ACRE: Actor Critic Reinforcement Learning for Failure-Aware Edge Computing Migrations

Marie Siew, Shikhar Sharma, Carlee Joe-Wong, *Senior Member, IEEE*
Electrical and Computer Engineering, Carnegie Mellon University.
msiew@andrew.cmu.edu; shikhar2@andrew.cmu.edu; cjoewong@andrew.cmu.edu

***Abstract*—In edge computing, users' service profiles are migrated in response to user mobility, to minimize the user-experienced delay, balanced against the migration cost. Due to imperfect information on transition probabilities and costs, reinforcement learning (RL) is often used to optimize service migration. Nevertheless, current works do not optimize service migration in light of occasional server failures. While server failures are rare, they impact the smooth and safe functioning of latency sensitive edge computing applications like autonomous driving and real-time obstacle detection, because users can no longer complete their computing jobs. As these failures occur at a low probability, it is difficult for RL algorithms, which are data and experience driven, to learn an optimal service migration policy for both the usual and rare event scenarios. Therefore, we propose an algorithm ImACRE, which integrates importance sampling into actor critic reinforcement learning, to learn the optimal service profile and backup placement policy. Our algorithm uses importance sampling to sample rare events in a simulator, at a rate proportional to their contribution to system costs, while balancing service migration trade-offs between delay and migration costs, with failure costs, backup placement and migration costs. We use trace driven experiments to show that our algorithm gives cost reductions in the event of failures.**

***Index Terms*—Edge Computing, Service migration, Resilient Resource Allocation**

I. Introduction

Mobile (or multi-access) edge computing (MEC) enables computationally intensive and latency sensitive mobile applications such as real time image processing, augmented reality, interactive gaming, etc, on resource-constrained mobile devices. In MEC, computing resources such as a cluster of servers are placed geographically close to end-users, for example at base stations or WiFi access points [1], [2]. Users can offload their computationally intensive jobs to the edge servers. Their geographical proximity helps avoid the wide-area network delay costs experienced during cloud offloading [2]. User mobility across geographical areas then presents a challenge towards task offloading [3]–[5]. As a user moves across coverage areas, keeping its service profile at its original location may lead to higher user-experienced latency, due to the increased network distance. Therefore, service migration has been proposed as a strategy [4], [5]. Nevertheless, constant service migration in response to user mobility causes additional energy and operational expenditure. Hence, service migration while balancing the delay-cost tradeoff, under various scenarios, is a widely studied problem [3], [4], [6]–[9].

Marie Siew is supported by the SUTD Presidential Postdoctoral Fellowship, under the Singapore Teaching and Academic Research Talent Scheme (START).

Few migration works, however, account for another challenge: the **resilience** of edge computing systems to rare but serious events like server failures. They occur due to reasons such as system overload, hardware failures, or malicious attacks, and they can be costly: the average cost of outage at a cloud data center, for example, has increased to $\$740000$ in 2016 [10]. In comparison to the cloud, failures at edge servers are even more likely, as edge servers are distributed geographically, making their management and maintenance more challenging: for example, edge servers do not have advanced heat management or outage support systems such as direct liquid cooling, fire suppressing gaseous systems and fully duplicated electrical lines with transfer switches [11].

Edge server failures may have a particularly sizeable impact as many edge computing applications are latency-sensitive and safety-critical, such as the Internet of vehicles, augmented reality, and video processing [1]. If the user's service profile is migrated to an access point that then experiences a server failure, its job is not able to be completed, jeopardizing the smooth and safe functioning of edge applications, such as autonomous driving and real time obstacle detection.

In this work we address the following question: *How can the network operator make edge computing systems resilient and adaptable to such failures?* We propose the use of backups. When backup services are placed at a different edge server from the primary service, they can take over if the primary service's edge server fails, allowing the application to continue functioning. However, the number of potential migration paths of the primary and backup services grows at least exponentially with the number of access points and timeslots. Furthermore, the network operator has a lack of knowledge of the user mobility distributions and the corresponding costs resulting from mobility. Therefore, Markov decision processes and reinforcement learning (RL) have been proposed as a method to solve the service migration problem [3], [6], [12], [13]. While an RL-based backup placement solution has been used for virtual network function instance placement [14], to the best of our knowledge RL-based backups have not been used for edge computing services.

At the same time, these rare events and failures occur at a low probability, making it difficult to jointly learn an optimal policy for the placement of service profiles and backups, for both the usual and rare event scenarios. This is because RL relies on past reward data. The low probabilities of rare events may make the learner miss the importance of rare events, impacting the training of the learning algorithm [15]. For example, a policy trained on data with few failures might conclude that always placing backups is not worth the storage cost. The work on RL-based backups for virtual network function instance placement [14] did not consider failures as rare events having low probabilities. Therefore, in this paper we introduce an importance sampling based actor critic reinforcement learning framework, for *resilient* edge computing service migration. Prior work has proposed using importance sampling to over-sample rare events for policy evaluation [15], [16]. However, this involves estimating the value function given a fixed policy, and does not consider learning the optimal policy π^* which entails policy changes. [17] proposed re-sampling events to accelerate learning for rare states, but they can only replay historical trajectories instead of learning on new ones. Our prior work [18] proposed an importance sampling q-learning algorithm to deal with rare events. In this work, we extend our prior work to propose an actor-critic version.

- We present a service migration optimization problem in light of user mobility across coverage areas, that introduces the modelling of server failures as rare events.
- We propose an importance sampling based actor critic algorithm **ImACRE**, which learns the optimal policy for the overall system with true rare event probabilities. In our algorithm, we sample rare events at a rate proportional to their contribution to the value function (i.e. to system costs), in a simulator training setup. Error correction is done through the integration of importance sampling weights on the advantage function.
- We perform trace driven simulations which show that our algorithm attains cost reductions in the event of failures.

II. System Model and Problem Formulation

A. Service Migration Model

We consider an MEC system with N *access points*, such as a base station or wifi-access point. Each access point is equipped with a server, to which users can offload their time-sensitive computing jobs, and is associated with a *region* in which it is the closest access point. Users are mobile and move across regions at different times of the day. For example, office workers may move to the city districts for work during morning rush hours. This changes their nearest access point. The user's location at time t is represented by $l_u(t)$. The user's mobility pattern follows a probability distribution $m(l'_u|l_u)$. At each time-slot t, the user sends a service request to the network operator. The service profile for the applications run on the user's mobile devices will be placed at virtual machines or containers [4] at the access point's servers. To maintain a lower job delay for the time-sensitive computing applications, services are pre-migrated to nearby access points, thus ensuring they will have finished migrating by the time the

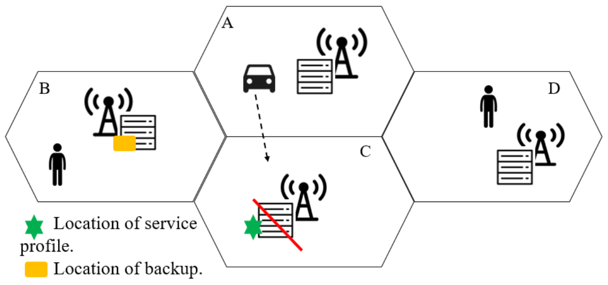


Fig. 1. User mobility, and the placement of service profiles and backups in light of potential server failures in edge computing.

user moves to a new region. The location of the user's service at time t is represented by $l_s(t)$, and due to pre-migrations the network operator makes this placement decision in advance, before knowledge of the user location at time t is known.

We let $d^{\text{comm}}_{l_u(t),j}$ denote the communication delay the user faces when it offloads its computation job, given that the user's current location is l_u and its service is placed at access point j. Therefore, the **communication delay cost** the user faces at time t would be

$$D(t) = \sum_j d^{\text{comm}}_{l_u(t),j} \mathbb{1}_{\{l_s(t)=j\}}, \tag{1}$$

where $\mathbb{1}_{\{l_s(t)=j\}}$ is the indicator function on the service location l_s. The communication delay $d^{\text{comm}}_{l_u(t),j}$ consists of the access latency of job uploading to the user's associated access point, and the transfer latency of forwarding the job to the edge server's location if the service is placed at another access point (i.e. $l_u \neq j$). Therefore, the delay is a function of the distance between the location of the user and the access point j [4].

Greedily migrating service profiles along with the user may be suboptimal, because doing so incurs a service **migration cost** for the network operator, requiring the operator to balance this cost with the communication delay; user mobility may also be unknown in advance. The migration cost includes the operational and energy costs on network devices like routers and switches [4], and hence is a function of the distance across two access points. The cost of migrating the service profile from access point i to j is denoted by the variable m_{ij}. Hence, the migration cost at time t, $M(t)$, can be expressed as

$$M(t) = \sum_{i\in A}\sum_{j\in A} m_{ij} \mathbb{1}_{\{l_s(t-1)=i\}} \mathbb{1}_{\{l_s(t)=j\}}, \tag{2}$$

where $\mathbb{1}_{\{l_s(t-1)=i\}}$ is the indicator function on the service profile's previous location.

B. Failure Model and Backups

In reality, rare events (system anomalies) such as server failures or shutdowns occur. Such events will impact the quality of service the user receives, as the user is not able to have his or her job computed on time, or at all, impacting the safe and smooth functioning of edge applications and jeopardizing the low latency benefits of edge computing.

To take into account the potential failures and reduce the costs experienced by the user, a backup of the user's service profile can be placed in the system. This way, even if the server $l_s(t)$ at which the user's service is placed at experiences a failure, the user can still offload its job. As seen in Fig. 1, the vehicle moves from A to C, and its service profile was pre-emptively placed at C. Despite a server failure at C, job offloading can occur, as a backup service profile was placed at B. We formulate a Markov Decision Process (MDP) in which the system state $s(t) \in S$ is $(l_u(t), l_s(t), f_{ind}(t), b_{ind}(t))$. l_u is the user's current location, l_s is the location of the service, f_{ind} is an indicator on whether the service is placed at a location with a server failure, and b_{ind} indicates whether there currently is a backup in the system. At each timeslot t, the network operator takes action $a(t) = \{l_s(t+1), b_u(t+1)\}$ pre-emptively, before the user moves to the next location $l_u(t+1)$. This involves determining the placement location of the service $l_s(t+1)$, and the position of the backup in the system $b_u(t+1) \in \{0, 1, ...N, \text{no backup}\}$, where no backup refers to the choice of there not being a backup in the system.

Backup costs. There is a cost $B(t)$ incurred by *storing* the backup at a server. This cost is a sum of the backup migration cost and the backup storage cost, as it takes up space and prevents other content from being stored.

$$\begin{aligned} B(t) = &\sum_{j \in A} \rho_j \mathbb{1}_{\{b_u(t)=j\}} \\ &+ \sum_{i \in A} \sum_{j \in A} m_{ij} \mathbb{1}_{\{b_u(t-1)=i\}} \mathbb{1}_{\{b_u(t)=j\}}, \end{aligned} \tag{3}$$

where ρ_j is the storage cost at access point j. There is also a **failure cost** $F(t)$ incurred when there is a server failure at the user's service placement location ($f_{ind}(t) = 1$), because the user's task is not served within the task deadline.

$$F(t) = F\mathbb{1}_{\{f_{ind}(t)=1\}}(\mathbb{1}_{\{b_u(t)=\text{no backup}\}} \vee \mathbb{1}_{\{b_u(t)=l_s(t)\}}) \tag{4}$$

The failure cost is incurred when there is no backup in the system ($b_u(t) =$ no backup), or when the backup location $b_u(t)$ equals the main service location $l_s(t)$. Note that the presence of failures themselves, i.e., $f_{ind}(t)$, is independent of the user actions.

We let $\tilde{s} \in \tilde{S}$ refer to the state record without the failure indicator f_{ind}:

$$\tilde{s}(t+1) = (l_u(t+1), l_s(t+1), b_{ind}(t+1)) \in \tilde{S} \tag{5}$$

At each state s, there is a small probability $\epsilon(s)$ (for example $\epsilon(s) = 0.0001$), that in the next timeslot, a server failure occurs, i.e. $f_{ind} = 1$. We define the set of states in which a server failure occurs (state set $T \subset S$) as the *"Rare event states"*. With a larger probability of $1 - \epsilon(s)$, no server failure occurs, i.e. $f_{ind} = 0$. We define these states ($S \backslash T$) as the set of *"Normal States"*. Hence we have

$$s(t+1) = \tilde{s}(t+1) \cup f_{ind}(t+1).$$

The overall state transition probability of the system is

$$p(s'|s,a) = \begin{cases} (1-\epsilon(s))h(\tilde{s}|s,a), & \text{if } s' \notin T \\ \epsilon(s)h(\tilde{s}|s,a), & \text{if } s' \in T, \end{cases} \tag{6}$$

where $h(\tilde{s}|s,a) = m(l'_u|l_u)Pr(a)$, the product of the user's mobility distribution and the probability that the network operator takes a particular action.

In this study we focus on a single user, as our focus is on rare events and failure adaptive resource allocation. We will consider a multi-user system with rare events in future work.

C. *Optimizing the Delay-Cost Trade-off in light of Rare Events*

To optimize the placement and migration of the user's service profile and backup, in light of rare events such as server shutdowns, we study the following problem:

$$\min_{\pi(s)} \mathbb{E}_\pi[\sum_t (w_D D(t) + w_M M(t) + w_B B(t) + F(t))] \tag{7}$$

$$\begin{aligned} \text{s.t. } &(l_u(t), l_s(t), f_{ind}(t), b_{ind}(t)) \sim p(s'|s,a). \\ &\text{Eqs.}(1), (2), (3), (4). \end{aligned} \tag{8}$$

We aim to minimize the expectation (over the policy π) of the sum of the delay, migration, storage and failure costs. The weights w_D, w_M and w_B can be adjusted based on the priority given to the different costs. The decision variable $\pi(s,a) = \{P_r(a(t) = a|s(t) = s)\}$ is the network operator's policy, which indicates the probability it will take each action $a(t) = \{l_s(t+1), b_u(t+1)\}$ (service placement and backup storage decision) given the current state $s(t)$. The first constraint indicates that the state dynamics follows the transition probability in Eq. (6), incorporating rare event transitions, while Eqs. (1), (2), (3), (4) indicate the delay, migration, backup and failure costs respectively.

The network operator may not have information on the mobility distributions and the corresponding latency costs, making it difficult to solve the migration problem in a look-ahead manner. Thus, Markov Decision Processes and Reinforcement Learning have been used to derive service migration policies in the literature [3], [6], [12], [13].Nevertheless, the difficulty in our problem further arises because there are occasional server failures, with a drastically distinct cost from normal events due to their impact on smooth and safe functioning of applications. As these failures occur at a low probability, and because reinforcement learning relies on past reward data, the learner may miss the importance of these rare events and be unable to learn an optimal policy adaptive to server failures. We introduce our solution in the next section.

III. Reinforcement learning in the presence of Server Failures

In this paper, we use an importance sampling - based actor critic reinforcement learning algorithm to learn optimal service migration and backup placement policies, in light of potential server failures.

We are concerned with rare event states if they collectively have a sizeable impact on the value function, meaning they have a sizeable impact on system costs. The value function $V^\pi(s) : S \to \mathbb{R}$ for the policy π is the expected discounted sum of rewards, starting from state s, and following π:

$$V^\pi(s) = \mathbb{E}_{a_t, p(s'|s,a)} \left[\sum_{t=1}^{\infty} \gamma^k r(s(t), a(t), a(t+1)) | s_0 = s \right],$$

where the reward (cost) is $r(s(t), a(t), a(t+1)) = w_D D(t) + w_M M(t) + w_B B(t) + F(t)$ is the per-timeslot cost of our optimization problem, under state transition probability $p(s'|s,a)$. It is the solution to the recursive Bellman equation [19]:

$$V^\pi(s) = \sum_{a \in A} \pi(s,a) \sum_{s' \in S} p(s'|s,a)[r(s,a,s') + \gamma V^\pi(s')].$$

Defining $T^\pi(s)$ as the collective contribution of the rare states towards the value of state s [15], given π, we have

$$T^\pi(s) = \epsilon(s) \sum_{a \in A} \sum_{\tilde{s}' \in \tilde{S}} h(\tilde{s}'|s,a)\pi(a|s)[r(s,a,s') + \gamma V^\pi(s')]$$

Likewise, defining $U^\pi(s)$ as the collective contribution of the non-rare states (states in $S \backslash T$) towards the value of state s, given a fixed policy π, we have :

$$U^\pi(s) = (1 - \epsilon(s)) \times \sum_{a \in A} \sum_{\tilde{s}' \in \tilde{S}} \pi(a|s) h(\tilde{s}'|s,a)[r(s,a,s') + \gamma V^\pi(s')]. \quad (9)$$

We now give a formal definition of T, the set of rare states:

Definition 1. *A subset of states $T \subset S$ is called the Rare Event State Set if the following properties hold:*

1. There exists $s \in S, s' \in T$ for which $p(s'|s,a) > 0$.

2. Let $T^\pi(s)$ denote the contribution of the rare states towards the value of state s, according to Eq. (III). For a given policy π, there exists $s \in S$ for which $|T^\pi(s)| \gg 0$.

The first property means that transition to the rare event state set T is possible. The second property means that the rare event states in T collectively have a meaningful (sizeable) impact on the value function, and hence on system costs.

A. Importance Sampling

To solve our optimization problem, we will use importance sampling to sample the rare events (server failures) at a higher rate than they are expected to occur, to help the model learn an optimal policy in light of these server failures. We use importance sampling because if rare events such as server failures are sampled at their natural probabilities, the reinforcement learning algorithm is not able to converge to the optimal policy as it may decide that the storage cost isn't worth having backups, or may place the backups in suboptimal locations. This can be catastrophic in the event of failures. Specifically, we want to sample the server failures at a rate proportional to their contribution to system costs, i.e. proportional to the contribution they have to the value of state s. As mentioned in Section II, the rare event probability at state s is $\epsilon(s)$. We let $\hat{\epsilon}(s)$ denote the importance sampling probability at which we sample the rare events.

As we want to sample the server failures at a rate proportional to their system costs, a possibility for $\hat{\epsilon}(s)$ is $\hat{\epsilon}(s) = \frac{T^\pi(s)}{T^\pi(s) + U^\pi(s)}$. Nevertheless, we do not have a fixed policy π, as we are constantly updating the policy while learning the optimal policy π^*. Therefore, in our algorithm we set

$$\hat{\epsilon}(s) \leftarrow min(max(\delta, \frac{|T(s)|}{|T(s)| + |U(s)|}), 1 - \delta). \quad (10)$$

The bounds $(\delta, 1-\delta)$ help to ensure sufficient rare event sampling. $T(s)$ is updated via the following equation:

$$T(s^t) \leftarrow (1 - \alpha_T) T(s^t) + \alpha_T \epsilon(s^t)(r^{t+1} + \gamma V(s^{t+1})), \quad (11)$$

where α_T is the learning rate. $U(s)$ is updated via:

$$U(s^t) \leftarrow (1 - \alpha_U) U(s^t) + \alpha_U (1 - \epsilon(s^t))(r^{t+1} + \gamma V(s^{t+1})), \quad (12)$$

where α_U is the learning rate.

B. ImACRE: Importance Sampling based Actor Critic for Resilient Service Migration in Edge Computing

Our algorithm makes use of a simulator which trains an optimal service migration and backup placement policy in light of rare events. Such a simulator would likely be used in practice to train a migration algorithm, in order to avoid incurring the large failure costs experienced during server failures in reality. The transition probabilities $h(\tilde{s}|s,a)$, which is mostly dependent on the user mobility probabilities (Eq. (**??**)), can be obtained from historical data. The converged policy trained by the simulator is then applied to online scenarios, where rare events happen at their natural probabilities.

Algorithm description: The algorithm is presented in Algorithm 1 (**ImACRE**). As usual in actor-critic methods [20], we use neural networks to approximate the "actor" function that chooses the optimal policy according to a reward estimate given by the "critic" function. Firstly, we initialise the neural networks $\boldsymbol{\theta}$ and $\boldsymbol{c}$ randomly, $\hat{T}(s)$ and $\hat{U}(s)$ to be 0 (for all states), and $\hat{\epsilon}(s)$ to be $\frac{1}{2}$ for all states. We also initialise the learning rates $\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{c}}, \alpha_T, \alpha_U$. For every timeslot, the importance sampling rare event probability $\hat{\epsilon}(s)$ will determine whether or not a rare-event (failure) occurs. Thereafter, the rest of the state transition will occur according to the probability distribution $h(\tilde{s}|s,a)$, where $\tilde{s}(t) = (l_u(t), l_s(t), b_{ind}(t))$. This would result in a new state s^{t+1}, and a reward value $r^{t+1} = r(s^t, a^t, s^{t+1})$ (line 4). Based on the next state s^{t+1}, a new action a^{t+1} is selected according to the current policy network $\pi_{\boldsymbol{\theta}}$ (line 5).

Because the rare events are sampled at the probability $\hat{\epsilon}$ instead of their actual probability ϵ, we need a method of correction, in order to learn the optimal policy for the original system with transition probability $p(s'|s,a)$ (Eq. (6)). The importance sampling correction weight $w(s,a,s')$ will be defined by the actual probability divided by the importance sampling based probability (line 6):

$$w^t(s^t, a^t, s^{t+1}) \leftarrow \begin{cases} \epsilon(s^t)/\hat{\epsilon}(s^t), & \text{if } s^{t+1} \in T. \\ (1 - \epsilon(s^t))/(1 - \hat{\epsilon}(s^t)), & \text{if } s^{t+1} \notin T. \end{cases} \quad (13)$$

The importance sampling correction weight $w^t(s^t, a^t, s^{t+1})$ will be used when calculating the advantage function in (line 8). To obtain the advantage function, we first obtain the value of state s^t from the critic network $\boldsymbol{c}$ (line 7). The advantage function characterizes by how much better it is to take action a^t at state s^t, as compared to the average action at state s^t [20]. In our algorithm, we multiply $V(s^t)$, the value of state s^t, by the importance sampling correction weight $w^t(s^t, a^t, s^{t+1})$ (line 8). The advantage value $A(s^t, a^t)$ is then stored in the buffer (line 9). It will be used for optimizing the loss function at the end of every epoch.

We then update either $T(s^t)$ or $U(s^t)$, depending on whether the next state s^{t+1} is a rare event state or not (lines 10-11), according to Eqs. (11) and (12) respectively. Finally, based on $T(s^t)$ and $U(s^t)$, the importance sampling rare events probability $\hat{\epsilon}(s^t)$ is updated in line 12, according to $\frac{|T(s^t)|}{|T(s^t)|+|U(s^t)|}$, and bounded by δ and $1-\delta$. The process iterates for every timeslot.

At the end of every epoch, which consists of K timeslots, the actor critic network is optimized via gradient descent. The gradient for the actor network will be

$$\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{K} [\frac{1}{K} log\pi_{\boldsymbol{\theta}}(a^i|s^i) A(s^i, a^i)], \tag{14}$$

and the gradient for the critic network will be

$$\nabla_{\boldsymbol{c}} \sum_{i=1}^{K} \frac{1}{K} [A(s^i, a^i)]^2. \tag{15}$$

Algorithm 1 ImACRE: Importance Sampling based Advantage Actor Critic for Rare Events

```
1:  **Initialise:** parameters $\boldsymbol{\theta}, \boldsymbol{c}$ randomly, $\hat{T}(s), \hat{U}(s) \leftarrow 0$, $\hat{\epsilon}(s) \leftarrow \frac{1}{2}$, learning rates $\alpha_{\boldsymbol{\theta}}, \alpha_{\boldsymbol{c}}, \alpha_T, \alpha_U$.
2:  Select the initial state $s^0$ and action $a^0$.
3:  **for** all timeslots $t$ **do**
4:      $\hat{\epsilon}(s^t)$ determines if an anomaly happens. Thereafter, sample according to $h(\tilde{s}|s, a)$. The new state $s^{t+1}$ and a reward value $r^{t+1} = r(s^t, a^t, s^{t+1})$ is observed.
5:      Sample next action $a^{t+1} \sim \pi_{\boldsymbol{\theta}}(a|s^{t+1})$
6:      $w^t \leftarrow \begin{cases} \epsilon(s^t)/\hat{\epsilon}(s^t), & \text{if } s^{t+1} \in T. \\ (1-\epsilon(s^t))/(1-\hat{\epsilon}(s^t)), & \text{if } s^{t+1} \notin T. \end{cases}$
7:      $V(s^t) \leftarrow \boldsymbol{c}(s^t)$
8:      Advantage function $A(s^t, a^t) \leftarrow r^{t+1} + \gamma V(s^{t+1}) - w^t(s^t, a^t, s^{t+1}) V(s^t)$
9:      Store $A(s^t, a^t)$ in buffer.
10:     $T(s^t) \leftarrow (1-\alpha_T) T(s^t) + \alpha_T \epsilon(s^t)(r^{t+1} + \gamma V(s^{t+1}))$
11:     $U(s^t) \leftarrow (1-\alpha_U) U(s^t) + \alpha_U (1-\epsilon(s^t))(r^{t+1} + \gamma V(s^{t+1}))$
12:     $\hat{\epsilon}(s^t) \leftarrow min(max(\delta, \frac{|T(s^t)|}{|T(s^t)|+|U(s^t)|}), 1-\delta)$
13:     **for** every K steps **do**
14:         Update policy parameters using data in buffer: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{K} [\frac{1}{K} log\pi_{\boldsymbol{\theta}}(a^i|s^i) A(s^i, a^i)]$
15:         Update critic $\boldsymbol{c} \leftarrow \boldsymbol{c} - \alpha_{\boldsymbol{c}} \nabla_{\boldsymbol{c}} \sum_{i=1}^{K} \frac{1}{K} [A(s^i, a^i)]^2$
16:         Empty buffer.
17:     **end for**
18: **end for**
```

IV. Simulations

In this section, we provide numerical simulations to show the performance of our algorithms ImACRE. We perform our experiments with the help of real world traces.

Parameter setting: In our experiments, we have 9 access points (base stations). We use the ns-3 network simulator [21], which simulates realistic network conditions involving dynamic channels, to obtain realistic latency (delay) costs across different user-server (access point) location pairs. The latency costs obtained are in the range (2,10), and are dependent on the user-server distance. As the migration cost is also a function of the distance, we set m_{ij} to be $d_{i,j}^{comm} + \epsilon$, where $\epsilon \in (-0.5, 0.5)$. The natural failure rate is set to be 0.00001 and user mobility is modeled with uniformly drawn transition probabilities between user locations.

Convergence of ImACRE: Figure 2a. illustrates that algorithm ImACRE converges to the optimal reward. We set the cost of server failures to be -500, representing the high cost the user experiences when it is no longer able to get its task computed.The learning rate used is 0.05 and both the actor and critic have a network structure of $[24, 48, 24]$. ImACRE is able to learn the optimal actions through importance sampling, enabling the user to avoid the large cost of server failures.

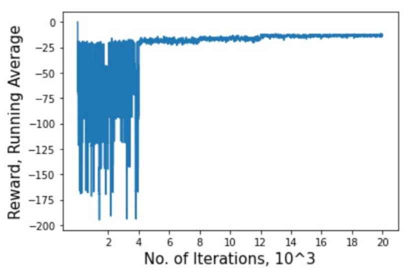
We compare our proposed algorithm ImACRE with the following **baselines**. Firstly, we train ImACRE and three Actor-Critic baselines on a simulator to avoid the large cost of failures. Next, we compare the cost incurred by the converged policy of each algorithm in an online scenario where the rare events occur at their natural rate $\epsilon(s)$.

AC with No Importance-Sampling (NIS): Rare events are simulated at their natural rate, and backups are permitted.

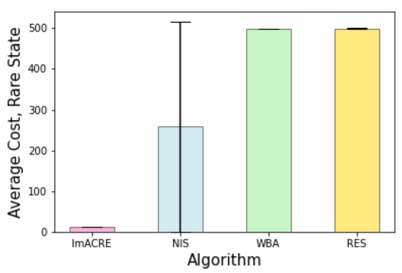
AC without Backups as an Action (WBA): Backups are not used. Rare events are still simulated, at their true rate.

AC without Rare Events Sampled (RES): Rare events are not considered (not modelled) at all during training. Likewise, backups are not considered.

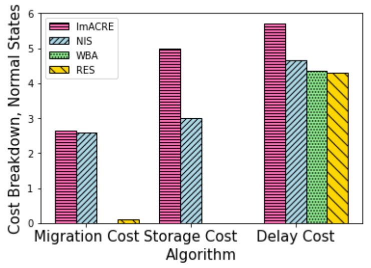
Cost Comparisons: Next, in Figs. 2b-c. we illustrate how our algorithm ImACRE performs in comparison to the baselines NIS, WBA and RES. In Fig. 2b, we plot the average cost experienced at rare event states, and in Fig. 2c, we plot the cost breakdown at ‘normal’ (non-rare event) states. We run all algorithms 10 times, and plot the average results, showing the standard deviation. It can be seen that our algorithm ImACRE gives a lower average cost at rare states in the online scenario, because our importance sampling algorithm sufficiently samples the rare events to learn an optimal policy. Our results show that it is optimal to place backups, and to place the main service at a different location from the backup. This allows users to offload their jobs at another location when the server at which their service was migrated to fails. The algorithm NIS is able to avoid the high rare event cost during some of the runs, nevertheless there is a higher variance in its performance, as can be seen in Fig. 2b. WBA and RES yield the highest costs during rare events, because backups are not used at all, and because rare events are not sampled at all during the simulator training (for RES). In Fig. 2c, we show how the average normal state cost breakdown differs across our algorithm and the baselines. It can be seen that our algorithm results in a higher cost at the non-rare event states, especially with respect to migration and storage, as a trade-



(a) Convergence of our Importance Sampling Algorithm ImACRE.



(b) Comparison of Costs at Rare Event States, averaged across 10 runs.



(c) Comparison of Costs at Non-Rare Event 'Normal' States, averaged across 10 runs.

Fig. 2. Numerical Simulations

off for being more risk-averse and preparing for rare events/ server failures. As an extension of this, we can propose an algorithm which takes the user risk level as input, which it uses to make a risk level-weighted decisions over our algorithm's policy (risk-averse) and the baseline's policy (risk-taking).

V. Conclusion

In edge computing, service profiles are migrated in light of user mobility to maintain quality of service. Concurrently, server failures may occur, due to system overload and overheating. Failures are rare but serious. They have an impact on the smooth and safe functioning of edge computing's latency sensitive applications. We propose the use of service profile backups. Nevertheless, as server failures occur at a low probability, it is challenging to jointly learn an optimal service migration and backup placement solution for both the usual and rare event scenarios. Therefore, we propose ImACRE, an importance sampling-based actor critic algorithm. It uses importance sampling in a simulator training setup to sample rare events at a rate proportional to their contribution to system costs, to learn an optimal service migration and backup placement policy. Finally, we use trace driven simulations to show that ImACRE converges, and is resilient towards server failures, subject to a higher preparation cost.

For future work, we will study the service migration multi-user scenario, where there is coupling across users in the computing delay. Our framework can be extended and applied to other resource allocation problems in communications and networking involving rare events with large consequences.

References

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[2] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Communications Surveys Tutorials*, vol. 19, no. 3, pp. 1628–1656, 2017.

[3] 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 Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.

[4] T. Ouyang, Z. Zhou, and X. Chen, "Follow me at the edge: Mobility-aware dynamic service placement for mobile edge computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 10, pp. 2333–2345, 2018.

[5] S. Wang, J. Xu, N. Zhang, and Y. Liu, "A survey on service migration in mobile edge computing," *IEEE Access*, vol. 6, pp. 23 511–23 528, 2018.

[6] S. Wang, Y. Guo, N. Zhang, P. Yang, A. Zhou, and X. Shen, "Delay-aware microservice coordination in mobile edge computing: A reinforcement learning approach," *IEEE Transactions on Mobile Computing*, vol. 20, no. 3, pp. 939–951, 2019.

[7] B. Gao, Z. Zhou, F. Liu, F. Xu, and B. Li, "An online framework for joint network selection and service placement in mobile edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

[8] M. Siew, K. Guo, D. Cai, L. Li, and T. Q. Quek, "Let's share vms: Optimal placement and pricing across base stations in mec systems," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[9] T. Kim, S. D. Sathyanarayana, S. Chen, Y. Im, X. Zhang, S. Ha, and C. Joe-Wong, "Modems: Optimizing edge computing migrations for user mobility," in *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*. IEEE, 2022, pp. 1159–1168.

[10] L. Ponemon, "Cost of data center outages," *Data Center Performance Benchmark Serie*, 2016.

[11] A. Aral and I. Brandić, "Learning spatiotemporal failure dependencies for resilient edge computing services," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 7, pp. 1578–1590, 2020.

[12] D. Zeng, L. Gu, S. Pan, J. Cai, and S. Guo, "Resource management at the network edge: A deep reinforcement learning approach," *IEEE Network*, vol. 33, no. 3, pp. 26–33, 2019.

[13] 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 Transactions on Networking*, vol. 27, no. 3, pp. 1272–1288, 2019.

[14] W. Mao, L. Wang, J. Zhao, and Y. Xu, "Online fault-tolerant vnf chain placement: A deep reinforcement learning approach," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 163–171.

[15] J. Frank, S. Mannor, and D. Precup, "Reinforcement learning in the presence of rare events," in *Proceedings of the 25th international conference on Machine learning*, 2008, pp. 336–343.

[16] D. Precup, "Eligibility traces for off-policy policy evaluation," *Computer Science Department Faculty Publication Series*, p. 80, 2000.

[17] L. Szlak and O. Shamir, "Convergence results for q-learning with experience replay," *arXiv preprint arXiv:2112.04213*, 2021.

[18] M. Siew, S. Sharma, K. Guo, C. Xu, T. Q. Quek, and C. Joe-Wong, "Fire: A failure-adaptive reinforcement learning framework for edge computing migrations," *arXiv preprint arXiv:2209.14399*, 2022.

[19] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[20] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[21] "ns3 network simulator." [Online]. Available: https://www.nsnam.org/