Using Deep Reinforcement Learning to Improve Sensor Selection in the Internet of Things

HOOTAN RASHTIAN\textsuperscript{1} AND SATHISH GOPALAKRISHNAN\textsuperscript{1,2}

\textsuperscript{1}Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada
\textsuperscript{2}Peter Wall Institute for Advanced Studies, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada

Corresponding author: Hootan Rashtian (rhootan@ece.ubc.ca)

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant, and in part by the Peter Wall Institute for Advanced Studies through a research grant.

ABSTRACT We study the problem of handling timeliness and criticality trade-off when gathering data from multiple resources in complex environments. In IoT environments, where several sensors transmitting data packets - with various criticality and timeliness, the rate of data collection could be limited due to associated costs (e.g., bandwidth limitations and energy considerations). Besides, environment complexity regarding data generation could impose additional challenges to balance criticality and timeliness when gathering data. For instance, when data packets (either regarding criticality or timeliness) of two or more sensors are correlated, or there exists temporal dependency among sensors, incorporating such patterns can expose challenges to trivial policies for data gathering. Motivated by the success of the Asynchronous Advantage Actor-Critic (A3C) approach, we first mapped vanilla A3C into our problem to compare its performance in terms of criticality-weighted deadline miss ratio to the considered baselines in multiple scenarios. We observed degradation of the A3C performance in complex scenarios. Therefore, we modified the A3C network by embedding long short term memory (LSTM) to improve performance in cases that vanilla A3C could not capture repeating patterns in data streams. Simulation results show that the modified A3C reduces the criticality-weighted deadline miss ratio from 0.3 to 0.19.

INDEX TERMS Internet of Things, asynchronous advantage actor-critic networks, soft scheduling, deep reinforcement learning, soft real-time systems.

I. INTRODUCTION

Connected devices are part of IoT environments in which every device talks to other related devices - by gathering and transmitting data - to timely communicate important/critical sensor data to interested parties for further usage. With the enormous amounts of data generation, selecting only important/critical/relevant data to be timely used for various usages is still an issue. Besides, environment complexity regarding data generation could impose additional challenges to balance criticality and timeliness when gathering data. For instance, when data or arrival of data for two or more sensors correlate, or there exists temporal dependency among sensors, incorporating such patterns can expose challenges to trivial policies for data gathering. Examples of environments with such attributes could be a network of sensors for capturing marine temperature, where there is temporal dependence among data [10] or intelligent buildings where deployed sensors generate data that is temporally and spatially dependent [3].

Capturing patterns such as temporal dependency or correlation on the data arrival of sensors could help to collect data more efficiently concerning the desired goal. For example, consider a network of four sensors ($S_A$, $S_B$, $S_C$, and $S_D$) where at each time step, we can collect data from only one sensor. The goal is to collect data packets in such a way that we have the highest accumulated criticality values over a time horizon, where each data packet is characterized by a pair of criticality and deadline value (e.g., data of sensor $S_i$ is referred as $(Cr_i, d_i)$). As an example (shown in Table 1 and Table 2), imagine that there is a temporal dependency between sensors $S_A$ and $S_B$ due to their close location. In case of not capturing the dependency mentioned above (detailed in Table 1 at the time step $t$, we have data from sensors $S_A$ and $S_C$, where data at $S_C$ has a higher criticality value and deemed as the better choice for transmission despite having larger deadline
value compared to \( S_A \). However, other data packets will be expired by the time that we want to collect data again at step \( t + 1 \), and the total criticality value collected at the end of \( t = 4 \) would be 8 (equivalent to criticality-weighted deadline miss ratio of 8/13 = 0.62). Alternatively, as shown in Table 2, by knowing and capturing the temporal dependency between sensors \( S_A \) and \( S_B \), one could transmit data at \( S_A \) since a data packet will arrive for \( S_B \) soon after \( S_A \). So, the total amount of criticality for the transmitted data packets from \( S_A \) and \( S_B \) can be as large as the criticality of data at \( S_C \) (if it was alternatively selected), and we may be able to collect data of \( S_C \) still if it is not expired yet. In this case, the total criticality value collected at the end of \( t = 4 \) would be 11 (equivalent to criticality-weighted deadline miss ratio of 11/13 = 0.85).

We discuss how we can prioritize and gather data from multiple devices when the data may have different timelines and criticality requirements - especially in complex environments. We use the term “complex” to refer to scenarios where the sensors may be correlated in time or space. Such correlations make sensor polling decisions harder. Given the challenges in capturing dependencies among sensors, and motivated by the success of Deep Reinforcement Learning (DRL) in policy estimation, we explore leveraging DRL techniques to tackle our problem. DRL techniques have shown to outperform alternative methods (e.g., traditional Q-learning) in terms of handling large state spaces, which is a requirement for deploying large IoT networks. Also, DRL techniques can handle both continuous and discrete state spaces. We propose an approach based on the Asynchronous Advantage Actor-Critic (A3C) [13] by improving the network structure such that it captures the likely case of having temporal dependency and correlation within data streams. We achieved this improvement by adding an LSTM layer to consider some previous states (rather than only one state) to learn recurring patterns within data. Access to previous states implies the notion of embedding memory into the model. Besides the addition of memory to the model, the A3C itself offers benefits. It outperforms some other RL alternatives (e.g., Q-learning methods based on Q-table and DQN) concerning resource requirement and time performance [13].

We provide the formal presentation of the problem as well as the simulation-based evaluation of scheduling policies for this problem.

### A. CONTEXT

The Internet of Things (IoT) refers to the vast number of things (i.e., electronics-infused devices) connected to the internet, which acts as sensors in their hosting environment, generating massive volumes of data. In such settings, everything from data acquisition to processing and analysis can leverage Machine Learning (ML) techniques to preserve efficiency and performance. As all the steps (i.e., data acquisition, processing, and analysis) mostly involve decision making of some sort, ML techniques would help in making informed decisions by capturing patterns in data and sensors behaviour. In a sense, the integration of ML into the IoT world would transform simple sensor-actuator devices into ambient intelligent devices.

In different application settings such as factory automation, hospital, and more environments [15], the notions of message criticality and timeliness co-exist. Imagine the case of factory automation, where sensory messages that require immediate actions compete with less urgent messages (i.e., no need for instant actions) though being critical. In such cases, there needs to be a decision-maker as to act upon sufficiently reasonable concerning the desired objective function.

While the ultimate goal is to deploy applications that can identify changes such as the one described above in the environment and appropriately adapt to them, sometimes such changes could occur in a more challenging way:

- There could be cases that correlation exists between the arrival of events for two or more sensors. For instance, in a factory, temperature and pressure sensors may generate data mostly at about the same time, different from optical sensors.

### TABLE 1. Example of sensor selection in a 4-step time window, when dependency cannot be captured. In this case, there is a temporal dependency for the arrival of data packets. The available data packets at sensor \( S_i \) is shown as \( S_i = (cr, d) \). As the table shows, the total accumulated criticality at the end of the time step \( t = 4 \) is 8.

| Time step | Current available data | Selected (sensor) data | Total accumulated criticality |
|-----------|------------------------|------------------------|-----------------------------|
| \( t=1 \) | \( S_A: (3,1) \), \( S_B: (1) \), \( S_C: (5,5) \), \( S_D: (1) \) | \( S_A: (3,1) \) | 3 |
| \( t=2 \) | \( S_A: (1) \), \( S_B: (3,2) \), \( S_C: (5,4) \), \( S_D: (2,1) \) | \( S_B: (3,2) \) | 3+3 = 6 |
| \( t=3 \) | No available data (all expired and no new arrivals) | \( S_C: (5,3) \) | 3+3+5 = 11 |
| \( t=4 \) | No available data (all expired and no new arrivals) | - | 11 |

### TABLE 2. Example of sensor selection in a 4-step time window, when dependency can be captured. In this case, there is a temporal dependency for the arrival of data packets. The available data packets at sensor \( S_i \) is shown as \( S_i = (cr, d) \). As the table shows, the total accumulated criticality at the end of the time step \( t = 4 \) is 11.

| Time step | Current available data | Selected (sensor) data | Total accumulated criticality |
|-----------|------------------------|------------------------|-----------------------------|
| \( t=1 \) | \( S_A: (3,1) \), \( S_B: (1) \), \( S_C: (5,5) \), \( S_D: (1) \) | \( S_A: (3,1) \) | 3 |
| \( t=2 \) | \( S_A: (1) \), \( S_B: (3,2) \), \( S_C: (5,4) \), \( S_D: (2,1) \) | \( S_B: (3,2) \) | 3+3 = 6 |
| \( t=3 \) | No available data (all expired and no new arrivals) | \( S_C: (5,3) \) | 3+3+5 = 11 |
| \( t=4 \) | No available data (all expired and no new arrivals) | - | 11 |
Furthermore, there could be cases where temporal dependence exists among sensors due to the physical arrangement of devices. For instance, imagine that two optical sensors (sensor A and sensor B) installed in separate locations in a factory. Sensors A and B may observe the same event, but at different points in time.

- It is more challenging to capture the correlation among sensor data. In the context of factory automation, there could be cases where the criticality/timeliness of messages for some sensors correlate with some other sensors. This situation may happen because of reasons such as related functionalities of the operating devices there. For example, two temperature sensors that are installed in the same factory warehouse sense correlated temperature values, whereas their sensed data may not correlate with temperature data of sensors in other warehouses.

From a system architecture perspective, we consider a network of devices (i.e., sensors) that sense events in the environment. There is a central unit managing the transmission of data from sensors by selecting a sensor at a time. Sensors communicate their sensing data to the central unit upon selection. The system that we study deals with the challenge of possessing minimal resources (e.g., in terms of energy and memory) as a likely setting for many networks ([4], [5], [23]), which emphasizes the essence of having an effective decision-making policy. For efficiency of energy and memory consumption, sensors would hold a limited number of messages. Such architecture makes it possible to be extended and creates clustered networks with multiple hierarchy levels. For instance, in a more extensive network, each central unit can operate as a relay device, such as the model described by Rashtian and Gopalakrishnan [15]. In the scope of our work here, we consider a central unit responsible for deciding during each decision-making epoch.

B. CONTRIBUTIONS

We propose and evaluate a scheduling mechanism for the described architecture where the environment adds complexity by causing correlations among messages and arrival dependency messages that need to be transmitted. We start by exploring the applicability of the A3C method [13] as a successful Deep Reinforcement Learning method and proposing our approach by improving upon that. The contributions of the work have four folds:

- Mapping the A3C approach to our problem and establishing a baseline method that showed consistent performance in less complex scenarios.
- Showing that the A3C performs no better than the greedy baselines in complex scenarios.
- Embedding memory to the network in the vanilla A3C to improve the performance in complex scenarios with high dependencies.
- Showing that based on the simulations, the modification to A3C did not negatively affect the performance in other scenarios where the vanilla A3C had already performed well. Such observation confirms the advantage of the proposed solution over the vanilla A3C in all studied scenarios.

II. SYSTEM MODEL

We consider a system with a centralized architecture, as shown in Figure 1. The IoT devices (i.e., sensors) are communicating to a central unit. Sensors sense the environment events and capture them as messages. The central unit transmits one message at a time from one of the sensors.

We consider messages of equal-sized lengths arriving at sensors. For energy consumption and memory costs, we assume each sensor would hold one data message. Sensors have enough resources to maintain only one message at the time. We use the following notation to describe a message and its characteristics: Message $M_i$ has an applicable deadline $d_i$ and criticality $\kappa_i$. If a message arrives at time $t$, then the absolute deadline for delivery of this message is $t + d_i$. Until the message reaches its expiration time, it will provide its highest value if selected. If the message is selected after its expiry, it will provide a discounted value (as will be discussed in Section 14). If a new message arrived at the sensor, and the sensor already has a message, the older message would be replaced by the new message.

At each scheduling epoch, the central unit selects one sensor to use the available bandwidth to transmit its associated message. Also, no assumptions are being made about the event arrival rate.

A. PERFORMANCE METRIC

In the model that we have described, the system goal is to minimize the criticality-weighted deadline miss ratio that is defined as follows: Let $N$ be the total number of messages generated during the time interval of interest that also have deadlines within that time interval. Let $x_i$ be an indicator variable for whether message $M_i$ missed its deadline or not. The criticality-weighted deadline miss ratio is

$$\rho := \frac{\sum_{i=1}^{N} x_i \kappa_i}{\sum_{j=1}^{N} \kappa_j}$$

(1)
With such performance, we can define the problem that we want to tackle.

**B. PROBLEM STATEMENT**

The problem that we want to tackle is to determine an efficient bandwidth allocation policy at the central unit to minimize the miss criticality ratio over a finite time interval as the optimal policies may be intractable:

$$\min \{ \rho := \frac{\sum_{i=1}^{N} x_i k_i}{\sum_{j=1}^{N} k_j} \}$$  \hspace{1cm} (2)

We do not know what messages will arrive until they arrive, and decisions regarding the selection of messages need to be made at each scheduling epoch.

**III. BACKGROUND**

We have chosen to take advantage of Reinforcement Learning (RL) to tackle our problem. We provide a brief background on RL techniques in general, A3C, and later discuss our proposed method.

**A. OVERVIEW OF DEEP REINFORCEMENT LEARNING**

In this section and before discussing the A3C network, we first introduce the basic concepts of RL and Deep Learning (DL), based on which Deep Reinforcement Learning (DRL) is defined.

1) **REINFORCEMENT LEARNING (RL)**

RL is a class of algorithms in machine learning that can achieve optimal control of a Markov Decision Process (MDP) [9], [19], [24]. There are generally two entities in RL - an agent and an environment. The environment evolves in a stochastic manner within a state-space at any time. The agent operates as the action executor and interacts with the environment. When it acts within a particular state, the environment will generate a reward/penalty to indicate how good was the action taken by the agent. The agent learns from this generated response from the environment over time. The policy determines the strategy for an agent to take action when being in a state. The agent task is to learn from its already taken actions such that in the future, it will take actions that are optimal concerning the value function $V_\pi(s_0)$.

We define the value function as the expected reward from actions taken by a policy ($\pi$) over a finite time horizon:

$$V_\pi (s_0) = E_{\tau_{s_0} \sim \pi} [R_{\text{total}}(\tau_{s_0})],$$  \hspace{1cm} (3)

where $\tau_{s_0}$ denotes a chain of states selected by adopting $\pi$ policy starting from $s_0$. $R_{\text{total}}$ is the reward accumulated from traversing such sequence of states.

Apart from value function, another important function is $Q$ function $Q^\pi(s_0, a_0)$, which is the expected reward for taking action $a_0$ in state $s_0$ and thereafter following a policy $\pi$. When policy $\pi$ is the optimal policy $\pi^*$, value function and $Q$ function are denoted by $V^* (s)$ and $Q^* (s, a)$, respectively. Note that $V^* (s) = \max_a Q^* (s, a)$. If the $Q$ functions $Q^* (s, a)$, $a \in A$ are given, the optimal policy can be easily found by $\pi^* = \arg \max_a Q^* (s, a)$. In order to learn the value functions or $Q$ functions, the Bellman optimality equations can usually help. Taking the discounted MDP with a discount factor of $\gamma$ for example, the Bellman optimality equations for the value function and $Q$ function is:

$$V^*(s_t) = \max_{a_t} \left[ r_t + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) V^*(s_{t+1}) \right],$$  \hspace{1cm} (4)

and

$$Q^*(s_t, a_t) = r_t + \gamma \sum_{s_{t+1}} P(s_{t+1}|s_t, a_t) \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}),$$  \hspace{1cm} (5)

respectively.

Bellman equations represent the relation between the value/$Q$ functions of the current state and the next state.

Usually, a large amount of memory is required to store the value functions and $Q$ functions. In some cases, when only small, finite state sets are involved, it is possible to store these in the form of tables or arrays. This method is called the tabular method. However, in most of the real-world problems, the state sets are large, sometimes infinite, which makes it impossible to store the value functions or $Q$ functions in the form of tables. Therefore, the trial-and-error interaction with the environment is hard for learning the environment dynamics due to formidably computation complexity and storage capacity requirement. Even if we can learn the dynamics, it imposes massive consumption of computing resources. In this case, one can approximate some functions of RL such as $Q$ functions or policy functions with a smaller set of parameters by the application of DL. The combination of RL and DL results in the more powerful DRL.

2) **DEEP LEARNING (DL)**

DL refers to a family of machine learning algorithms that leverage artificial neural networks (ANN) to learn patterns from a large amount of data. It can perform well in tasks like regression and classification. The weights and bias of every node in a neural network (NN) are the parameters of the NN. Usually, a neural net with two or more hidden layers is called a Deep Neural Network (DNN). A loss function $L(\theta) = g(Y(\theta), Y)$ is used in deep learning, which is a function of the output $Y(\theta)$ from the network and the desired output $(Y)$. The loss function evaluates the performance of the desired NN (in terms of learning the corresponding parameters till this point) in terms of modelling the given data (i.e., $Y = f(X)$). Depending on the task type, various loss functions can contribute. For instance, the standard regression loss functions include Mean Square Error (MSE), Mean Absolute Error (MAE), Mean Bias Error (MBE). Concerning the classification task, loss functions such as Cross-Entropy loss and Support Vector Machine (SVM) loss perform well. After that, gradient descent methods are used to update $\theta$ parameters in NNs and consequently minimize the loss function. Given a
loss function \( L(\theta) \), the parameters get updated by a gradient method such as the simple gradient:

\[
\nabla_\theta L(\theta) = \frac{\partial L(\theta)}{\partial \theta} \tag{6}
\]

Such gradient descent methods start from an initial point of \( \theta_0 \). As the input data is fed to NN, the average loss function over all input data is calculated and used to minimize \( L(\theta) \) by taking a step along the descent direction, i.e.,

\[
\theta \leftarrow \theta - \alpha \nabla_\theta L(\theta), \tag{7}
\]

where \( \alpha \) is a hyper-parameter named step size and indicates how fast the parameter values move towards the optimal direction. The above process is repeated iteratively by inputting more data to NN until convergence.

3) DEEP REINFORCEMENT LEARNING (DRL)

As discussed earlier, DRL refers to a family of methods that combine RL and DL to approximate either Value or Q functions (or even both) via a deep NN. In general, the DRL approaches can be categorized into two main groups: Value-based and Policy Gradient.

In Value-based methods for DRL the states \( s_t \in S \) or state-action pairs \((s_t, a_t)\in S \times A\) are inputs of NNs, while Q functions \( Q^\pi(s_t, a_t) \) or value functions \( V^\pi(s_t) \) are approximated by parameters \( \theta \) of NNs. An NN returns the approximated Q functions or value functions for the input states or state-action pairs. There can be a single output neuron or multiple output neurons. For the former case, the output can be either \( V^\pi(s_t) \) or \( Q^\pi(s_t, a_t) \) corresponding to the input \( s_t \) or \((s_t, a_t)\). For the latter case, the outputs are the \( Q \) functions for state \( s_t \) combined with every action, i.e., \( Q^\pi(s_t, a_t^1), \ldots, Q^\pi(s_t, a_{|A|}) \).

In Policy Gradient methods, NNs can directly approximate a policy as a function of the state, i.e., \( \pi^\theta(s) \). The states are used as inputs to the NNs, while policy \( \pi \) is approximated by parameters \( \theta \) of NNs as \( \pi^\theta \). In contrast to value-based DRL methods, the policy gradient methods for DRL is a direct mapping from state to action, which leads to better convergence properties and higher efficiency in high-dimensional or continuous action spaces [6]. Therefore, we chose to take leverage of policy gradient methods for our research problem. Specifically, we focused on A3C as the basis of our solution and improved upon it concerning our problem.

B. A3C NETWORK

A3c networks consist of multiple independent agents (i.e., neural networks) with their weights, which interact with a different copy of the environment in parallel. Therefore, they can explore a more significant part of the state-action space in much less time. The agents (or workers) are trained in parallel and periodically update a globally shared neural network, which holds shared parameters. The updates are not happening simultaneously, and that is where the asynchronous notions come from. After each update, the agents reset their parameters to those of the global network and continue their independent exploration and training until they update themselves again.

We shall now briefly elaborate on Asynchronous Advantage Actor-Critic (A3C) as the underlying structure in our approach.

We define a value function \( V(s) \) of a stochastic policy \( \pi(s) \) (that returns distribution of probabilities over actions) as an expected discounted reward:

\[
V(s) = E_{\pi(s)}[r + \gamma V(s')] \tag{8}
\]

where \( V(s) \) is the weight-average of \( r + \gamma V(s') \) for every action that can be potentially taken in state \( s \in S \).

We also define the action-value function \( Q(s, a) \) as:

\[
Q(s, a) = r + \gamma V(s') \tag{9}
\]

where we emphasize that the action is given and there is only one following \( s' \).

We define the advantage function as:

\[
A(s, a) = Q(s, a) - V(s) \tag{10}
\]

\( A(s, a) \) is the advantage function as it informs how good it is to take action \( a \) in a state \( s \) compared to the average performance. In case the action \( a \) is better than the average, the advantage function has a positive value. It gets a negative value when the action is worse than average.

Furthermore, we define \( \rho \) as the distribution of states, which indicates the probability of being in states. \( \rho_0^\pi \) and \( \rho^\pi \) denotes the distribution of beginning states in the environment and states under policy \( \pi \), respectively.

Since policy \( \pi \) is only a function of state \( s \), we can approximate it directly. In this case, a neural network (with \( \theta \) as the weights) would take a state \( s \) and output an action probability distribution \( \pi_\theta \). We shall use \( \pi \) and \( \pi_\theta \) interchangeably as the policy parametrized by the network weights \( \theta \).

On the other hand, we want to optimize the policy. We define a metric function \( J(\pi) \) as an averaged discounted reward that a policy \( \pi \) can accumulate over possible beginning states \( s_0 \):

\[
J(\pi) = E_{\rho_0^\pi}[V(s_0)] \tag{11}
\]

Now, we use the gradient of \( J(\pi) \) to improve it. The gradient of \( J(\pi) \) is derived in the Policy Gradient Theorem ([20], [22]) and has the following form:

\[
\nabla_\theta J(\pi) = E_{s \sim \rho^\pi, a \sim \pi(\cdot)}[A(s, a) \nabla_\theta \log \pi(a|s)] \tag{12}
\]

where the first part (i.e., \( A(s, a) \)) informs the advantage of taking action \( a \) in state \( s \). The second part of it (i.e., \( \nabla_\theta \log \pi(a|s) \)) informs a direction in which logged probability of taking action \( a \) in state \( s \) rises. Since both terms are together, the equation 12 increases that the likelihood of actions that are better than average performance while decreases the likelihood of actions worse than average performance. Since it is not feasible to compute the gradient over every state and every action, we use sampling for this computation (as the mean of samples lays near the expected value).

H. Rashtian, S. Gopalakrishnan: Using DRL to Improve Sensor Selection in the IoT
H. Rashtian, S. Gopalakrishnan: Using DRL to Improve Sensor Selection in the IoT

VOLUME 8, 2020
95213

experience replay
a method called
ples with high correlation. To avoid such an issue in DQN,
that the running of an agent would result in gathering sam-
the 
criticality-weighted deadline miss ratio
vanilla A3C, we examined its performance (quantified using
replay method. 
experience (i.e., various states and transitions). Second, this
it avoids the correlation as agents would have their unique
they arrive. The advantage of this approach is two folds: First,
has its copy of the environment and would use its samples as
random [1]. However, the way A3C handles this issue is to
run multiple agents simultaneously. In this case, each agent
consider studying the following cases, where:

• There exists a correlation among the arrival of data for
two or more sensors in the system.
• There is temporal dependence for the arrival of data for
two or more sensors in the system.
• There is temporal dependence concerning the arrival of
data as well as the correlation among data values for two
or more sensors.

We chose the above cases as a representative set of sce-
narios for a complex environment inspired by real-world
examples (e.g., in [10], [11]). We acknowledge that such a
set of cases can get extended to represent more complex
environments. We provide more details about each case when
we discuss the evaluation more completely (Section IV).

FIGURE 2. An example of A3C’s limitations, where its performance degrades in complex scenarios with 8 sensors. The Y-axis is the
Criticality-weighted Deadline Miss Ratio and the X-axis represents the Workload Intensity that we define as: \( \sum_{i \in M} \frac{C_r i \cdot \Delta \text{arrival}}{d_i} \). According to the
definition of the workload intensity, it increases with an increase in criticality \((C_r)\), an increase in data arrival \((\Delta \text{arrival})\) or a decrease in deadline \((d_i)\)
of messages.

The advantage function also needs to be computed. Let us expand the definition as:

\[
A(s, a) = Q(s, a) - V(s) = r + \gamma V(s') - V(s)
\]

where we can see that running an episode with a policy \( \pi \) would provide us with an unbiased estimate of the \( Q(s, a) \).
In other words, it is sufficient to know the value of \( V(s) \) to compute \( A(s, a) \). Therefore, we can also approximate \( V(s) \) by a neural network (similar to approximating action-value in DQN [7]).

Moreover, we can combine the two neural networks for estimating \( V(s) \) and \( \pi(s) \) to learn faster and more effectively. Also, on the negative side, separate networks are likely to
learn very similar low-level features. Besides, combining the networks would act as a regularizing element that results in better stability. In the case of the two networks, our neural network share all hidden layers and outputs two set of results - \( \pi(s) \) and \( V(s) \). The part that optimizes the policy is the **actor** and the one that estimates the value function is the **critic**. In fact, the **critic** provides **actor** with insights about its action.

The asynchronous notion in A3C originates from the fact that the running of an agent would result in gathering samples with high correlation. To avoid such an issue in DQN, a method called **experience replay** is used by storing the samples in memory and form a batch by retrieving them in random [1]. However, the way A3C handles this issue is to run multiple agents simultaneously. In this case, each agent has its copy of the environment and would use its samples as they arrive. The advantage of this approach is two folds: First, it avoids the correlation as agents would have their unique experience (i.e., various states and transitions). Second, this method requires less memory compared to the experience replay method.

**Limitations of A3C:** Given all the properties of the vanilla A3C, we examined its performance (quantified using the criticality-weighted deadline miss ratio) in the presence of temporal dependencies and correlation of sensor data values as an example of complex environments. Specifically, we experimented with 8 sensors in the setup explained in detail later (Section IV-B), where there is temporal dependence concerning the arrival of data as well as the correlation among data values for two or more sensors as described later in Scenario 4 at Section IV-A.

Figure 2 shows the criticality-weighted deadline miss ratio of policies concerning the workload intensity as defined further in equation 16. As indicated in Figure 2, the performance of A3C is comparable to the naïve greedy policies. With this observation, we hypothesized that such turbulence in the A3C performance relates to the lack of any memorization mechanism to capture recurrent patterns in data streams. Therefore, we decided to examine our hypothesis by proposing a modified version of A3C.

C. PROPOSED APPROACH

We propose an architecture for determining policy at the central hub for deciding on data streams in complex environments. Aligned with the examples provided in Section I-A, we consider studying the following cases, where:

- There exists a correlation among the arrival of data for two or more sensors in the system.
- There is temporal dependence for the arrival of data for two or more sensors in the system.
- There is temporal dependence concerning the arrival of data as well as the correlation among data values for two or more sensors.

We chose the above cases as a representative set of scenarios for a complex environment inspired by real-world examples (e.g., in [10], [11]). We acknowledge that such a set of cases can get extended to represent more complex environments. We provide more details about each case when we discuss the evaluation more completely (Section IV).
We make no assumptions concerning the similarity of information when there is any data arrival correlation or temporal dependency. It means that in such scenarios, if we collect data from two correlated/dependent sensors at two consecutive times (e.g., at \( t \) and \( t+1 \)), it does not mean that we have collected “redundant” data. We are solely focusing on collecting data packets in such a way that we have the highest accumulated criticality values over a time horizon. However, in some environment settings, it could be the case that such data arrival correlations/dependence is lead to redundant data. As mentioned above, the proposed model does make this assumption and consequently does not consider having redundant data/messages.\(^1\)

As shown in Figure 3, we can abstract the network in two parts: input and learning model.

![The proposed learning model based on A3C](image)

**FIGURE 3.** The proposed A3C-based network with embedded memory (i.e., LSTM layer).

At the input layer, the system state is an array of \( n = n(s) \) tuples \( (n(s) = |S|, S \text{ is the set of sensors in the system}) \), each tuple representing the available data-packet (criticality, deadline).

Concerning the learning model, we use the Asynchronous Advantage Actor-Critic (A3C) algorithm [13], which leverages a deep neural network to learn the policy and value functions while running parallel threads to update the parameters of the network. Regularization with policy entropy improves exploration by limiting the premature convergence to sub-optimal policy [13]. The core of our network contains an LSTM layer (output space = \( n(S) \)) followed by a fully connected network (output space = \( 2 \times n(S) \)) to perform the both required estimations. The LSTM layer is introduced to arm the agent such that it can have some memory of previous states. We propose such embedding of memory to the model as it is likely for an agent in complex environments to encounter recurring patterns in states. In these cases, the model is capable of making efficient decisions only if it can distinguish such patterns. Therefore, we chose to embed memory (i.e., by adding the LSTM layer) to enrich the model with this feature. Alternatively, one could argue about embedding memory to the sensors. However, we believe that such a decision would lead to a more fragile architecture and would negatively impact the scalability of IoT deployments.

Similar to any Reinforcement learning technique, we are concerned with how to take actions in the environment to maximize some notion of cumulative reward. It is essential to define a concrete and reasonable optimization goal, i.e., the reward. We define the reward in such a way that if a message is chosen after the expiry, a penalty is incurred:

\[
r(t) = \kappa^\alpha - I(\frac{t}{t - d})^\beta
\]

where \( \alpha \) and \( \beta \) are parameters that help a system architect achieve a balance between criticality and timeliness. These may be adapted on a per-sensor basis, but in this discussion, all local hubs use the same settings for these two parameters. \( t \) denotes the current time step, and \( d \) is the corresponding deadline of a message. \( \iota \) is the parameter that characterizes the penalty for a message, and it may be adapted per-message or per-class of messages. For simplicity of exposition, we use the same value of \( \iota \) for all messages. An example graph of the reward function for some choices of the parameters mentioned above is shown in Figure 4.

![An example graph of the reward function for some parameter choices](image)

**FIGURE 4.** An example graph of the reward function for some parameter choices \( (\iota = 1, \alpha = 2, \beta = 4, \text{ and } s \text{ and } d \text{ are randomly selected from interval of } (1, 5)). \) The peaks correspond to the cases where no penalty is incurred (i.e., \( f = 0 \)), whereas the troughs correspond to the cases with a penalty (i.e., \( f = 1 \)).

In terms of the loss function, we define it as follows:

\[
L = L_{\pi} + c_v L_v + c_{reg} L_{reg}
\]

where \( L_{\pi} \) is the loss of the policy, \( L_v \) is the value error and \( L_{reg} \) is a regularization term. The constants \( c_v \) and \( c_{reg} \) can show which part we want to emphasize on.

**D. ENVIRONMENT**

We created an environment for a complex IoT environment where data-packet streams arriving at sensors may have not only different distributions but also temporal dependency and correlation concerning their arrival times and data-packet values. Such properties are known in IoT environments such as intelligent buildings [3], marine environments [10], and multiple mobile sensing and computing applications [25]. In our environment, as mentioned in Section II, data packets arrive at sensors and the central unit chooses a sensor to fetch

\(^1\)As an extension to this work, we can model messages that have related information using a metric such as mutual information, and incorporating this metric into our objective will allow us to optimize the system behaviour suitably.
its data at each polling turn. Then, it verifies whether the deadline of the chosen sensor is still valid. If so, it calculates a reward without any extra penalty; otherwise, it still generates a reward (like a soft scheduler [15]) but subtracts a penalty to reflect the deadline expiry for the chosen data packet. This iteration continues at each time step until the episode finishes.

IV. QUANTITATIVE EVALUATION

Having described the proposed approach based on deep reinforcement learning, we discuss other heuristics that we can use to compare with our proposal. To recap the performance goal, we want to reduce the criticality-weighted deadline miss ratio.

We evaluate the policy from the proposed approach along with three other policies, namely: the vanilla A3C, critical greedy, and deadline greedy. Given the explanations for the A3C (Section III-B) and the proposed approach (Section III-C), we briefly elaborate on the greedy policies:

- Critical greedy: In this policy, the central unit selects a sensor using the highest criticality value among the messages at each sensor.
- Deadline greedy: In this policy, the central unit selects a sensor using the earliest deadline among the messages at each sensor.

A. EVALUATION SCENARIOS

As mentioned above, we have four different approaches (i.e., policies) to select a sensor each time. We compare these policies concerning their criticality-weighted deadline miss ratio. To understand the performance differences, we consider four different scenarios, each representing an environment concerning data arrival and values. We chose Scenario 1 to study our proposed solution in less complex environments (where there is no dependency among sensors). Inspired by the scenarios reported in the literature ([3], [10]), we considered Scenarios 2, 3, and 4 to explore the performance of our approach in complex environments to overcome the A3C limitation as discussed earlier and shown in Figure 2:

- Scenario 1: The arrival of data to sensors follows different distributions: a uniform distribution and multiple Poisson distributions - without dependencies.
- Scenario 2: The arrival of data to half of the sensors is dependent on each other in a pairwise manner. In other words, sensors in each pair are dependent on each other. In the case of having eight sensors in the system, two pairs (i.e., half of the sensors) correlated the arrival of data. For example, the arrivals of data for the $(s_1, s_2)$ pair of sensors were correlated. The same correlation existed between the arrival of data for the $(s_3, s_4)$ pair. We did not consider any such correlation for the rest of the sensors.
- Scenario 3: The arrival of data to half of the sensors is temporally dependent on each other in a pairwise manner. Similar to the details provided in Scenario 2, for half of the sensors, pairs of sensors had temporal dependence on each other. For example, when we had eight sensors, there was temporal dependence for the $(s_1, s_2)$ pair of sensors, and similarly for the $(s_3, s_4)$ pair while there was no dependence between other sensors.
- Scenario 4: The arrival of data to half of the sensors is temporally dependent on each other in a pairwise manner. Also, the state value (i.e., data value) of those sensors is dependent on each other. In the case that of eight sensors, for the pair of $(s_1, s_2)$, the data arrivals were temporally dependent, and data values were correlated. The same setting holds for another pair of sensors (e.g., $(s_3, s_4)$), and the rest of the sensors did not have a dependency of any kind.

B. EXPERIMENTAL SETUP

We perform our experiments in all of the above scenarios, where we tried multiple numbers of sensors ($n = 4, 8, 16$). We chose to report the results for $n = 8$ here due to the similarity of results and the results for the other two cases of 4 and 16 sensors are available in the Appendices (Section VI-A) and B (Section VI-B), respectively. To be consistent, we fixed the values of temporal dependencies and correlations (regarding arrival time or state values) wherever they existed across the scenarios. For temporal dependency, we used $t_w = 5$ to denote the temporal difference with correlation value of $corr = 0.9$ (i.e., to hold high correlation). This correlation value was also used for the cases that only correlation exists (no temporal dependency). As mentioned in Section IV-A, half of the sensors (i.e., equivalent to four sensors as $n = 8$) have dependencies in scenarios 2, 3, and 4, which means two pairs of sensors have either or both of temporal dependency or correlation. We used the arrival rate of $\lambda = 0.15$ for these scenarios. For the first scenario, we chose the arrival time for one sensor from a uniform distribution over the time horizon of the study, and the other sensors from multiple Poisson processes ($\lambda = [0.05, 0.15, 0.3, 0.35, 0.5, 0.65, 0.85]$). Also, for each message, the deadline and criticality values are chosen from a uniform distribution. Therefore, we chose deadlimes of messages from the uniform distribution $U_d(1, 10)$, and criticality values from the uniform distribution $U_c(1, 10)$. In the case of dependencies among data values (i.e., criticality and deadline values), the chosen values are still within the ranges of the above uniform distributions.

We ran the simulations with eight threads as it directly impacts the performance due to affecting the quality of gradient. We tweaked the hyperparameters and ended up having simulation runs through 250 episodes each 5000 steps, the minimum batch size of $n_{batch} = 32$, and the learning rate of $\delta = 0.005$. Also, we chose the values for the constants of the loss function as $c_v = 0.5$ and $c_{reg} = 0.01$.

C. RESULTS

We elaborate on our observations from the simulations. In each scenario, we considered the aggregate results of the 250 episodes for the evaluations.
Workload Intensity: We define a workload intensity metric as:

$$\sum_{i \in M} \frac{C_{r_i} \times \lambda_{\text{arrival}}}{d_i}$$  \hspace{1cm} (16)$$

where $M$ is the set of all messages arriving in a simulation episode, $\lambda_{\text{arrival}}$ is the arrival rate of events, and $C_{r_i}$ and $d_i$ are the criticality and deadline for each message, respectively. We could calculate this metric for each episode of simulations over 5000 steps. Such a metric essentially captures, to some extent, the workload intensity. Therefore, we report the performance of the policies concerning the “criticality-weighted deadline miss ratio” (on the Y-axis) along with the “workload intensity” (on the X-axis) when presenting the simulation results.

As depicted in Figures 5, 6, 7, and 8, each data point represents the overall “criticality-weighted deadline miss ratio” in terms of the workload-intensity metric (as defined earlier) over 5000 steps during an episode. Table 3 summarizes the results of experiments for the case of 8 sensors. Across all scenarios, we find the proposed approach performs well. In two of the scenarios (III and IV), it outperforms all others, which confirms the idea of adding memory to capture temporal relations. In scenario I, it still performs slightly better than the vanilla A3C and outperforms the other two policies. In scenario II, it performs competitively when compared with vanilla A3C while still outperforming the two greedy baselines. Table 4 (Appendix-C) provides a more comprehensive set of results by including the experiments for the cases of having 4 and 16 sensors.

Observing that the modification to the A3C did not negatively affect the performance in less complex scenarios suggests the advantage of the proposed solution over the vanilla A3C as a generalized solution.

In addition to generalizability, our proposed solution also offers scalability compared to the traditional RL approaches, such as the tabular Q-learning. As described earlier (Section III-A1), with the proposed DRL solution, we do not deal with updating the Q-table of states and actions. Instead, we have the states as the input of the NN and
approximate the value function and action values. Therefore, we alleviate the massive consumption of computing resources for updating a Q-table. In this way, it is easier to increase the number of sensors as it only changes the size of the input array in the NN, which requires considerably lower computing resources than Q-tables that grow exponentially. One possible bottleneck towards the large scale implementation of our approach would be the training time that could increase in environments with more sensors and complexity. This issue, however, could be addressed by leveraging GPUs for computational speedup, as shown to be effective in improving A3C performance by Babaeizadeh et al. [2].

V. RELATED WORK
Our research brings multiple areas such as DRL, performance analysis, and job scheduling together. Therefore, the related work section is mostly about how DRL has helped to solve problems of scheduling specifically about the performance of sophisticated sensors deployment that capturing data patterns is non-trivial [21], [26].
TABLE 4. The summary of all experiments concerning the criticality-weighted deadline miss ratio.

| Number of Sensors | Scenario 1 | Scenario 2 | Scenario 3 | Scenario 4 |
|-------------------|------------|------------|------------|------------|
| 4                 | m_A3C       | Vanilla A3C | Criticality Greedy | Deadline Greedy |
|                   | 0.2436  | 0.2493     | 0.2806     | 0.2744     |
|                   | 0.1643  | 0.1683     | 0.2616     | 0.2934     |
|                   | 0.1710  | 0.2586     | 0.2749     | 0.2674     |
|                   | 0.1585  | 0.2306     | 0.2636     | 0.2572     |
| 8                 | m_A3C       | Vanilla A3C | Criticality Greedy | Deadline Greedy |
|                   | 0.2883  | 0.2989     | 0.3277     | 0.3201     |
|                   | 0.2073  | 0.2063     | 0.3012     | 0.3398     |
|                   | 0.2077  | 0.3139     | 0.3145     | 0.3056     |
|                   | 0.1938  | 0.2779     | 0.3018     | 0.2942     |
| 16                | m_A3C       | Vanilla A3C | Criticality Greedy | Deadline Greedy |
|                   | 0.3234  | 0.3335     | 0.3550     | 0.3494     |
|                   | 0.2222  | 0.2249     | 0.3367     | 0.3803     |
|                   | 0.2254  | 0.3482     | 0.3542     | 0.3458     |
|                   | 0.2084  | 0.3076     | 0.3431     | 0.3350     |

FIGURE 9. Range of criticality-weighted deadline miss ratio ($\rho$) values concerning the workload intensity for scenario 1 with 4 sensors. The plot indicates the range of criticality-weighted deadline miss ratio values using the same data reported in Table 4. The proposed approach (m_A3C) outperforms the vanilla A3C.

FIGURE 10. Range of criticality-weighted deadline miss ratio ($\rho$) values concerning the workload intensity for scenario 2 with 4 sensors. The plot indicates the range of criticality-weighted deadline miss ratio values using the same data reported in Table 4. The proposed approach (m_A3C) outperforms the vanilla A3C.

In the scheduling of distributed tasks in grid and cloud, there are several efforts that optimize metrics such as lifespan and load balance [12], [18]. However, such works focus only on criticality/importance or timeliness. In our work, we study the trade-off between these two system properties.

In terms of performance analysis, our work shares similarities such as the notion of the metrics between what we study as the criticality-weighted deadline miss ratio (Section 1) and other metrics such as the one explained by Salah et al. [16]. Aside from the performance metrics, different works consider hierarchical models, such as the ones studied by Khazaei et al. [8] and Rashtian and Gopalakrishnan [15]. Our work, though, is focusing on the case of a centralized model as a standard model in infrastructure-as-a-service (IaaS) platforms.
FIGURE 11. Range of criticality-weighted deadline miss ratio ($\rho$) values concerning the workload intensity for scenario 3 with 4 sensors. The plot indicates the range of criticality-weighted deadline miss ratio values using the same data reported in Table 4. The proposed approach (m_A3C) outperforms the vanilla A3C.

FIGURE 12. Range of criticality-weighted deadline miss ratio ($\rho$) values concerning the workload intensity for scenario 4 with 4 sensors. The plot indicates the range of criticality-weighted deadline miss ratio values using the same data reported in Table 4. The proposed approach (m_A3C) outperforms the vanilla A3C.

FIGURE 13. Range of criticality-weighted deadline miss ratio ($\rho$) values concerning the workload intensity for scenario 1 with 16 sensors. The plot indicates the range of criticality-weighted deadline miss ratio values using the same data reported in Table 4. The proposed approach (m_A3C) performs competitively compared to the vanilla A3C.

On the RL side, our work compares to the body of work done via policy gradient methods. Policy gradient methods include a large family of reinforcement learning algorithms. They have a long history [24], but only recently were backed by neural networks and had success in high-dimensional cases. A3C algorithm was published in 2016 and can do better than DQN [14] with a fraction of time and resources [13]. Although there are several papers leveraging policy gradient methods, our work is distinguished from many of them by both the application and network structure.
VI. CONCLUSIONS AND FUTURE WORK

We have demonstrated that a deep reinforcement learning approach to balancing message criticality and timeliness in a complex IoT environment is effective. We proposed an approach by improving upon the A3C algorithm via embedding memory to the model such that it can capture recurring patterns of data in complex environments. Our solution outperforms the studied alternative heuristics that one could use in this context. We envision the applicability of such an idea in the context of IoT environments that host temporally dependent and correlated data arrival. We have tackled this problem in the specific settings where messages have two essential properties of criticality and timeliness.

We have shown that our solution is effective in four scenarios. The proposed approach outperforms the rest of the policies in complex scenarios, where we have data arrival correlations, data arrival temporal dependency, and data arrival temporal dependency plus correlation of data (concerning criticality and deadline values). Also, our solution remains effective in the more straightforward, where the arrival of data to sensors follows different distribution without dependencies. The observation of the results both in complex and non-complex scenarios suggest the generalizability of the proposed solution.

As future work, we envision multiple avenues to explore. One natural extension to this work is to explore the effectiveness of the approach with more scenarios. This path may lead to the modification of network architecture. Another possible plan could be to compare the proposed approach with solutions based on some other DRL algorithms, such as PPO [17]. Such explorations could shed light on the trade-off of convergence vs. stability across DRL-based approaches. Lastly, another direction for future work would be to answer the question of “How should we model information correlation to preventing redundant message collection?” In this case, we may be able to use mutual information as a metric to capture the value of different messages; we could ignore messages with high mutual information correlation in relation to messages that have been scheduled. As our proposed model here does not assume and address collecting redundant data,
it would be interesting to explore potential modification to the framework, to prevent collecting repetitive data while still capturing recurring patterns within data. We envision that addressing this problem in our proposed framework would require updates to the reward function and performance metric, but the overall solution structure is likely to remain the same. Finally, it would be interesting to solve a similar case when the assumption is to have two or more classes of tasks (e.g., critical and regular/normal tasks), and the goal is to prioritize critical tasks over other classes of tasks. Such a study would complement our work in this paper as the current problem settings do not assume such a clear distinction among tasks and solely focuses on minimizing the weighted-criticality deadline miss ratio metric over a finite horizon.


cite{references}

APPENDIX

Here we present the rest of the results from our experiments as mentioned in the paper.

A. EXPERIMENT RESULTS FOR 4 SENSORS

In this section, we present the experiment results for the case of having 4 sensors (see Figs. 9–12).

B. EXPERIMENT RESULTS FOR 16 SENSORS

In this section, we present the experiment results for the case of having 16 sensors (see Figs. 13–16).

C. TABLE OF RESULTS SUMMARY

Table 4 summarizes the results for the cases of 4, 8, and 16 sensors. It reports the average Criticality-weighted Deadline Miss Ratio for the policies in each of the scenarios. As shown in green-coloured cells, the proposed policy (m\_A3C) almost consistently performs as the best policy. Even when it is not the best, it still performs reasonably well with only one percent of difference (concerning $\rho$) compared to the best result. The red-coloured cells show the worst performance among the policies, which is dominantly shared between the two greedy policies.

REFERENCES

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba, “Hindsight experience replay,” in Proc. Adv. Neural Inf. Process. Syst., 2017, pp. 5048–5058.
[2] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, “GA3C: GPU-based A3C for deep reinforcement learning,” CoRR, vol. abs/1611.06256, Nov. 2016.
[3] M. Charfi, Y. Gripay, and J.-M. Petit, “Spatio-temporal functional dependencies for sensor data streams,” in Proc. Int. Symp. Spatial Temporal Databases. Arlington, VA, USA: Springer, 2017, pp. 182–199. [Online]. Available: http://conference.researchbib.com/view/event/65260
[4] H. Cho and J. Jeong, “Implementation and performance analysis of power and cost-reduced OPC UA gateway for industrial IoT platforms,” in Proc. 28th Int. Telecommun. Netw. Appl. Conf. (ITNAC), Nov. 2018, pp. 1–3.
[5] D. G. Costa, “Visual sensors hardware platforms: A review,” IEEE Sensors J., vol. 20, no. 8, pp. 4025–4033, Aug. 2020.
[6] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An introduction to deep reinforcement learning,” Found. Trends Mach. Learn., vol. 11, nos. 3–4, pp. 219–354, 2018.
[7] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, “Continuous deep q-learning with model-based acceleration,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 2829–2838.
[8] H. Khazaee, J. Misic, and V. B. Misic, “A fine-grained performance model of cloud computing centers,” IEEE Trans. Parallel Distrib. Syst., vol. 24, no. 11, pp. 2138–2147, Nov. 2013.
[9] V. R. Konda and J. N. Tsitsiklis, “Actor-critic algorithms,” in Proc. Adv. neural Inf. Process. Syst., 2000, pp. 1008–1014.
[10] J. Liu, T. Zhang, G. Han, and Y. Gou, “TD-LSTM: Temporal dependence-based LSTM networks for marine temperature prediction,” Sensors, vol. 18, no. 11, p. 3797, Nov. 2018.
[11] M. Á. López Medina, M. Espinilla, C. Paggetti, and J. M. Quero, “Activity recognition for IoT devices using fuzzy spatio-temporal features as environmental sensor fusion,” Sensors, vol. 19, no. 16, p. 3512, Aug. 2019.
[12] M. C. Lucas-Estan and J. Gozalvez, “Load balancing for reliable self-organizing industrial IoT networks,” IEEE Trans. Ind. Informat., vol. 15, no. 9, pp. 5052–5063, Sep. 2019.
[13] V. Minn, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, “Asynchronous methods for deep reinforcement learning,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 1928–1937.
[14] I. Osband, C. Blundell, A. Pritzel, and B. V. Roy, “Deep exploration via bootstrapped DQN,” in Proc. Adv. Neural Inf. Process. Syst., 2016, pp. 4026–4034.
[15] H. Rashtian and S. Gopalakrishnan, “Balancing message criticality and timeliness in IoT networks,” IEEE Access, vol. 7, pp. 145738–145745, 2019.
H. Rashtian, S. Gopalakrishnan: Using DRL to Improve Sensor Selection in the IoT

[16] K. Salah, K. Elbadawi, and R. Boutaba, “An analytical model for estimating cloud resources of elastic services,” J. Netw. Syst. Manage., vol. 24, no. 2, pp. 285–308, Apr. 2016.

[17] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” 2017, arXiv:1707.06347. [Online]. Available: http://arxiv.org/abs/1707.06347.

[18] J. Shi, J. Luo, F. Dong, and J. Zhang, “A budget and deadline aware scientific workflow resource provisioning and scheduling mechanism for cloud,” in Proc. IEEE 18th Int. Conf. Comput. Supported Cooperat. Work Design (CSCWD), May 2014, pp. 672–677.

[19] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, MA, USA: MIT Press, 2018.

[20] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, “Policy gradient methods for reinforcement learning with function approximation,” in Proc. Adv. Neural Inf. Process. Syst., 2000, pp. 1057–1063.

[21] A. A. Szpiro, P. D. Sampson, L. Sheppard, T. Lumley, S. D. Adar, and J. D. Kaufman, “Predicting intra-urban variation in air pollution concentrations with complex spatio-temporal dependencies,” Environmetrics, vol. 21, no. 6, pp. 606–631, 2010.

[22] P. S. Thomas and E. Brunskill, “Policy gradient methods for reinforcement learning with function approximation and action-dependent baselines,” 2017, arXiv:1706.06643. [Online]. Available: http://arxiv.org/abs/1706.06643.

[23] A. A. Visheratin, M. Melnik, and D. Nasonov, “Workflow scheduling algorithms for hard-deadline constrained cloud environments,” Procedia Comput. Sci., vol. 80, pp. 2098–2106, Jan. 2016.

[24] R. J. Williams, “Simple statistical gradient-following algorithms for connectionist reinforcement learning,” Mach. Learn., vol. 8, nos. 3–4, pp. 229–256, May 1992.

[25] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, “DeepSense: A unified deep learning framework for time-series mobile sensing data processing,” in Proc. 26th Int. Conf. World Wide Web, Apr. 2017, pp. 351–360.

[26] J. Zhang, Y. Zheng, and D. Qi, “Deep spatio-temporal residual networks for citywide crowd flows prediction,” in Proc. 31st AAAI Conf. Artif. Intell., 2017, pp. 1655–1661.

HOOTAN RASHTIAN received the M.Sc. degree from The University of British Columbia (UBC), in 2014, where he is currently pursuing the Ph.D. degree with the RADICAL Computing Systems Lab. He has received the NSERC Engage Grant and multiple Mitacs Accelerate grants for industry-related projects. He has collaborated with multiple Research and Development projects in the industry, such as Oracle Labs to apply machine learning techniques, including predictive modeling, recommendation systems, meta learning, and unsupervised learning on real-world problems. His research interests include developing techniques in statistical machine learning, reinforcement learning, and dynamic programming to improve data collection and prioritization in data-rich environments such as IoT settings.

SATHISH GOPALAKRISHNAN is currently an Associate Professor of electrical and computer engineering with The University of British Columbia (UBC) and also an Associate Head of undergraduate programs with his department. He works on research problems related to the safety of cyber-physical systems (computing systems embedded in and interacting with the physical world). He is interested in techniques and tools for resource allocation and the verification of correct system behavior in such embedded computing systems. The broader context for his work relates to how we think about and develop dependable software. He has been recognized for his research with Best Paper Awards from the IEEE Real-Time Systems Symposium and with the Peter Wall Scholarship from the UBC.

* * *