational capability at the edge of the network to reduce the latency in handling the client requests. The performance of an edge server is adversely affected when it is overloaded, especially if it crashes due to overload and causes service failures. In this paper, a solution to prevent node from getting overloaded is analyzed by introducing an admission control policy. An adaptive admission control policy based low complexity RL (Reinforcement Learning) SALMUT (Structure-Aware Learning for Multiple Thresholds) is validated using several scenarios mimicking real world deployments. This approach performs as well as to the state-of-the-art deep RL algorithms such as PPO (Proximal Policy Optimization) and A2C (Advantage Actor Critic), but requires an order of magnitude less time to train, and outputs easily interpretable policy.**

***Index Terms*—Reinforcement Learning, Structure-aware Reinforcement Learning, Markov Decision Process, Mobile Edge Computing, Node Overload Protection.**

I. Introduction

In recent years, there has been a proliferation of computationally intensive smartphone applications such as Video-On-Demand, real-time online gaming, augmented reality, and virtual reality applications. There are several advantages of moving such computationally intensive tasks to a cloud computing platform but doing so runs the risk of increased latency. One option to minimize such latency is to place the computational capabilities close to the edge of the network using the paradigm known as Mobile Edge Computing (MEC) [1].

Various aspects of MEC from the point of view of the mobile user have been investigated in the literature. For example, the questions of when to offload to a mobile server, to which mobile server to offload, and how to offload have been studied extensively. See, [2]–[6] and references therein.

However, the design questions at the server level have not been investigated extensively. When an edge server receives a large number of requests in a short period of time (for example be due to a sporting event), the edge server can get overloaded, which can lead to service degradation or even node failure. When such service degradation occurs, edge servers are configured to offload requests to other nodes in the cluster in order to avoid the node crash. However, performing this migration takes extra time and reduces the resources available for other services provided by the cluster. Therefore, it is paramount to design *pro-active* mechanisms that prevent a node from getting overloaded using dynamic offloading policies that can adapt to service request dynamics.

In this paper, we study the problem of node-overload protection for a single edge node. We first model the problem to incorporate practical considerations of server holding, processing and offloading costs. Then we develop an offloading policy that will reject and offload new requests to balance the overall running cost of the system. In the simplest case, when the request arrival process is time-homogeneous, we model the system as a continuous-time Markov decision process (MDP) and use the *uniformization technique* [7], [8] to convert the continuous-time MDP to a discrete-time MDP, which can then be solved using standard dynamic programming algorithms [9].

However, solving a dynamic program requires the knowledge of the system parameters, which are not typically known and also vary with time. In such time-varying environments, the offloading policy must adapt to the environment. Reinforcement learning (RL) [10] is a natural choice to design such adaptive policies and has already been successfully applied in various optimization problems arising in MEC [11]–[13].

Although RL has achieved considerable success in various application domains including communication networks, this success is generally achieved by using deep neural networks to model the policy and the value function. Such deep RL algorithms require considerable computational power and time to train, are notoriously brittle to the choice of hyper-parameters, may not transfer well from simulation to the real-world, and give policies which are difficult to interpret. These features make them impractical to be deployed on the edge nodes to continuously adapt to the changing network conditions.

For the aforementioned reasons, rather than using general purpose deep RL algorithms, in this paper we design a node-overload protection scheme that uses a recently proposed low-complexity RL algorithm called SALMUT (Structure-Aware Learning for Multiple Thresholds) [14]. SALMUT exploits the structure of the optimal policy, requires considerably fewer computational resources to train and provides policies which are easy to interpret. We compare the performance of deep RL algorithms with SALMUT in a variety of scenarios which are

motivated by real world deployments. Our experiments show that SALMUT performs as well as the state-of-the-art deep RL algorithms such as PPO [15] and A2C [16] but requires an order of magnitude less time to train and provides policies which are easy to interpret.

The rest of the paper is organized as follows. We present the system model and problem formulation in Sec. II. Then, we present a dynamic programming decomposition for the case of time-homogeneous statistics for the arrival process in Sec. III. Then, we present the structure aware RL algorithm (SALMUT) proposed in [14] for our model in Sec. IV. Finally we conduct a detailed experimental study to compare the performance of SALMUT with other state-of-the-art RL algorithms in Sec. V.

II. Model and Problem Formulation

A. System model

A simplified mobile edge computing (MEC) system consists of an edge server and several mobile users accessing that server. Mobile users independently generate service requests according to a Poisson process. The rate of requests and the number of users may also change with time. The edge server takes CPU resources to serve each request from mobile users. The request is buffered in a queue before it is served. When a new request comes, the server has the option to offload the request. The mathematical model of the edge server and the mobile users is presented below.

1) Edge server: Let $X_t \in \{0, 1, \dots, \mathsf{X}\}$ denote the number of service requests buffered in the queue, where X denotes the size of the buffer. Let $L_t \in \{0, 1, \dots, \mathsf{L}\}$ denote the CPU load at the server where L is the capacity of the CPU. We assume that the CPU has k cores.

We assume that the requests arrive according to a (potentially time-varying) Poisson process with rate λ. If a new request arrives when the buffer is full, the request is offloaded to another server. If a new request arrives when the buffer is not full, the server has the option to either accept or offload the request.

The server can process up to a maximum of k requests from the head of the queue. Processing each request requires CPU resources for the duration for which the request is being served. The required CPU resources is a random variable $R \in \{1, \dots, \mathsf{R}\}$ with probability mass function P. The realization of R is not revealed until the server starts working on the request. The duration of service is exponentially distributed random variable with rate μ.

Let $\mathcal{A} = \{0, 1\}$ denote the action set. Here $A_t = 1$ means that the server decides to offload the request while $A_t = 0$ means that the server accepts the request.

2) Traffic model for mobile users: We consider multiple models for traffic. Let N denote the total number of users accessing the server.

- **Scenario 1:** All users generate requests according to the same rate λ and the rate does not change over time.
- **Scenario 2:** All users generate requests according rate λ_{M_t}, where M_t is a global state which changes over time.
- **Scenario 3:** Each user n has a state $M_t^n \in \{1, \dots, \mathsf{M}\}$. When the user n is in state m, it generates requests according to rate λ_m. The state M_t^n changes over time.
- **Time-varying users:** In each of the scenarios above, we can consider the case when the number of users is not fixed and changes over time. We call them Scenario 4, 5, and 6 respectively.

3) Cost and the optimization framework: The system incurs three types of a cost:

- a holding cost of h per unit time when a request is buffered in the queue but is not being served.
- a running cost of $c(\ell)$ per unit time for running the CPU at a load of ℓ.
- a penalty of $p(\ell)$ for offloading a packet at CPU load ℓ.

We combine all these costs in a cost function

$$\rho(x, \ell, a) = h[x - k]^+ + c(\ell) + p(\ell)\mathbb{1}\{a = 1\}, \qquad (1)$$

where $[x]^+$ is a short-hand for $\max\{x, 0\}$ and $\mathbb{1}\{\cdot\}$ is the indicator function. Note that to simplify the analysis, we have assumed that the server always serves $\min\{X_t, k\}$ requests. It is assumed that $c(\ell)$ and $c(\ell) + p(\ell)$ are increasing in ℓ.

Whenever a new request arrives, the server uses a memoryless policy $\pi\colon \{0, 1, \dots, \mathsf{X}\} \times \{0, 1, \dots, \mathsf{L}\} \to \{0, 1\}$ to choose an action

$$A_t = \pi_t(X_t, L_t).$$

The performance of a policy π starting from initial state (x, ℓ) is given by

$$V^\pi(x, \ell) = \mathbb{E}\left[\int_0^\infty e^{-\alpha t}\rho(X_t, L_t, A_t)dt \;\middle|\; X_0 = x, L_0 = \ell\right], \qquad (2)$$

where $\alpha > 0$ is the discount rate and the expectation is with respect to the arrival process, CPU utilization, and service completions.

The objective is to minimize the performance (2) for the different traffic scenarios listed above. We are particularly interested in the setting where the arrival rate and potentially other components of the model such as the resource distribution are not known to the system designer and change during the operation of the system.

B. Solution framework

When the model parameters (λ, N, μ, P, k) are known and time-homogeneous, the optimal policy π can be computed using dynamic programming. However, in a real system, these parameters may not be known, so we are interested in developing a reinforcement learning algorithm which can learn the optimal policy based on the observed per-step cost.

In principle, when the model parameters are known, Scenarios 2 and 3 can also be solved using dynamic programming. However, the state of such dynamic programs will include the state M_t of the system (for Scenario 2) or the states $(M_t^n)_{n=1}^N$ of all users (for Scenario 3). Typically, these states change at a slow time-scale. So, we will consider reinforcement learning algorithms which do not explicitly keep track of the states of the user and check if the algorithm can adapt quickly whenever the arrival rates change.

III. Dynamic programming to identify optimal admission control policy

When the arrival process is time-homogeneous, the process $\{X_t, L_t\}_{t\geq 0}$ is a finite-state continuous-time Markov decision process (MDP) controlled through $\{A_t\}_{t\geq 0}$. To specify the controlled transition probability of this MDP, we consider the following two cases.

First, if there is a new arrival at time t, then

$$\mathbb{P}(X_t = x', L_t = \ell' \mid X_{t^-} = x, L_{t^-} = \ell, A_t = a) = \begin{cases} P(\ell' - \ell), & \text{if } x' = x+1 \text{ and } a = 0 \\ 1, & \text{if } x' = x,\ \ell' = \ell, \text{ and } a = 1 \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

We denote this transition function by $q_+(x', \ell' | x, \ell, a)$. Note that the first term $P(\ell' - \ell)$ denotes the probability that the accepted request required $(\ell' - \ell)$ CPU resources.

Second, if there is a departure at time t,

$$\mathbb{P}(X_t = x', L_t = \ell' \mid X_{t^-} = x, L_{t^-} = \ell) = \begin{cases} P(\ell - \ell'), & \text{if } x' = [x-1]^+ \\ 0, & \text{otherwise.} \end{cases} \quad (4)$$

We denote this transition function by $q_-(x', \ell' | x, \ell)$. Note that there is no decision to be taken at the completion of a request, so the above transition does not depend on the action. In general, the reduction in CPU utilization will correspond to the resources requested by the request whose service was completed. However, keeping track of those resources would mean that we would need to expand the state and include $(R_1, \ldots, R_k)$ as part of the state, where R_i denotes the resources required by the request which is being processed by CPU i. In order to avoid such an increase in state dimension, we assume that when a request is completed, CPU utilization reduces by amount $\ell - \ell'$ with probability $P(\ell - \ell')$.

We combine (3) and (4) into a single controlled transition probability function from state (x, ℓ) to state (x', ℓ') given by

$$p(x', \ell' \mid x, \ell, a) = \frac{\lambda}{\lambda + \min\{x, k\}\mu} q_+(x', \ell' \mid x, \ell, a) + \frac{\min\{x, k\}\mu}{\lambda + \min\{x, k\}\mu} q_-(x', \ell' \mid x, \ell). \quad (5)$$

Let $\nu = \lambda + k\mu$ denote the uniform upper bound on the transition rate at the states. Then, using the *uniformization technique* [7], [8], we can convert the above continuous time discounted cost MDP into a discrete time discounted cost MDP with discount factor $\beta = \nu/(\alpha + \nu)$, transition probability matrix $p(x', \ell' | x, \ell, a)$ and per-step cost

$$\bar{\rho}(x, \ell, a) = \frac{1}{\alpha + \nu} \rho(x, \ell, a)$$

Therefore, we have the following.

Theorem 1 *Consider the following dynamic program*

$$V(x, \ell) = \min\{Q(x, \ell, 0), Q(x, \ell, 1)\} \quad (6)$$

where

$$Q(x,\ell,0) = \frac{1}{\alpha+\nu}\big[h[x-k]^+ + c(\ell)\big] + \beta\Big[\frac{\lambda}{\lambda + \min\{x,k\}\mu} \sum_{r=1}^{\mathsf{R}} P(r) V([x+1]_{\mathsf{X}}, [\ell + r]_{\mathsf{L}}) + \frac{\min\{x,k\}\mu}{\lambda + \min\{x,k\}\mu} \sum_{r=1}^{\mathsf{R}} P(r) V([x-1]^+, [\ell - r]^+)\Big]$$

and

$$Q(x,\ell,1) = \frac{1}{\alpha+\nu}\big[h[x-k]^+ + c(\ell) + p(\ell)\big] + \beta \frac{\min\{x,k\}\mu}{\lambda + \min\{x,k\}\mu} \sum_{r=1}^{\mathsf{R}} P(r) V([x-1]^+, [\ell - r]^+)$$

where $[x]_{\mathsf{B}}$ *denotes* $\min\{x, \mathsf{B}\}$.

Let $\pi(x, \ell) \in \mathcal{A}$ denote the argmin the right hand side of (6)*. Then, the time-homogeneous policy $\pi(x, \ell)$ is optimal for the original continuous-time optimization problem.*

Proof The equivalence between the continuous and discrete time MDPs follows from the uniformization technique [7], [8]. The optimality of the time-homogeneous policy π follows from the standard results for MDPs [9]. ∎

Thus, for all practical purposes, the decision maker has to solve a discrete-time MDP, where he has to take decisions at the instances when a new request arrives. In the sequel, we will ignore the $1/(\alpha + \nu)$ term in front of the per-step cost and assume that it has been absorbed in the constant h, and the functions $c(\cdot)$, $p(\cdot)$.

When the system parameters are known, the above dynamic program can be solved using standard techniques such as value iteration, policy iteration, or linear programming. However, in practice, the system parameters may slowly change over time. Therefore, instead of pursuing a planning solution, we consider reinforcement learning solutions which can adapt to time-varying environments.

IV. Structure-aware reinforcement learning

Although, in principle, the optimal admission control problem formulated above can be solved using deep RL algorithms, such algorithms require significant computational resources to train, are brittle to the choice of hyperparameters, and generate policies which are difficult to interpret. For these reasons, we investigate an alternate class of RL algorithms which circumvents these limitations.

A. Structure of the optimal policy

We first establish basic monotonicity properties of the value function and the optimal policy.

Proposition 1 *The value function satisfies the following:*

- *For a fixed queue length x, the value function is weakly increasing in the CPU utilization ℓ.*
- *For $k = 1$ and a fixed CPU utilization ℓ, the value function is weakly increasing in the queue length x.*

Proposition 2 *The optimal policy π satisfies the following:*

- *For a fixed queue length x, if it is optimal to reject a request at CPU utilization ℓ, then it is optimal to reject a request at all CPU utilizations $\ell' > \ell$.*
- *For $k = 1$ and for a fixed CPU utilization ℓ, if it is optimal to reject a request at queue length $x \geq k$, then it is optimal to reject a request at all queue lengths $x' > x$.*

Both results follow from standard monotonicity arguments for MDPs [9]. The details are omitted due to limited space.

Remark 1 We are able to establish monotonicity in the queue length in Proposition 1 under the restriction that $k = 1$. Since Proposition 2 depends on Proposition 1, a similar restriction applies in that case as well. When $k > 1$, we are unable to use the standard monotonicity arguments of [9] because for a fixed ℓ, the transition matrix p is *not* stochastically monotone in x. However, simulations suggest that both Propositions 1 and 2 continue to hold for $k > 1$. So, we believe that the restriction $k = 1$ is a limitation of our proof technique and conjecture that the result will hold in general as well.

B. The SALMUT algorithm

Proposition 2 shows that the optimal policy can be represented by a threshold vector $\tau = (\tau(x))_{x=0}^{\mathsf{X}}$, where $\tau(x) \in \{0, \dots, \mathsf{L}\}$ is the smallest value of the CPU utilization such that it is optimal to accept the packet for CPU utilization less than or equal to $\tau(x)$ and reject it for utilization greater than $\tau(x)$. Furthermore, (for $k = 1$ and also for general k as per the conjecture in Remark 1), the thresholds $\tau(x)$ are decreasing in x.

The SALMUT algorithm was proposed in [14] to exploit a similar structure in admission control for multi-class queues. It was proposed for the average cost setting but, as explained below, it generalizes to the discounted cost setting as well.

A threshold-based policy π_τ is a parameterized policy with the parameters $(\tau(x))_{x=0}^{\mathsf{X}}$ taking values in $\{0, \dots, \mathsf{L}\}^{\mathsf{X}+1}$. The key idea behind SALMUT is that, instead of deterministic threshold-based policies, we consider a random policy parameterized with parameters taking value in the compact set $[0, \mathsf{L}]^{\mathsf{X}+1}$. Then, for any state (x, ℓ), the randomized policy π_τ chooses action $a = 0$ with probability $f(\tau(x), \ell)$ and chooses action $a = 1$ with probability $1 - f(\tau(x), \ell)$, where $f(\tau(x), \ell)$ is any continuous decreasing function w.r.t ℓ, which is differentiable in its first argument, e.g., the sigmoid function

$$f(\tau(x), \ell) = \frac{\exp((\tau(x) - \ell)/T)}{1 + \exp((\tau(x) - \ell)/T)}, \tag{7}$$

where $T > 0$ is a hyper-parameter (often called "temperature"). Let $p^{(\tau)}(x', \ell' | x, \ell)$ denote the transition probability matrix under policy π_τ, i.e.,

$$\begin{aligned} p^{(\tau)}(x', \ell' | x, \ell) &= f(\tau(x), \ell) p(x', \ell' | x, \ell, 0) \\ &\quad + (1 - f(\tau(x), \ell)) p(x', \ell' | x, \ell, 1). \end{aligned} \tag{8}$$

Since $\nabla p^{(\tau)}(x', \ell' | x, \ell) = \nabla f(\tau(x), \ell)[p(x', \ell' | x, \ell, 0) - p(x', \ell' | x, \ell, 1)]$, where the gradients are with respect to τ, an unbiased estimator of $\nabla p^{(\tau)}(\cdot | x, \ell)$ is given by

$$(-1)^a \nabla f(x, \tau(\ell)), \quad \text{where } a \sim \pi_\tau(\cdot | x, \ell). \tag{9}$$

Fix an initial state (x_0, ℓ_0) and let $J(\tau)$ denote the performance of policy π_τ when starting from the initial state (x_0, ℓ_0). Now, from the policy gradient theorem [10], we know that

$$\nabla J(\tau) = \sum_{x=0}^{\mathsf{X}} \sum_{\ell=0}^{\mathsf{L}} \mu(x, \ell) \nabla Q(x, \ell; \tau)$$

where $\mu(x, \ell)$ is the occupancy measure on the states starting from the initial state (x_0, ℓ_0) and

$$\begin{aligned} \nabla Q(x, \ell; \tau) = \sum_{x'=0}^{\mathsf{X}} \sum_{\ell'=0}^{\mathsf{L}} \nabla p^{(\tau)}(x', \ell' | x, \ell) \\ \times \sum_{a \in \mathcal{A}} \pi_\tau(a | x, \ell) Q(x, \ell, a). \end{aligned}$$

Therefore, an unbiased estimator of $\nabla J(\tau)$ is given by

$$\nabla p^{(\tau)}(x', \ell' | x, \ell) Q(x, \ell, a), \quad \text{where } a \sim \pi_\tau(\cdot | x, \ell). \tag{10}$$

Combining (9) with (10), we get that

$$(-1)^a \nabla f(\tau(x), \ell) \big[\bar{\rho}(x, \ell, a) + \beta V(x', \ell')\big], \tag{11}$$

where $a \sim \pi_\tau(\cdot | x, \ell)$ is an unbiased estimator of $\nabla J(\tau)$.

Thus, we can use the standard two time-scale Actor-Critic algorithm [10] to simultaneously learn the policy parameters τ and the action-value function Q as follows. We start with an initial guess Q_0 and τ_0 for the action-value function and the optimal policy. Then, we update the action-value function using temporal difference learning:

$$\begin{aligned} Q_{n+1}(x, \ell, a) = Q_n(x, \ell, a) + b_n^1 \big[\bar{\rho}(x, \ell, a) \\ + \beta \min_{a' \in A} Q_n(x', \ell', a') - Q_n(x, \ell, a)\big], \end{aligned} \tag{12}$$

and update the policy parameters using stochastic gradient descent while using (11) as the unbiased estimator of $\nabla J(\tau)$:

$$\begin{aligned} \tau_{n+1}(x) = \mathrm{Proj}\big[\tau_n(x) + b_n^2 (-1)^a \nabla f(\tau(x), \ell) \big[\bar{\rho}(x, \ell, a) \\ + \beta \min_{a' \in \mathcal{A}} Q(x', \ell', a')\big]\big], \end{aligned} \tag{13}$$

where Proj is a projection operator which clips the values to the interval $[0, \mathsf{L}]$ and $\{b_n^1\}_{n \geq 0}$ and $\{b_n^2\}_{n \geq 0}$ are learning rates which satisfy the standard conditions on two time-scale learning: $\sum_n b_n^k = \infty$, $\sum_n (b_n^k)^2 < \infty$, $k \in \{1, 2\}$, and $\lim_{n \to \infty} b_n^2 / b_n^1 = 0$.

Note that the update of $\tau(x)$ only changes the value of the threshold at the current queue length x which might give a threshold vector τ which is not monotone decreasing. So, we also update the values of other components of τ to make it monotone decreasing as follows:

$$\tau_{n+1}(y) = \begin{cases} \max\{\tau_{n+1}(x), \tau_{n+1}(y)\}, & \text{if } y \leq x \\ \min\{\tau_{n+1}(x), \tau_{n+1}(y)\}, & \text{if } y \geq x \end{cases} \tag{14}$$

where x is the queue state at the current time.

Algorithm 1: Two time-scale SALMUT algorithm

Result: τ
Initialize Q-values $\forall x, \forall \ell, \forall a,\ Q(x, \ell, a) \leftarrow 0$
Initialize threshold vector $\forall x,\ \tau(x) \leftarrow \text{rand}(0, \mathsf{L})$
Initialize start state $(x, \ell) \leftarrow (x_0, \ell_0)$
while TRUE **do**
  if EVENT == ARRIVAL **then**
    Choose action a according to Eq. (8)
    Update Q-value $Q(x, \ell, a)$ according to Eq. (12)
    Update threshold τ using Eqs. (13) and (14)
    $(x, \ell) \leftarrow (x', \ell')$
  end
end

The complete algorithm is presented in Algorithm 1. Similar to [14, Theorem 2], we can show that under standard technical assumptions [17], the two time-scale SALMUT algorithm described above converges almost surely to a τ^* such that $\nabla J(\tau^*) = 0$.

V. Numerical experiments

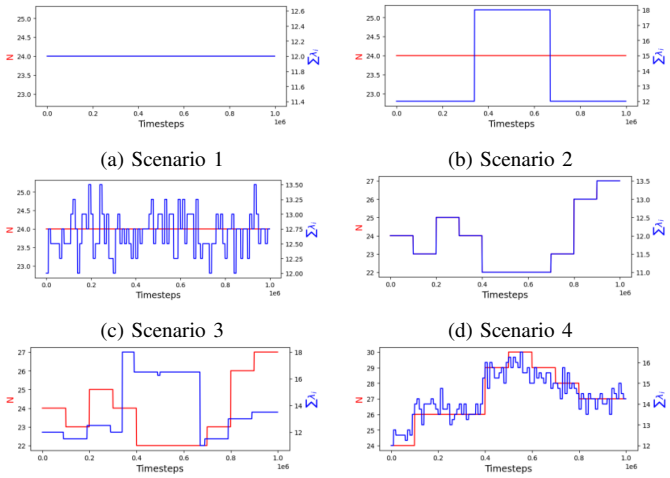
In this section, we present detailed numerical experiments to evaluate the proposed reinforcement learning algorithm on various scenarios described in Sec. II-A.

We consider an edge server with buffer size $\mathsf{X} = 20$, CPU capacity $\mathsf{L} = 20$, $k = 2$ cores, service-rate $\mu = 6.0$ for each core, holding cost $h = 0.12$. The CPU capacity is discretized into 20 states for utilization $0-100\%$, with $\ell = 0$ corresponding to a state with CPU load $\ell \in [0\% - 5\%)$, and so on. The CPU running cost is $c(\ell) = 10$ for $\ell \geq 18$, which correspond to a high cost for an overloaded system, $c(\ell) = -0.2$ for $6 \leq \ell \leq 17$, a positive reinforcement for being in the optimal CPU range, and $c(\ell) = 0$ otherwise. The offload penalty is $p = 1$ for $\ell \geq 3$ and $p = 10$ for $\ell < 3$ to discourage offloading when the system is idle. The probability mass function of resources requested per request $P(r) = 0.6$, when $r = 1$, and $P(r) = 0.4$ for r = 2.

Rather than simulating the system in continuous-time, we simulate the equivalent discrete-time MDP by generating the next event (arrival or departure) using a Bernoulli distribution with probabilities and costs described in Sec. III. We assume that the parameter $1/(\alpha + \nu)$ in (6) has been absorbed in the cost function. We assume that the discrete time discount factor $\beta = \alpha/(\alpha + \nu)$ equals 0.95.

A. Simulation scenarios

We consider a number of traffic scenarios which are increasing in complexity and closeness to the real-world setting. Each scenario runs for a horizon of $T = 10^6$. The scenarios capture variation in the transmission rate and the number of users over time. Due to space limitations, we omit the details on how these scenarios were generated and simply provide their realization in Fig. 1.



(a) Scenario 1 (b) Scenario 2

(c) Scenario 3 (d) Scenario 4

(e) Scenario 5 (f) Scenario 6

Fig. 1. The evolution of λ and N for the different scenarios that we described. In scenarios 1 and 4, λ and N overlap in the plots.

B. The RL algorithms

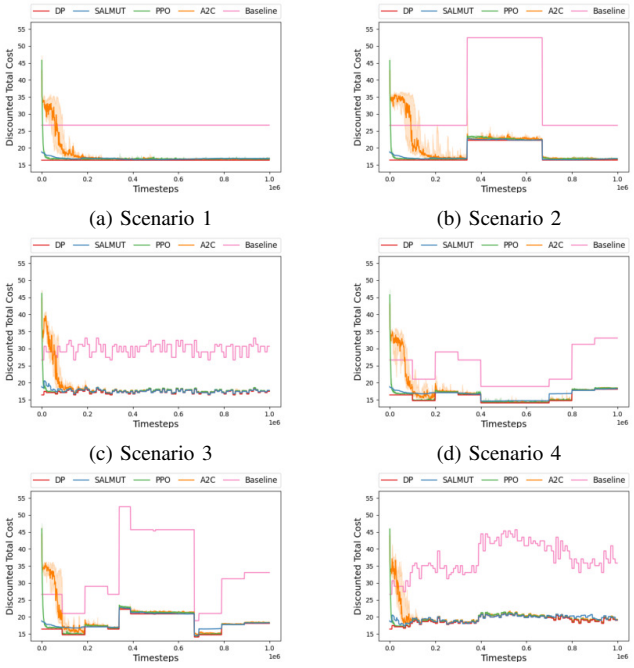
For each scenarios, we compare the performance of the following policies

1) Dynamic Programming (DP), which computes the optimal policy using Theorem 1.
2) SALMUT, as described in Sec. IV-B
3) PPO (Proximal Policy Optimization) [15], which is a family of trust region policy gradient method and optimizes a surrogate objective function using stochastic gradient ascent.
4) A2C (Advantage Actor-Critic) [16], which is a two time-timescale learning algorithms where the critic estimates the value function and actor updates the policy distribution in the direction suggested by the critic.
5) Baseline, which is a fixed-threshold based policy, where the node accepts requests when $\ell < 18$ (non-overloaded state) and offloads requests otherwise. Such static policies are currently deployed in many real-world systems.

C. Results and Discussions

For each of the algorithm described above, we train SALMUT, PPO, and A2C for 10^6 steps. The performance of each algorithm is evaluated every 10^3 steps using independent rollouts of length $H = 1000$ for 100 different random seeds. The experiment is repeated for the 10 sample paths and the median performance with an uncertainty band from the first to the third quartile are plotted in Fig. 2.

For Scenario 1, all RL algorithms (SALMUT, PPO, A2C) converge to a close-to-optimal policy relatively quickly (with A2C being slightly slower) and remain stable after convergence. Since all policies converge quickly, they are also able to adapt quickly in Scenarios 2–6 and keep track of the time-varying arrival rates and number of users. There are small differences in the peformance of the RL algorithms, but these are minor. Note that, in contrast, the baseline policy of offloading when the server is overloaded performs poorly.



(a) Scenario 1 (b) Scenario 2

(c) Scenario 3 (d) Scenario 4

(e) Scenario 5 (f) Scenario 6

Fig. 2. Performance of RL algorithms for different scenarios.

The main difference among these three RL algorithms is the training time and interpretability of policies. We ran our experiments on a Intel(R) Xeon(R) Gold 6148 CPU @ 2.40GHz server. Referring to Table I, SALMUT is about 28 times faster to train than PPO and 17 times faster than A2C.

By construction, SALMUT searches for (randomized) threshold based policies. For example, for Scenario 1, SALMUT converges to the policy shown in Fig. 3. It is easy for a network operator to interpret such threshold based strategies and decide whether to deploy them or not. In contrast, in deep RL algorithms such as PPO and A2C, the policy is parameterized using a neural network and it is very difficult to simply visualize the learned weights of such a policy and decide whether the resultant policy is reasonable. Thus, by leveraging on the threshold structure of the optimal policy, SALMUT is able to learn faster and at the same time provide threshold based policies which are easier to interpret.

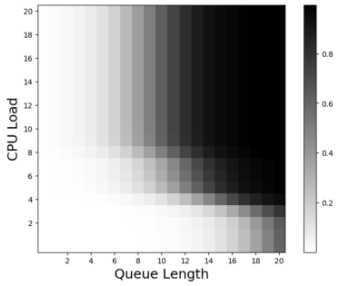


Fig. 3. The converged policy using SALMUT along one of the sample paths. The colorbar represents the probability of the offloading action.

VI. Conclusion

In this paper we considered a single node optimal policy for overload protection on the edge server in a time varying environment. We proposed a RL approach which exploits a randomized policy with compact value set. We showed that the policy based on SALMUT is completely characterized by the optimum value of the CPU utilization, is predictable, and easy to interpret. It performs as well as the standard deep RL approach but has a far better computational and storage complexity. This algorithm can be suitably deployed in real systems for online training. Future work can be extended to similar problems by controlling multi-edge nodes within a cluster or inter-cluster. Heterogeneous applications running in the cluster might also be considered.

TABLE I
Training time of RL algorithms

Algorithm	Mean Time (s)	Std-dev (s)
SALMUT	95.67	3.29
PPO	2673.17	23.33
A2C	1677.33	9.99

Acknowledgments

This research was supported by Mitacs Grant IT10968.

References

[1] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
[2] J. Liu, Y. Mao, J. Zhang, and K. B. Letaief, "Delay-optimal computation task scheduling for mobile-edge computing systems," in *Int. Symp. Inform. Theory (ISIT)*, 2016, pp. 1451–1455.
[3] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, 2017.
[4] D. Van Le and C.-K. Tham, "Quality of service aware computation offloading in an ad-hoc mobile cloud," *IEEE Trans. Veh. Technol.*, vol. 67, no. 9, pp. 8890–8904, 2018.
[5] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, 2018.
[6] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge computing based on Markov decision process," *IEEE/ACM Trans. Netw.*, vol. 27, no. 3, pp. 1272–1288, Jun. 2019.
[7] A. Jensen, "Markoff chains as an aid in the study of Markoff processes," *Scandinavian Actuarial Journal*, vol. 1953, no. sup1, pp. 87–91, 1953.
[8] R. A. Howard, *Dynamic Programming and Markov Processes*. The M.I.T. Press, 1960.
[9] M. Puterman, *Markov decision processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 1994.
[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT Press, 2018.
[11] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J*, vol. 6, no. 3, pp. 4005–4018, 2018.
[12] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online offloading in wireless powered mobile-edge computing networks," *arXiv:1808.01977*, 2018.
[13] J. Wang, J. Hu, G. Min, W. Zhan, Q. Ni, and N. Georgalas, "Computation offloading in multi-access edge computing using a deep sequential model based on reinforcement learning," *IEEE Commun. Mag.*, vol. 57, no. 5, pp. 64–69, 2019.
[14] A. Roy, V. Borkar, A. Karandikar, and P. Chaporkar, "Online reinforcement learning of optimal threshold policies for Markov decision processes," *arXiv:1912.10325*, 2019.
[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.
[16] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," 2017.
[17] V. S. Borkar, "Stochastic approximation with two time scales," *Systems & Control Letters*, vol. 29, no. 5, pp. 291–294, 1997.