On Designing Smart Agents for Service Provisioning in Blockchain-Powered Systems

Mhaisen, Naram; Allahham, Mhd Saria; Mohamed, Amr; Erbad, Aiman; Guizani, Mohsen

DOI
10.1109/TNSE.2021.3118970

Publication date
2022

Document Version
Final published version

Published in
IEEE Transactions on Network Science and Engineering

Citation (APA)
Mhaisen, N., Allahham, M. S., Mohamed, A., Erbad, A., & Guizani, M. (2022). On Designing Smart Agents for Service Provisioning in Blockchain-Powered Systems. *IEEE Transactions on Network Science and Engineering*, *9*(2), 401-415. Article 9573346. https://doi.org/10.1109/TNSE.2021.3118970

On Designing Smart Agents for Service Provisioning in Blockchain-Powered Systems

Naram Mhaisen, Mhd Saria Allahham, Amr Mohamed, *Senior Member, IEEE*, Aiman Erbad, *Senior Member, IEEE*, and Mohsen Guizani, *Fellow, IEEE*

***Abstract*—Service provisioning systems assign users to service providers according to allocation criteria that strike an optimal trade-off between users' Quality of Experience (QoE) and the operation cost endured by providers. These systems have been leveraging Smart Contracts (SCs) to add trust and transparency to their criteria. However, deploying fixed allocation criteria in SCs does not necessarily lead to the best performance over time since the blockchain participants join and leave flexibly, and their load varies with time, making the original allocation sub-optimal. Furthermore, updating the criteria manually at every variation in the blockchain jeopardizes the autonomous and independent execution promised by SCs. Thus, we propose a set of light-weight agents for SCs that are capable of optimizing the performance. We also propose using online learning SCs, empowered by Deep Reinforcement Learning (DRL) agent, that leverage the chained data to continuously self-tune its allocation criteria. We show that the proposed learning-assisted method achieves superior performance on the combinatorial multi-stage allocation problem while still being executable in real-time. We also compare the proposed approach with standard heuristics as well as planning methods. Results show a significant performance advantage over heuristics and better adaptability to the dynamic nature of blockchain networks.**

***Index Terms*—IoT, blockchain, smart contracts, service provisioning, deep reinforcement learning, edge computing.**

I. Introduction

THE Internet of Things (IoT) is a technology paradigm envisioned as a global network of heterogeneous devices that can interact with each other and/or the environment around them. IoT devices are evolving from mere collectors of data to sophisticated task processing entities that are capable of processing and exchanging data using the edge computing paradigm [1]. These devices can range from small medical wearable devices to industrial equipment or autonomous electric vehicles (EV). One of IoT's main challenges is facilitating service management among them, especially for constrained devices. Most IoT devices have limitations on storing and processing the needed information to manage their resources and services. Thus, the necessity of having a secure, autonomous, and reliable service management framework arises. Moreover, the fact that IoT devices can form largely decentralized and dynamic networks necessitates the need for the decentralization and security of such service management systems. Hence, blockchain-based systems are foreseen to be one of the essential service management and data sharing systems between IoT devices [2].

Blockchain has recently emerged as one of the most secure distributed system architectures. It is based on P2P networking, where all participant nodes exchange transactions and reach a consensus on the general state of an asset. Each of the blockchain's nodes preserves an append-only, cryptographically-linked list of all events of interests and transactions that occurred in the network. This record is also referred to as the distributed ledger. The peer-enforced consensus rules and absence of a centralized third party make data manipulation extremely hard [3]–[5]. Blockchains enable any programmed logic to be deployed on the distributed network. These programs are called Smart Contracts (SCs) [6]. SCs are general-purpose software applications that are deployed on the blockchain distributed network. They represent real-world agreements digitally, which offers multiple appealing security guarantees [7]. Namely, being a form of distributed software, SCs have high availability in case of node failures across the network. Further, their code is immutable as it is stored on all nodes. Lastly, blockchain's data is edited only through cryptographic consensus, making the execution of an SC automatic, and the resulting output cryptographically verifiable by all participant nodes [8]. These security benefits make the software programs (SCs) transparent and self-enforceable. [9]–[11].

Recently, multiple domains started to leverage the autonomy, resilience, and transparency features of SCs in service provisioning and resource management for IoT devices. These include peer-to-peer P2P energy trading of household renewable energy, or EV [12], with some projects already deployed in industry [13], where SCs are used to enhance

Manuscript received January 5, 2021; revised August 10, 2021; accepted September 30, 2021. Date of publication October 14, 2021; date of current version March 23, 2022. This work was supported by NPRP under Grant #NPRP12S-0305-190231 from Qatar National Research Fund. Recommended for acceptance by Dr. Yan Zhang. *(Corresponding author: Amr Mohamed.)*

Naram Mhaisen is with the Faculty of Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology, 2600 AA Delft, The Netherlands, and also with Computer Science and Engineering, Qatar University, Doha, Qatar (e-mail: naram@qu.edu.qa).

Mhd Saria Allahham is with Computer Science and Engineering, Qatar University, Doha, Qatar, and also with the School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada (e-mail: ma1517219@qu.edu.qa).

Amr Mohamed and Mohsen Guizani are with the College of Engineering, Qatar University, Doha 2713, Qatar (e-mail: amrm@qu.edu.qa; mguizani@ieee.org).

Aiman Erbad is with the Division of Information and Computing Technology, College of Science and Engineering, Hamad Bin Khlifa University, Doha, Qatar (e-mail: aerbad@ieee.org).

Digital Object Identifier 10.1109/TNSE.2021.3118970

P2P energy trading. Cloud and edge resource allocation market also heavily utilizes SCs to facilitate the resource auction mechanism through recording requests and services for non-repudiation attack protection [14]. In the domain of IoT smart sensors, SCs are being utilized for the provenance of data sources, as well as machine-to-machine (M2M) payments to enable the direct transaction between sensing/actuating nodes without relying on a centralized server [15]. In general, SCs provide appealing features for service provisioning in any industry and have been shown to support large deployments of IoT devices for automated service provisioning [16].

A crucial consideration in service provisioning is the criteria of assigning users (or tasks) to service providers. Such allocation problems should consider multiple factors, including the *service cost* that users endure as well as the *operation cost* of service providers. In general, the service allocation should jointly optimize the service cost and operation cost. Obviously, service provisioning systems aim to assign users to the service provider with minimum fees to save service costs for the users. However, service providers have limited capacities, and overloading them might lead to high operation costs over time. For example, if the service provider is an IoT node delivering remote information as a service, then energy consumption is a significant concern. Similar arguments can be made if the service provider is a smart battery/EV trading energy since more trading operations can tear the battery. In general, service providers' operation cost, as deduced by their load, should also be considered by any service provisioning system. Hence, the joint optimization of service cost and operation cost is desirable.

The joint optimization in service provisioning is classically modeled as task allocation in a multi-agent setting [17]. However, this formulation is not portable to blockchain environments due to multiple factors. First, the task allocation problem requires a static set of users and service providers. However, blockchain is a dynamic system where service providers of different capacities join and leave flexibly. Second, as the blockchain is a temporal system, the task allocation committed at the current block will affect the decisions at future blocks as the providers' load will differ. Hence, even if the allocation is locally optimal, it might lead to a sub-optimal situation over time. Lastly, SCs, when deployed, are immutable and cannot be changed or require a complex update process [7]. In summary, the heterogeneity of participants and the temporal nature of blockchain systems impose a challenge to traditional optimization methods and calls for adaptive SCs.

Two major recent advances motivate our current work; First, the proposal of rational contracts, which can optimize their performance after they are deployed on the blockchain through learning from past chained data [18]. Second, the major shift of service provisioning systems from conventional centralized cloud-based platforms towards distributed-ledger-based platforms, providing transparency and security as a by-product from the latter. Thus, the current work, which proposes service provisioning through intelligent smart contracts, is a natural integration that results in not only secure and transparent SP systems but also adaptive ones that continuously tune their assignment criteria to optimize the users' QoE.

In this paper, we investigate the problem of service provisioning through IoT devices. We leverage smart contracts for secure and autonomous execution of the provisioning, modeling them as agents that perform the task allocation involved in the provisioning system. Several design approaches are investigated for these agents, including heuristic, planning, and learning-based (RL) agents. The contributions of this paper are summarized as follows:

- Formulating the service provisioning as a multi-objective Markov Decision Process (MDP) whose solution achieves the optimal trade-off between service providers' *operation cost* and users' *service cost* along with the QoE as deduced from an SC-based reputation system.
- Proposing a set of heuristic-based designs for SC agents based on the real-time participant's load information, serving cost, and reputation.
- Proposing a learning-assisted decision-making technique, namely, Deep Reinforcement Learning (DRL), to model intelligent SCs that can leverage the sequential data and adapt to the environment's dynamics to maximize the overall performance.
- Providing a comprehensive performance evaluation and analysis of the proposed method along with locally optimal heuristics as well as other planning techniques.
- Illustrating a case study that demonstrates the effectiveness of the proposed DRL-empowered SCs in a real-world application.

The rest of the paper is organized as follows: Related works on service provisioning over blockchains are explored in Section II. We introduce the system model in Section III and formalize the problem of interest in Section IV. We then propose the multiple design approaches in Section V and introduce their complexity analysis as well as their empirical performance in Sections VI and VII, respectively. Finally, we provide a concrete application through a case study in VIII and draw conclusions in Section IX.

II. Related Work

The proposed learning-based agent is based on Deep Reinforcement Learning (DRL), which is an artificial intelligence technique that tackles the problems of designing rational agents capable of making complex sequential decisions in a random environment to achieve maximum reward over time [19]. The success of RL-based optimization is mainly due to its ability to optimize utility functions effectively and, more importantly, adapt to changes in the optimized systems, leveraging the representation power of neural networks. Such adaptation leads to faster and better convergence to policies that maximize the utility function in dynamic systems. Due to this fact, DRL has been showing impressive results for service provisioning in dynamic domains whose entities might undergo a change with time. For example, Authors in [20]

jointly optimize resource allocation and user association in heterogeneous cellular networks for offloading stochastic mobile traffic and showed that it could achieve an optimal trade-off between the network utility and users Quality of Service (QoS). Task allocation in heterogeneous and dynamic (i.e., varying size) cloud clusters is studied in [21], where a set of jobs is sequentially allocated to a heterogeneous set of heterogeneous machines in a way that minimizes job completion time. In [22], the vehicles were modeled as edge nodes providing dynamic resources for the nearby user equipment. Then, the task allocations problem (in this case, which node to offload to) is shown to be non-convex, and RL techniques are utilized to solve such assignment problems. While these studies give insights into the potential of RL in dynamic environments for service provisioning, they do not directly model our case of interest. We investigate the joint optimization of the service cost and operation cost of the heterogeneous service providers in the continuously evolving data in blockchain networks.

For blockchain-based service provisioning, RL is increasingly used to tackle the complicated decision-making problem that arises in such a complex network. For example, the authors in [23] model blockchain peers as agents and optimize their offloading decision of various blockchain tasks (i.e., mining and regular processing tasks) in the context of mobile edge computing. The same goal is also addressed by [1]. However, the offloading is jointly optimized with other blockchain parameters such as power allocation, block size, and block interval. The work in [24] considers the privacy issue in such offloading processes and proposes quantifying the privacy levels of regular privacy tasks and adding this level in the offloading decision. These works differ from the current one in agent modeling; blockchain peers are modeled as agents to make intelligent decisions regarding the offloading of blockchain tasks (mining and regular processing) with the aim of improving blockchain transaction throughput. In contrast, we model the SCs deployed on the blockchain as the agents whose decisions are the allocations of general-purpose service provisioning logic written in them. This is independent of the blockchain/peers working mechanism itself. In general, there are multiple considerations and implications of selecting which system modules are to be "agentified," and hence, readers are referred to [25] for extensive discussion on agent selection. Nonetheless, it is conveyed that more investigation is needed in the "SCs as agents" paradigm.

In general, most of the literature research on task allocation in blockchain and SCs implemented static rules. However, this approach misses the opportunity of leveraging the ever-expanding data of blockchain [26]. An initiative towards integrating online control that leverages the blocks' data as feedback is introduced in [27]. The authors propose using control loop-style SCs to continuously tune participants' trust scores according to their recent behaviors/performance. However, the authors focus only on developing an online trust score. The allocation aspect of service provisioning was not considered.

In the current work, we envision SCs as *rational agents*, which means that they make the task allocation decisions

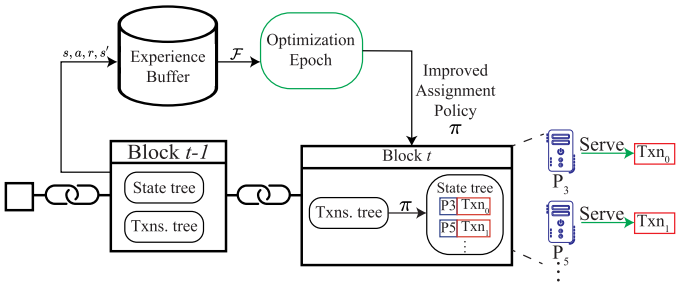

Fig. 1. Task allocation for transactions (service requests) to service providers (indicated as "P") in the blockchain.

intelligently to maximize the utility based on the data stored on the chain. Specifically, The chained data captures information about the performance of previous allocations in the form of a reward signal. Given these previous rewards, the allocation policies can be further optimized and improved at every new block generation (i.e., online) to maximize the utility (i.e., the total expected sum of rewards). We design the reward signal to reflect the bi-objective criterion of maximizing requesters' QoE, which is calculated based on multiple factors such as a blockchain-specific reputation score, as well as maintaining providers' loads in desired levels. To the best of our knowledge, such an ambitious framework has only been investigated in [18]. The authors address the current lack of intelligence in SCs systems and motivate the design of "rational" contracts that can make decisions based on the available data on-chain as well as their recent experience (i.e., transaction results), so as to maximize a given utility.

This paper is concerned with the design of such autonomous SCs in the context of IoT service provisioning. Specifically, we aim to design SCs that can allocate tasks to service providers in a way that maximizes the overall social welfare as measured by the global utility function. The utility function jointly considered the serving cost that the task requester will endure and the operation cost that the service provider will endure while considering the credibility of such service providers. Such SCs are an essential step towards realizing Decentralized Autonomous Organizations (DAO), which is a term used to describe the self-executing and decentralized code of SCs that are equipped with intelligent decision-making, also referred to as "agent SCs" [28].

III. System Model

There are two types of participants in a service provisioning SC; Service providers that offer services and users that submit requests for services in the form of transactions to the concerned SC. The service providers can be a base station offering spectrum sharing services, IoT devices providing remote data sensing, smart meters offering energy in P2P energy trading, or simply individuals. The blockchain is by design agnostic to the type of participant. We denote the set of N service providers as $\mathcal{I} = \{0, 1, 2, \ldots, N\}$, and the set of transactions $\mathcal{O} = \{0, 1, 2, \ldots, O\}$.

Fig. 1 shows the envisioned blockchain-based model that leverages the chained data to continuously improve the policy;

At a given point in time, a block would contain performance indicators of the allocation policy being followed by the SC (e.g., participants reviews), which can be cast in terms of an experience tuple of state s, action a, reward r, and next state s', as will be detailed in the following section. Based on these indicators, an optimization epoch is performed to improve the allocation policy and use it in the assignment to be done in the most recent block. The dynamically changing allocation policy, by means of learning, forms a promising paradigm that we investigate in this work.

Note that the assignment decisions and the resulting performance, as captured by the reward, are saved in the SC (i.e., inside the block). Then, at the following block (i.e., next time step), the agent's optimization process retrieves a batch from the experience buffer, which is utilized in improving the policy for future blocks and assignments. Thus, the chained data, which contains past assignments and their results on users' QoE and providers' load, is utilized in an online learning process that aims to improve the assignment policy.

This system model is, in fact, agnostic to the specific type of blockchain platform used and the SCs working mechanism. The model is based on the fact that the temporal structure of blockchain-based systems (i.e., blocks periodically generated and appended) makes them excellent candidates to leverage online-learning algorithms, especially that SCs are envisioned as autonomous entities that learn from the chained data. A major beneficiary from such gainful integration is service provisioning systems, whose policy determines the task allocation to maintain load balance across providers as well as users' QoE as determined by several factors such as trustable reputation score.

IV. Problem Formulation

In our formulation, we consider an agent interacting with an environment E in discrete time steps t. E will be a continuing environment, which means that its states keep transitioning with no terminal state, i.e., it has an infinite horizon. E is to be modeled by a Markov Decision Process (MDP). The MDP is a modeling framework of the multi-stage sequential decision-making problem. Generally, MDPs are defined as tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the states space, $\mathcal{A}$ is the actions space, $\mathcal{T}$ is the state transition probability given an action in a given state, $\mathcal{R}$ is the reward, and γ is the discount factor. In the sequel, we define the MDP's component in the context of service provisioning SCs.

1) State Space: At a given decision step t, a service request transaction $o \in \mathcal{O}$ should be allocated to a service provider $i \in \mathcal{I}$. The agent (i.e., SC) should decide on the allocation based on the state $s_t = \langle \mathbf{l_t}, \mathbf{c_t}, \mathbf{p_t}, d_t \rangle$ defined as follows:

- $\boldsymbol{l_t} = \{l_t^{(i)}\}$ for $i \in \mathcal{I}$, $l_t^{(i)} \in [0-1]$ is a vector representing the current load for each participant i. A load of 1 indicates a fully loaded participant that cannot serve users.
- $\boldsymbol{c_t} = \{c_t^{(i)}\}$ for $i \in \mathcal{I}$, $c_t^{(i)} \in [0-1]$ is a vector representing the normalized cost of serving the transaction o by each participant i. $c_t^{(i)} = 0$ indicates that the user cannot be served by participant i. 1 represents the highest cost.
- $\boldsymbol{p_t} = \{p_t^{(i)}\}$ for $i \in \mathcal{I}$, $p_t^{(i)} \in [0-1]$ is a vector of the normalized "reputation score" for each participant i. It can be used as a measure for the Quality of Experience (QoE) that this participant can provide. Such a score is popular and widely used in blockchain-based systems [29], [30], due to the previously explained provenance and immutability features, which made the blockchain one of the most reliable platforms for QoE data.
- $d_t \in [0-1]$ the normalized demand of transaction o. A demand of 1 is the maximum serviceable demand by the participants.

Such normalization of state elements is standard in the service provisioning system in order to focus on the system performance and abstract away units and unit conversions that might be specific to the application. Nonetheless, the normalized elements can always be interpreted or converted back to represent units of interest [21].

B. Action Space

Based on the state information, the action $\boldsymbol{a_t} = \{a_t^{(i)}\}, a_i \in \{0,1\}$, for $i \in \mathcal{I}$ is taken by the SC. $a_t^{(i)}$ represent whether the participant i is serving the user in time slot t. Note that while it is possible to serve the same user (transaction o) by multiple service providers, we study the case of individual assignments.

C. Transition Model

The state transition of the MDP defines the next state s_{t+1} based on the current state action pairs (s_t, a_t), we model the transition of the state elements as follows:

$$l_{t+1}^{(i)} = \begin{cases} l_t^{(i)} & \text{when } a_t^{(i)} \neq 1, \tau_k^{(i)} \neq 0 \\ l_t^{(i)} - u_k^{(i)} & \text{when } a_t^{(i)} \neq 1, \tau_k^{(i)} = 0 \\ l_t^{(i)} + (1 - \zeta^{(i)}) \times d_t & \text{when } a_t^{(i)} = 1, \tau_k^{(i)} \neq 0 \\ l_t^{(i)} + (1 - \zeta^{(i)}) \times d_t - u_k^{(i)} & \text{when } a_t^{(i)} = 1, \tau_k^{(i)} = 0 \end{cases}$$

This set of calculations describe how the load increases or decreases for a participant based on whether it was assigned a new task to serve or a task is released (i.e., its service time has ended).

The variable $\tau_k^{(i)}$ represents the number of time steps until a task from a previously assigned transaction k is released (user service is ended) from participant i, and can be calculated according to:

$$\tau_k^{(i)} = \frac{d_k^{(i)}}{\zeta^{(i)}} \tag{1}$$

Where $\zeta^{(i)}$ is the participant processing capabilities (PC) and is in the range of $[0,1]$. The term $(1 - \zeta^{(i)})$ represents the load increase factor, which is a characteristic of each participant as it depends on its PC. In other words, the service demand is

reflected differently on the participant's load according to their PC. For example, the same task can be insignificant to a workstation participant but causes considerable load on an embedded system participant. As for $u_k^{(i)}$, it represents the amount of load that will be released (set to free), after $\tau_k^{(i)}$ steps, for a participant i, which is equal to $d_{t^-} \times (1 - \zeta^{(i)})$ (i.e., the load due to assigning the user, at some previous time $t^- < t$, to participant i).

For modeling the reputation transitions, we adopt the Beta reputation model from [31]. This model builds the reputation of the participants based on the served users' feedback. Given a record of feedback for a participant, we assume the reputation Γ is a random variable that follows the Beta distribution, i.e., $\Gamma \sim Beta(\alpha, \beta)$, where α and β are the distribution parameters. The expected value of the reputation is expressed:

$$\mathbb{E}[\Gamma] = \frac{\alpha}{\alpha + \beta} \tag{2}$$

The parameters α and β in the reputation model are given by:

$$\alpha = n_+ + 1, \tag{3}$$

$$\beta = n_- + 1 \tag{4}$$

with n_+ and n_- denoting the number of positive and negative feedbacks respectively, and $n_+ + n_- = F$, where F is the total number of feedbacks. However, instead of considering positive and negative feedbacks only as discrete values (i.e., -1 for a negative feedback and +1 for a positive feedback), we can introduce a continuous feedback variable v, which can represent a single value of a feedback, where $v \in [-1, 1]$. Therein, the values n_+ and n_- can be re-expressed as:

$$n_+ = \frac{F(1+v)}{2} \tag{5}$$

$$n_- = \frac{F(1-v)}{2} \tag{6}$$

and then, from (2)-(6), the reputation expected value can be redefinded as:

$$\mathbb{E}[\Gamma] = \frac{F(1+v)+2}{2(F+2)} \tag{7}$$

To integrate this reputation model in our formulation, first, we define the feedback variable for a participant i at time t, as the following:

$$v_{t+1}^{(i)} = \frac{-\sum_{k=1}^{F_t^{(i)}} \hat{\tau}_k^{(i)}}{F_t^{(i)}} \tag{8}$$

where $\hat{\tau}_k^{(i)}$ is a normalization of the original value $\tau_k^{(i)}$ and falls in the range of $[-1, 1]$. The feedback variable is nothing but the negative average time that a participant takes to finish a transaction. More specifically, the feedback majorly depends on the participant's PC as the service time is conversely dependent on the PC. As such, participants with higher PC will receive better overall feedbacks. Finally, following (7), the reputation $p_t^{(i)}$ for a participant i at time t can be expressed as:

$$p_t^{(i)} = \frac{F_t^{(i)}(1 + v_t^{(i)}) + 2}{2(F_t^{(i)} + 2)} \tag{9}$$

It is worth mentioning that the value $F_t^{(i)}$ also represents the number of transactions served by the participant i up to time t. The other components of a given s_t are transitioned according to the following: $c_{t+1}^{(i)} \sim \mathcal{U}(0,1)$ and $d_{t+1} \sim \mathcal{U}(0,1)$.

D. Reward Structure

Since each service provider might have a different cost to service, it is desirable to assign users to low fees providers to save *service costs*. On the other hand, service providers have limited service capacities. In general, we assume that the more loaded the service provider is, the more *operation costs* it endures. Hence, the assignment should aim to also minimize the load across service providers.

In order to have viable assignments, the following constraints should hold:

$$l_t^{(i)} \leq 1, \forall i \in \{0, 1, \ldots, N\} \tag{10}$$

$$a_t^{(i)} = 1 \Rightarrow c_t^{(i)} > 0, \forall i \in \{0, 1, \ldots, N\} \tag{11}$$

The reward at times step t, r_t, can then be calculated as a function of the state and action pair (s_t, a_t) as:

$$r_t = \begin{cases} \underbrace{p_t^{(j)} \times (1 - c_t^{(j)})}_{\text{service cost saving}} + \underbrace{\frac{1}{N}\sum_{i=0}^{N}(1 - l_t^{(i)})}_{\text{operation cost saving}} & (10), (11) \text{ hold} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

where $j = i : a_t^{(i)} = 1$. The "service cost saving" term indicates the preference of lower service fees, scaled by the QoE, whereas the operation cost saving term indicates the preference to lower loads across service providers. Note that $r_t \in [0 - 2]$ as it is the sum of two normalized terms. Each provider sets the costs vector according to its own preferences. Then, given a set of offers (i.e., the cost vector), our task assignment objective, i.e., reward structure, aims to archive users' QoE, and provider load balance. As the MDP elements are now defined, we explore solution methods that lead to the optimal policy π^*.

V. Proposed Agents

In this section, we present the proposed designs of smart agents for service provisioning in IoT blockchain-powered systems.

A. Greedy Agents

We propose a set of heuristic-based service provisioning agents, which are greedy agents. This type of agents cares only about one specific criterion in the system. At any point in time, a greedy agent takes the action that maximizes this criterion. In what follows, we show various approaches for the design of this criterion.

1) The Greedy Reward (GR): The GR agent in the system simulates all the possible actions (i.e., tries to assign the transaction for each user) before taking a decision, and observes which assignment returns the highest reward according to (12), and choose this assignment as its decision.

2) The Greedy Load (GL): The GL agent observes the current participant loads and assigns the transaction to the participant with the lowest current load. Such an assignment is greedy due to the fact that it only looks at the current loads and disregards the transaction demand, participant serving cost, and reputation.

3) The Greedy Cost (GC): The GC agent observes the serving costs for all the participants and assigns the transaction to the participant with the lowest serving cost regardless of its current load, reputation, and the transaction demand.

4) The Greedy Cost Improved (GCI): Unlike the previous GC agent, the GCI agent takes into account the participant's load aspect. Ignoring the load aspect during the assignment might overload the participant. Thus, The GCI assign the transaction to the participant i with the lowest cost such that it satisfies the following condition:

$$l_t^{(i)} + (1 - \zeta^{(i)}) \times d_t \; < \; 1 \tag{13}$$

This condition prevents any assignment from overloading the participant. However, the GCI agent still disregards the participants' reputations and transaction demands.

5) The Greedy Reputation M (GREP-M): The GREP-M agent only considers M participants with the highest reputation and assigns the transaction to the participant with the lowest service cost from the considered ones. Since the GREP-M agent considers only high reputation participants, it indirectly considers the participant's PC. Moreover, by choosing the lowest service cost among these participants, it leverages all the aspects in the system, unlike the previous agents, where each one disregarded one aspect or more.

The majority of the proposed greedy agents are indeed light-weight in terms of complexity, but these agents do not take into account all the environment's aspects and ignore the fact that the environment is dynamic and follows a temporal structure, where future states changes depend on the current actions. Moreover, a good criterion might be hard to identify. For example, the GREP-M agent heavily depends on the hyperparameter M. Setting a proper fixed value for it is not trivial in an immutable SC, and the agent performance might be inefficient or unscalable for some values of M. In the next section, we propose an agent that will be trained on the environment changes over a horizon, such that it considers the dynamics of the environment. In addition, it can be improved to make use of a prediction model to forecast future information.

B. Reputation-Load Aware Agents

We further propose another heuristic-based agent, which is the reputation-load aware (RLA) agent. The RLA agent assigns the user to a participant with the lowest service cost whose load is less than a threshold ω, and reputation is greater than φ. The rationale behind such a heuristic is that by considering an upper threshold for assigning load, the agent is more likely not to experience overloading. Concurrently, to ensure the users' QoS, the agent will only assign the users to participants with acceptable reputations. However, choosing the right threshold for the load in a dynamic and stochastic system is a challenging task since the service demands and the serving costs vary immensely in the system. A possible estimator for the threshold can be the expected load of the participants after an assignment. Based on past experiences in a blockchain-powered system, we envision that using a regression model to estimate the threshold will make it change autonomously according to the instantaneous participant loads, service demands, and serving costs. Accordingly, we opt to use neural networks as a regression model. Specifically, the Long Short-Term Memory (LSTM) architecture [32]. LSTM neural networks in our service provisioning system can perform auto-regressive tasks and present the model that describes the time-varying service demands and serving costs. As for the training of the LSTM regression model, the training features X, and the targets Υ have to be defined. As such, the features will be all the states along a horizon H, where a single feature vector $\boldsymbol{x_n}$ can be the state at time t (i.e., $\boldsymbol{x_n} = s_t = \langle \mathbf{l_t}, \mathbf{c_t}, \mathbf{p_t}, d_t \rangle$). Whereas the target, we define it as the mean of the participants loads at time $t+1$, and can be given by:

$$\Upsilon_n = \frac{1}{|I|} \sum_{i \in I} l_{t+1}^{(i)} \tag{14}$$

Afterward, we define the loss function for the model as the Mean Squared Error (MSE) between the LSTM network output and the targets, which can be expressed as:

$$J(\Phi, \Upsilon) = \frac{1}{N} \sum_{n=1}^{N} (f_\Phi(\boldsymbol{x_n}) - \Upsilon_n)^2 \tag{15}$$

where Φ represents the LSTM model parameters, and $f_\Phi(.)$ is the model output. Minimizing such loss functions can be done by using the SGD algorithm. One can refer to [32] for more information about training LSTMs for regression tasks.

Nevertheless, if the RLA-LSTM agent's environment goes under drastic changes that have never been seen before, this agent fails in such cases. This is because the LSTM model has not been trained on such scenarios, and it gives predictions based on past experiences only, which does not represent the current experiences. Hence, in the next section, we propose an agent that can adapt to such an environment and its dynamics without the need to be retrained on the system's new dynamics.

C. DRL Agent

The DRL-based agent is a learning-assisted agent that continuously learns from interacting with the environment. At every decision epoch t, the agent receives a representation of the environment state $s \in \mathcal{S}$. The agent then executes an action $a \in \mathcal{A}$ using a policy $\pi(a|s)$, receives a reward $r_t \in \mathbb{R}$,

Algorithm 1. RLA-LSTM Service provisioning Agent

Input: LSTM model parameters Φ, reputation threshold φ
Output: j: The the serving participant index.

```
receive state s_t = ⟨l_t, c_t, p_t, d_t⟩
Estimate the future load using LSTM: ω ← f_Φ(s_t)
set j randomly
for i = 1 : I do
  j = {k | c_t^(k) = min{c_t}}
  Calculate the projected l̂_{t+1} for participant j
  if l̂_{t+1}^(j) ≤ ω & p_t^(j) ≥ φ then
    return j
  else
    c_t ← c_t \ c_t^(j)
  end if
end for
return j
```

and transition to the next state s' with probability $P(s'|s,a) = \mathcal{T}(s,a,s')$. The total feature discounted sum of rewards from time step t until some horizon H is denoted as $R_t = \sum_{t'=t}^{H} \gamma^{t'-t} r_t$, with the discount factor $\gamma \in [0,1)$. The state-action value function of a specific policy π is defined as $Q^\pi(s,a) = \mathbb{E}_{a\sim\pi, s'\sim\mathcal{T}}[R_t|s_t = s, a_t = a]$. It summarises the sum of rewards resulting from taking the action a in state s, and thereafter following policy π. The state value function $V^\pi(s) = \mathbb{E}_{a\sim\pi}[Q^\pi(s,a)]$ assesses the quality of a state when following the policy π. The advantage function is then defined as $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$, which reflects the advantage of taking action a in state s.

The optimal policy maximizes the Q-function $Q^{\pi^*}(s,a) = max_\pi Q(s,a)$ (hereafter referred to as Q^*). The goal of an RL agent is to find such optimal policy through direct interaction with the environment and without explicit or pre-encoded information about it, such as the transition probability $\mathcal{T}$. Q-learning finds such policy through firstly finding Q^* and then acting greedily with respect to it $\pi^*(a|s) = argmax_a Q^*(s,a)$ [19]. The optimal Q-function can be written recursively through the Bellman Optimally Equation [19]:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{T}}[r_t + \gamma \max_{a'} Q^*(s,' a')|s_t = s, a_t = a] \tag{16}$$

Q^* can be iteratively calculated through interaction with the environment using dynamic programming, where at each update iteration k (which is typically a time step t), the following Bellman update is calculated:

$$Q^*_{k+1}(s,a) = \mathbb{E}_{s'\sim\mathcal{T}}[r_t + \gamma \max_{a'} Q^*_k(s,' a')|s_t = s, a_t = a]$$

where Q_k is an estimate of Q^* at iteration k. As $k \to \infty$, Q_k converges to Q^* [19].

The Q-function (including the optimal one) can be of very high dimensionality or, as in our case, continuous. Thus, they should be approximated. Neural networks are general function approximators that proved successful in RL domains. Hence, we use a deep $Q-$network $Q(s,a;\theta)$ whose parameters are θ. We optimize those parameters using the Temporal Difference error (TD-error) loss function, which pushes the parameters in the direction that adequately approximate Q^* [19]. At each iteration k, the loss L is:

$$L_k(\theta_k) = (y_k - Q(s,a;\theta_k))^2 \tag{17}$$

where y_k is the TD-target defined as:

$$y_k = r_k + \gamma \max_{a'} Q(s,' a';\theta_k) \tag{18}$$

However, optimizing the above objective is likely to diverge or result in poor performance [33]. We utilize collective improvements from the RL community to stabilize learning. Namely, replay buffer, fixed targets, double estimation, and dueling network architecture. As illustrated in [33], keeping a replay buffer of previous experience (i.e., transition tuples s,a,r,s') and then optimizing (18) through stochastic gradient descent (SGD) greatly helps stability. In addition, the parameters used in the TD-target evaluation are frozen to some previous values $\bar{\theta}$ (fixed targets). The loss is then defined as:

$$L_k(\theta_k) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{U}(\mathcal{D})}[(y_k - Q(s,a;\theta_k))^2] \tag{19}$$

$$y_k = r + \gamma \max_{a'} Q(s,' a',\bar{\theta}) \tag{20}$$

Note that using the same network in (20) to choose the best action a' and to evaluate it can lead to over estimation bias. Thus, it is suggested that the freezed network is used for the evaluation of the action, wheras the online network is used for choosing that action [34], making the TD targets as:

$$y_k = r + \gamma Q(s,' \operatorname*{argmax}_{a'}(Q(s,' a';\theta_k));\bar{\theta}) \tag{21}$$

Of specific interest to this paper is the dueling network architecture introduced in [35], which is especially useful when multiple actions are approximately similar, which is the case in service provisioning. Estimating Q-function for every state-action pair might be impractical and slows down learning. This is because, in many states, the value of many actions might be either irrelevant or similar. For example, when two service providers have relatively similar states, then assigning one of them should provide some information about the value of assigning the other. We use the dueling network architecture, which employs multi-stream neural network design with two streams, one to estimate the state value function $V(s;\theta,\alpha)$ regardless of the action, and another stream to estimate the advantage function $A(s,a,\theta,\beta)$ where α and β are the parameters of the two streams, respectively. The two streams are then aggregated to provide the $Q-$values of a given state with every possible action, and the action with the high Q value is taken. As illustrated in [35], the simple sum aggregation may suffer from the identifiability issue. Hence, we use the aggregation in (22), which provides the best performance empirically.

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\alpha) + (A(s,a;\theta,\beta) - b) \tag{22}$$

Algorithm 2. DRL-based Service Provisioning Agent

Input: System's parameters,
Output: θ: The NN parameters for the approximation Q^*.
1: Initialize parameters of the online network θ randomly.
2: Initialize parameters of second (target) network $\bar{\theta} \leftarrow \theta$.
3: **for** episodes= 1:M **do**
4: Initialize state $s_t = \langle \mathbf{l_0}, \mathbf{c_0}, \mathbf{p_0}, d_0 \rangle$
5: **for** time step $t = 0 : L$ **do**
6: /**Interaction with the environment**/
7: Assign a service provider through selecting a_t based on ϵ-greedy policy
8: Execute a_t, observe s_{t+1} and r_{t+1}
9: Store the experience tuple $(s_t, a_t, s_{t+1}, r_{t+1})$ in $\mathcal{D}$
10: /**Updating the estimates**/
11: Randomly sample a minibatch $\mathcal{F} = \{(s_t^{(m)}, a_t^{(m)}, s_{t+1}^{(m)}, r_{t+1}^{(m)})\}_{m=1}^{|\mathcal{F}|}$ from $\mathcal{D}$
12: Calculate Q-targets using (21): $Y^{(m)} \leftarrow \{y^{(m)}\}_{m=1}^{|\mathcal{F}|}$
13: Fit $Q(s^{(m)}, a^{(m)}; \theta; \alpha; \beta)$ to the targets $Y^{(m)}$: $\theta \leftarrow \theta - \delta \nabla_\theta L(\theta)$
14: Every $target$ steps, update the target network $\bar{\theta} \leftarrow \tau\theta + (1-\tau)\bar{\theta}$

where $b = \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a'; \theta, \beta)$. The final algorithm used for training is provided in Algorithm 2. At every decision step (line 7), an assignment a_t is chosen to be either random (with probability ϵ, or the best action known so far (i.e., as determined by the network, with probability $1 - \epsilon$). This is known as $\epsilon-$greedy policy, and it allows for balancing the exploration-exploitation in RL agents. Then, the agent observes the next state and the reward. These values constitute a tuple of experience that is stored in the experience replay buffer $\mathcal{D}$ (line 11) and used for optimization (lines 11-13). Finally, the target network is softly updated towards the online network

Table I summarizes the presented strategies formulas and approaches. The parameters used for the DRL agents are listed in Table II. Note that these discussed strategies can be realized as software programs and implemented in SCs to be deployed in any general-purpose blockchain (such as Ethereum, Fabric, and others). Thus, they inherit the security features from the blockchain platform; these include, among others, code immutability, network availability, and autonomous execution. For security analysis of SCs, readers are referred to [36] which surveys security aspects guaranteed by SCs, and [37] which studies the development of secure SCs.

TABLE I
SUMMARY OF PROPOSED AGENTS' STRATEGIES

Strategy	Formula	Description
GR	$\boldsymbol{a_t} = \underset{\boldsymbol{a_t}}{\text{argmax}}\ r_t$	Selects the provider with the best expected reward.
GL	$\boldsymbol{a_t} = \underset{\boldsymbol{a_t}}{\text{argmin}}\ l_t$	Selects the least-loaded provider.
GC	$\boldsymbol{a_t} = \underset{\boldsymbol{a_t}}{\text{argmin}}\ c_t$	Selects the least-expensive provider.
GCI	$\boldsymbol{a_t} = \underset{\boldsymbol{a_t}}{\text{argmin}}\ c_t$ s.t : $l_t^{(i)} + (1 - \zeta^{(i)}) \times d_t < 1$	Selects the least-expensive provider that is not expected to be overloaded.
GREP-M	$\boldsymbol{a_t} = \underset{\boldsymbol{a_t} \in \mathcal{M}, \lvert\mathcal{M}\rvert = M}{\text{argmax}}\ c_t$	Selects the least-expensive provider from $\mathcal{M}$—The list of top M reputable providers.
RLA	$\boldsymbol{a_t} = \underset{\boldsymbol{a_t}}{\text{argmin}}\ c_t$ s.t : $\hat{l}_{t+1}^{(i)} \leq f_\Phi(s_t)\ \&\ p_t^{(i)} \geq \varphi$	Selects the least expensive provider whose load is not expected to surpass the best projected threshold, and meets a minimum reputation.
DRL	$\boldsymbol{a_t} = \underset{\boldsymbol{a_t}}{\text{argmax}}\, Q^*(s, a)$ where $Q(s,a)^* = \mathbb{E}_{a \sim \pi^*, s' \sim \mathcal{T}}[R_t \lvert s_t = s, a_t = a]$	Selects participant that maximize the future expected sum of rewards.

TABLE II
SERVICE PROVISIONING SYSTEM PARAMETERS

Parameter	Value
Learning episodes M	3×10^4
Episode length L	50
Service providers N	10
Discount factor γ	0.9
Exploration rate ϵ	1, with 5×10^{-4} Decay
Q-Network arch. & layers	1 common, 2 streams, 2 layers/stream
Q-Network neurons/layer	$51, 256$, streams :$(128, 10)$
Q-Network learning rate	10^{-4}
Activation function	Leaky ReLU, 0.01 -ve slope
Optimizer	ADAM [39]
Replay buffer size $\lvert\mathcal{D}\rvert$	2.5×10^5
Batch size $\lvert\mathcal{F}\rvert$	64
Soft update factor τ	10^{-4}
Soft update period $target$	4

VI. AGENTS' COMPLEXITY ANALYSIS

In this section, we study the complexity of our proposed algorithms against the Model Predictive Control-based (MPC-based) planning baseline [38]. The used variant of MPC planning method performs exhaustive search over a horizon η among all combinations of $(\mathbf{a_0}, \mathbf{a_1}, \ldots, \mathbf{a}_{\eta-1})$. MPC then observes the future reward from each combination and takes the current action that resulted in the highest future reward. The complexity of this MPC variant, given an action space $\mathcal{A}$ and a horizon η, is $O(|\mathcal{A}|^\eta)$, which means that the MPC complexity grows exponentially with longer horizons η, as the search space over the action combination becomes huge. It is worth mentioning here that the cardinalities of the sets $\mathcal{A}$ and $\mathcal{I}$ are equal (i.e., $|\mathcal{A}| = |\mathcal{I}|$). As for the greedy algorithms, the GR, GL, GC have time complexity of $O(|\mathcal{A}|)$, where they only perform a linear search for the maximum reward, the minimum participant load, and its serving cost respectively. As for the GCI, the algorithm searches for the participant with the lowest serving cost, such that it satisfies the condition (13), where the worst-case scenario is presented when only the participant with the highest cost serving cost satisfies the condition. Hence, the GCI's time complexity is $O(|\mathcal{A}|^2)$. The GREP-M algorithm search for M participants with the highest

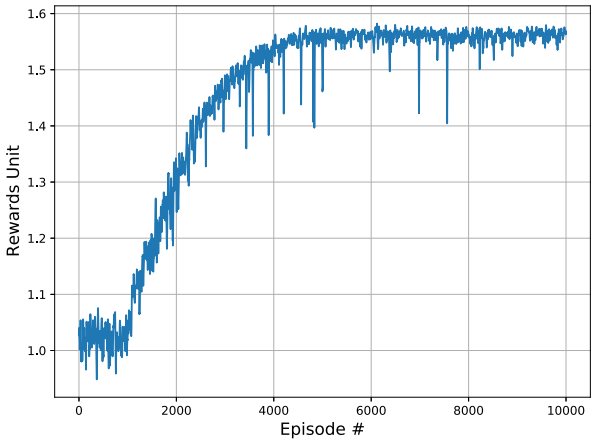

Fig. 2. RL agent training rewards over the episodes.

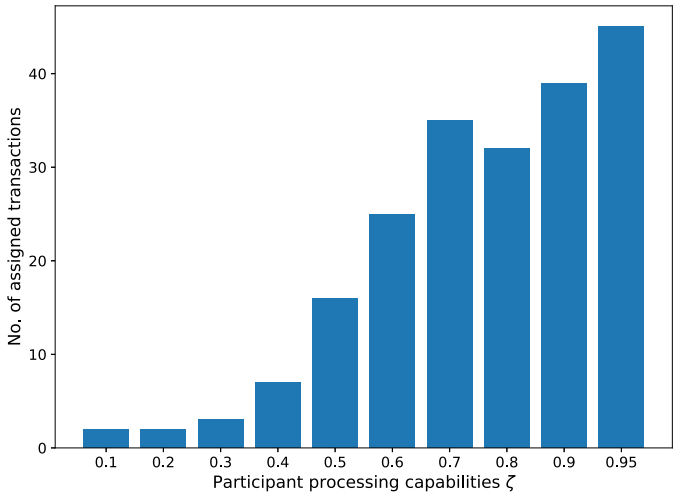

Fig. 3. Number of assigned transactions versus participant capabilities according to the learned policy.

reputation and then perform a linear search for the lowest serving cost over them, where the complexity of selecting the highest M participants is $O(M * |\mathcal{A}|)$. Therefore, the overall time complexity for the GREP-M is $O(M * (|\mathcal{A}| + 1))$.

Regarding the RLA algorithm, the vanilla version (i.e., no-load threshold forecasting) with given parameters ω and φ searches over the participants serving cost such that the participant's reputation is higher than φ and load is lower than ω. Thus, the time complexity for the vanilla RLA is $O(|\mathcal{A}|^2)$. The RLA-LSTM algorithm uses LSTM neural networks. For a single forward pass in the LSTM, it has been shown in [32] that the time complexity is $O(|\Phi|)$, where $|\Phi|$ is the number of parameters of the LSTM model. Hence, the RLA-LSTM has an overall time complexity of $O(|\Phi| + |\mathcal{A}|^2)$. However, before deploying the LSTM model in the algorithm, it needs to be pre-trained. The training complexity for the LSTM, with e epochs, and input size of $|\mathcal{S}|$, is $O(e * |\mathcal{S}| * |\Phi|)$.

The DRL agent employs the Deep Q-Network (DQN), where the DQN is a neural networks-based algorithm, and it has a simple architecture that is the Multi-Layer Perceptrons (MLPs) architecture. Therefore, the complexity of the algorithm is nothing but the complexity of the forward pass in MLP. In fact, a forward pass in MLP is only vector-matrix multiplication. Generally, in the MLP architecture, there exist hidden layers and an output layer, where each layer has a weight matrix $(k \times j)$, where k and j denotes the number of neurons in each layer, and each layer's output is a vector that has the size of that layer. Then, for a vector that has size k and a matrix of size $(k \times j)$, the vector-matrix multiplication complexity is $O(k * j)$, which can represent the multiplication complexity between the layers. Hence, for a single forward pass in an MLP, which has two hidden layers with sizes j and k respectively, with an input vector of size $|\mathcal{S}|$ and output layer of size $|\mathcal{A}|$, the time complexity for one forward pass in the neural network is $O(|\mathcal{S}| * j + j * k + k * |\mathcal{A}|)$. As a result, since we select the element with highest value from the DQN output, the overall time complexity is $O(|\mathcal{A}| + (|\mathcal{S}| * j + j * k + k * |\mathcal{A}|))$. It is worth mentioning that the aforementioned complexities are for a single full node execution in the blockchain. However, to calculate the overall system complexity, since each algorithm should be executed in every full node, each algorithm complexity should be multiplied by the number of the full nodes in the system $|\mathcal{A}|$ (e.g., the overall complexity of a system employing the GC algorithm is $O(|\mathcal{A}|^2)$).

VII. Agents' Performance Analysis and Simulation Results

In this section, we first analyze and discuss the DRL agent training procedure, then the RLA-LSTM agent, and finally, we show the comparison against the heuristics.

A. DRL Agent Training Analysis

In the training process of Algorithm 2, we set the values of the hyperparameters as in Table II. These values were set empirically in order to maximize performance. Fig. 2 shows the obtained rewards during training throughout the episodes. For the first 1000 episodes, the agent's policy is fully exploratory; Actions are taken in a random manner. After that, the agent starts following the ϵ-greedy policy, where the agent slowly learns a smarter behavior with time and learning to achieve the desired trade-offs in the system that results in the highest rewards. The convergence occurs approximately after 5000 episodes, with an average reward unit of $\sim$1.57.

After convergence, we extract and study the RL agent's learned policy in terms of the number of submitted transactions and their demands with respect to the participant PC. We execute the policy on 1000 testing episodes, and we analyze the behavior of the agent and the assignment patterns in the system. Fig. 3 depicts the number of assigned transactions for each participant according to its PC. The RL agent learned a policy that assigns the participants with the highest PC most of the transactions. Assigning transactions to such participants will prevent overloading the other ones, where the added load will be insignificant to those high PC ones. In fact, high PC participants finish their assigned transactions faster than the others, where the chances of overloading them become less. Moreover, Fig. 4 shows a statistical analysis for the demands

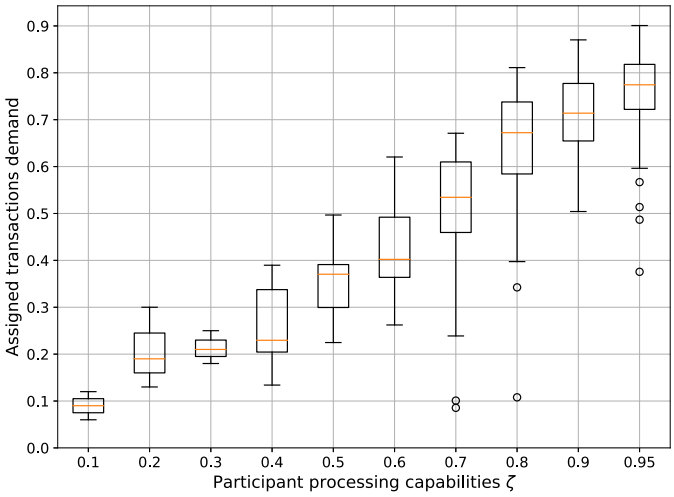

Fig. 4. Assigned transactions' demands according to the learned policy.

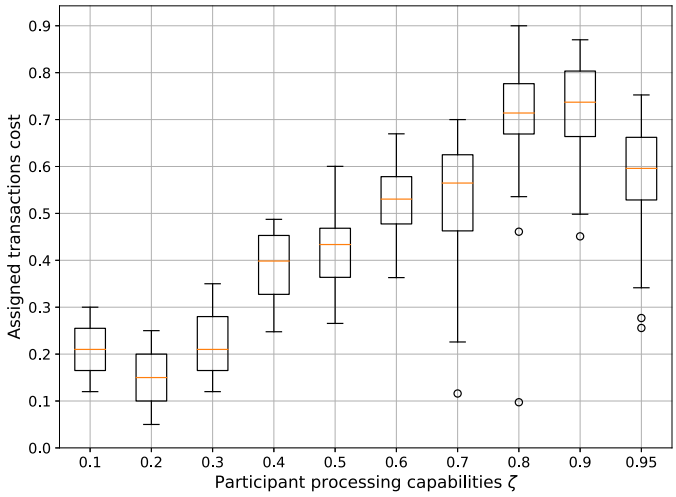

Fig. 5. Participant serving costs for the assigned transactions according to the learned policy.

of the assigned transactions with respect to the participant capabilities, according to the RL agent learned policy. It can be seen that the transactions with the highest demands are assigned to the participants with higher PC, and transactions with lower demands are assigned to the participants with low PC. By doing so, none of the participants will be overloaded, where the transaction assignment is based on the demand and participant PC, such that the added load for the participant is minimal. Nevertheless, the learned policy might not be optimal in terms of serving costs. In Fig. 5, we show the serving costs for the assigned transactions with respect to the participant PC. As can be seen from the figure, the costs are higher for the high PC participants. This is due to the fact that the policy avoids overloading the participants, even on account of high serving cost, since overloading the participants results in a 0 reward.

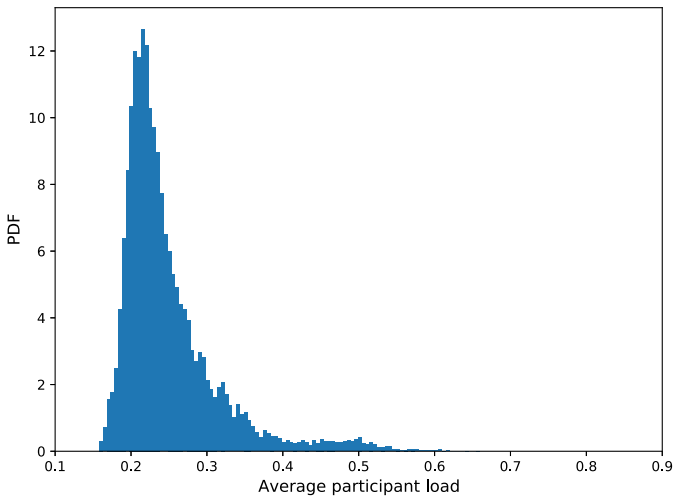

Fig. 6. The distribution of the average load in the LSTM model training data.

B. RLA-LSTM Agent Analysis

For training the LSTM regression model, the training data has been collected from the RL agent experience buffer after convergence. Specifically, we have collected 20×10^4 sequential states from the experience buffer to be the LSTM model training data, and the targets are the expected loads at each state. Fig. 6 shows the statistical distribution of the average participant load in the collected data. It can be seen that the average participant load is low, which indicates that the RL agent after convergence can achieve the optimal performance in terms of keeping the participant load as low as possible. In fact, this data is adequate for the LSTM model, as it can help the model to forecast the optimal load threshold at any given state, and by doing so, this gives a degree of freedom for the RLA agent to look for the participant with the minimum cost while its load does not exceed the predicted threshold.The data has been split into training and testing data, with ratios of 70% and 30%, respectively. Fig. 7 shows the loss of the model during training. The convergence of the model occurs approximately after 150 training epochs.

Lastly, in Fig. 8 we show the real-time prediction error for the LSTM model while the model is deployed in the environment. It can be noticed that the prediction error is insignificant, which indicates that the model can accurately predict the average load in the next time step by only looking at the current state of the environment.

C. Performance Comparisons

To illustrate our proposed agents' performance, we compare them against an agent that employs exhaustive planning with a certain horizon (i.e., MPC), which will be the benchmark of the comparisons, in addition to an agent that employs the random policy (i.e., the agent takes actions randomly), which represents the baseline in the comparisons. Fig. 9 depicts the comparisons between the agents in terms of rewards obtained while these agents are deployed in the environment E. In Fig. 9(a), the comparison between the RLA agents against the benchmark and baseline can be seen. Since the RLA agents need the reputation threshold φ hyperparameter to be set, we

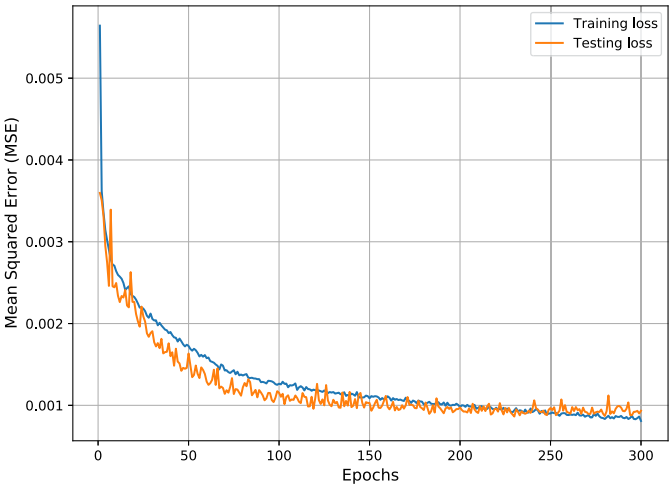

Fig. 7. The MSE loss for the LSTM regression model.

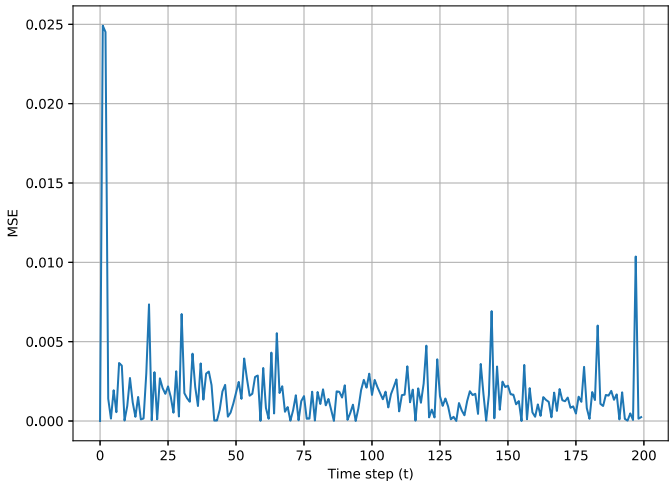

Fig. 8. The MSE loss in testing for the LSTM regression model.

performed a grid search over the parameter to obtain the best performance out of these types of agents, in addition to an intermediate performance for the sake of comparison. It has been found that a reputation of 0.4 is the optimal threshold. It can be noticed that all the RLA agents perform better than the baseline, namely, the random agent. In addition to that, a notable pattern can be seen, with increasing the load threshold ω, the vanilla RLA agents' performance increases and then decreases, along with some fluctuations in the performance. Having said that, we can conclude that there exists an optimal value for the threshold, which changes with time. As a result, since the vanilla RLA agents have their load threshold fixed and do not represent the optimal threshold, they fall behind the planning agent. However, after introducing the LSTM prediction model to the RLA agent in order to predict the load threshold, we can notice that its performance has been enhanced as it achieves a similar performance to that of the planning one.

The performance of the greedy agents is shown in Fig. 9(b). Similar to the RLA agents, all the greedy agents outperform the baseline. It can be noticed that the GL and the GC are the worst of the greedy agents. This is because the GL and the GC agents care only about one aspect, whether the load or the serving cost, and disregard the other, while the goal of the optimization is to optimize the serving cost of the participant load jointly. Moreover, it can be noticed that the GR agent is better than the previously mentioned two. In fact, the GR agent obtains the best immediate reward, which is based on minimizing the serving cost and the participant loads. However, since the GR takes its actions based on the immediate rewards only, its performance is not optimal in the long run. The GC-I agent enhances the GC agent's performance, achieving a closer performance to the benchmark one. This enhancement results from the fact that the GC-I agent avoids overloading the participants while looking for the minimum serving cost. Nevertheless, it ignores the reputation aspect, which prevents it from being more closer to the benchmark. Considering the reputation aspect, along with looking for the minimum serving cost, is what made the GREP-M agents achieve similar performance to the benchmark one. Indeed, the GREP-5 agent outperforms the planning agent. Even though the GREP-M agent might not perform well when increasing the number of agents in the environment, its algorithm complexity grows linearly with the number of agents, unlike the MPC planning agent algorithm, which is exponential.

In Fig. 9(c), the DRL agent performance is shown. One can see that the DRL agent outperforms the benchmark. This is attributed to the longer planning horizon of DRL, represented by the expectation of the sum of future rewards, as opposed to the limited samples of the rewards that MPC uses to deduce a current action. Finally, in Fig. 9(d), we show a closer look to compare the best agents. The RLA-LSTM agent achieves similar to the planning agent, while the GREP-5 outperforms the planning agent, and the DRL agent achieves the optimal performance.

D. Heuristic and Planning Agents in a Non-Stationary Environment

In this section, we study the performance of the agents under major changes in the environment. In general, all agents have access to state information, which is a part of the SC. Thus, they do know major changes as they occur (e.g., providers joining/leaving, or providers load increase/decrease). However, the DRL method is able to generalize across the state space and respond near-optimally to states not seen before, and continue improving, as opposed to re-running an optimization. Hence, the adaptation in blockchain networks is of utmost importance for optimized performance.

At some point in time, the participants declare that their PCs have been reduced to half. Fig. 10 shows the rewards obtained under the environment change throughout the testing episodes. It can be seen that learning-assisted agents, namely the RLA-LSTM and the DRL agents, suffer the most from this hit. This results from the fact that these agents face new experiences that they have never seen before, nor have they been trained on. However, since the DRL agent is experience-driven, it adapts to the new changes and learns the new dynamics of the

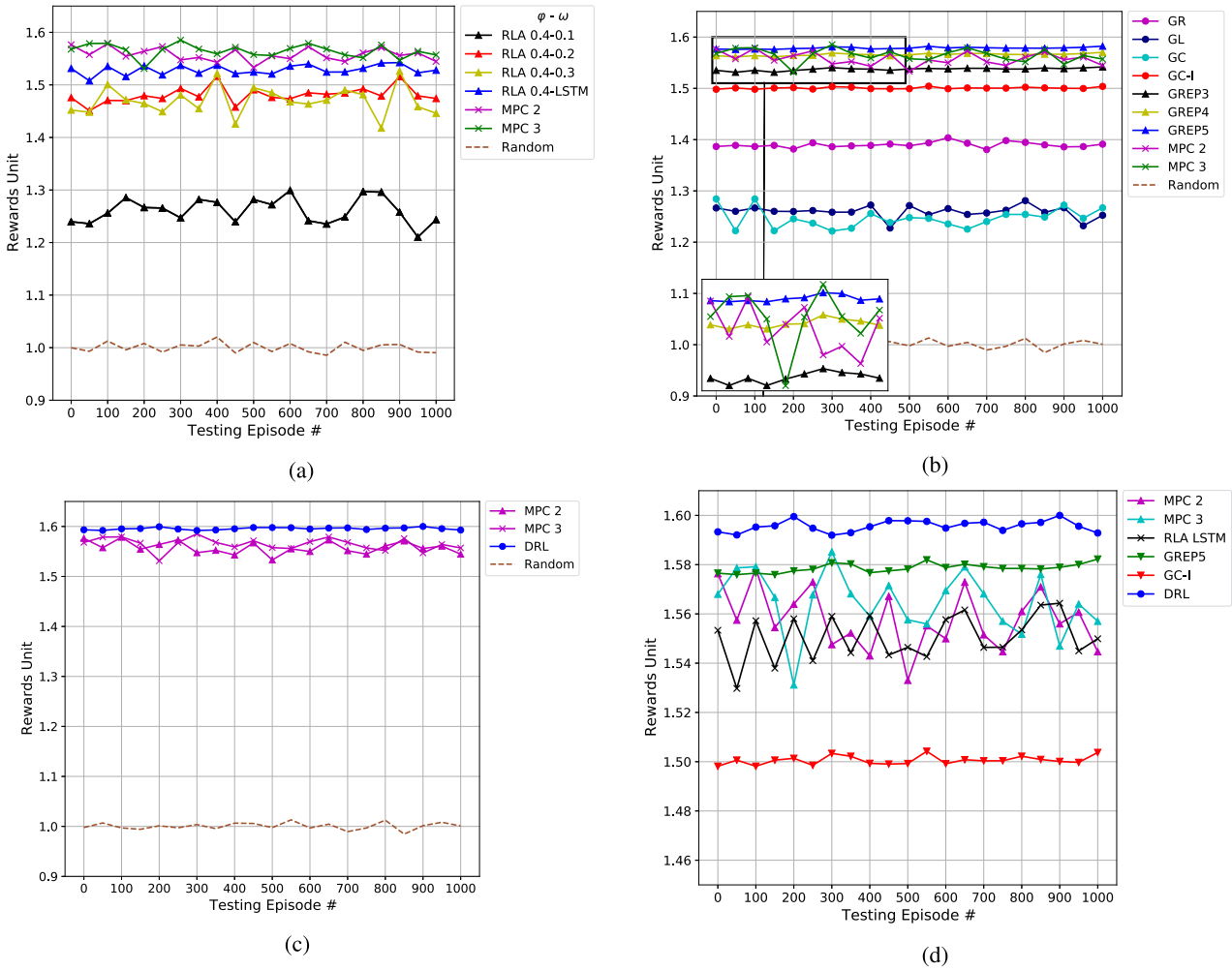

Fig. 9. Performance comparison in terms of rewards unit, for agents employing MPC and random policy against and (a) RLA agents, (b) greedy agents, (c) DRL agent (b) top performing agents.

environment, where it can change its policy that maximizes the obtained rewards in the new environment. Unlike the RLA-LSTM agent, the DRL agent does not need to be retrained on the new experiences from scratch to perform reasonably. The heuristic and planning agents perform similarly in the new environment but far behind the DRL agent after adapting to the new environment. Hence, it can be concluded that online and experience-driven learning is of utmost importance in a dynamic environment.

VIII. Case Study

In order to demonstrate the effectiveness of the proposed DRL-based technique, we introduce a concrete use-case in the area of edge computing. In edge computing, computational offloading is an emerging paradigm wherein computational tasks can be offloaded from end-user devices to the edge server, edge server to the end-user devices, or the user devices themselves. These offloading decisions are made depending on the required computation, as well as the communication cost of the data to be processed [40]. In our use case, we focus

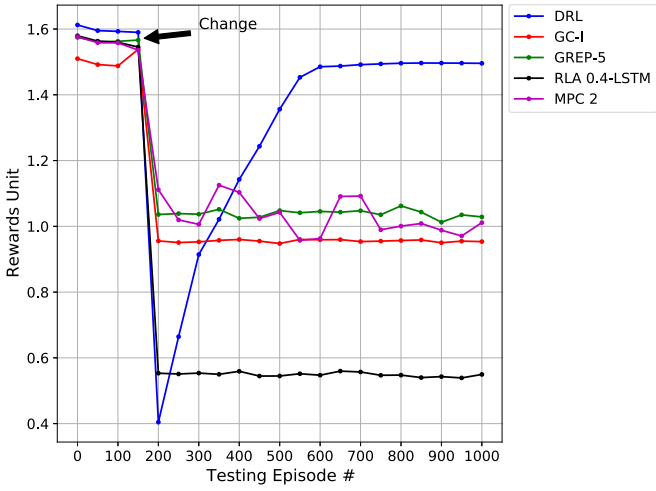

Fig. 10. Agents performance after introducing dynamic changes.

on the cases where the edge nodes offload tasks to end-user devices, which is used when the data is available at (or can effectively be communicated among) the end-user devices but

TABLE III
STATE VARIABLES MAPPING TO THE USE-CASE APPLICATION

Variable	Meaning	Unit
Processing capabilities	CPU Frequency	$cycle/second$
Serving cost	Fees	$\$/cycle$
Demand	Cycles required for a task	$cycles$
Participant Load	CPU Usage	Normalized
Time step	Unit of processing time	$second$
Reputation	Performance score	Normalized

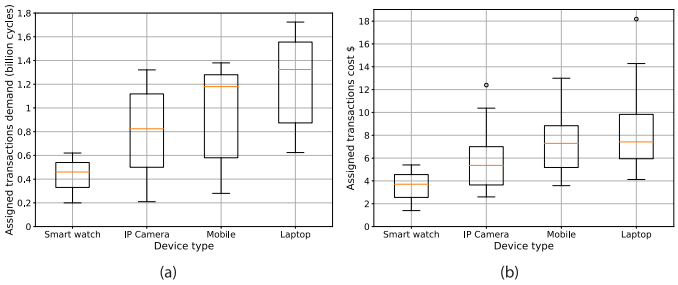

Fig. 11. Offloading policy statistics (a) Assigned transaction demands in the offloading scheme. (b) Assigned transaction costs in the offloading scheme.

uploading it to the edge node is expensive. In such a case, the edge nodes need to decide which end-device to perform a specific task. Table III projects the abstract state variables onto the edge computing offloading use-case.

We simulate a scenario with 4 different end-user devices whose CPU frequencies are $[0.2, 0.4, 0.6, 0.8] \times 10^9 cycles/second$. The serving cost is generated in the range $[10^{-9}$ to $10^{-8}]$ $\$/cycle$ uniformly distributed. The demands of transactions are generated from the range [0.1 to 2]$\times 10^9 cycles$. We train an agent with the same hyperparameters illustrated in Table II, then run its learned policy under those settings. We then analyze the policy by showing the average demands assigned for each participant (Fig. 11.a), as well as the average serving costs of the assigned transactions for each device (Fig. 11.b). The assigned transactions demands and their serving costs follow the same expected pattern discussed in the results sections; More powerful participants are assigned the more demanding tasks. They might as well get assigned to less demanding tasks. In contrast, devices with low CPU frequency are never assigned tasks expected to overload them. In terms of serving costs, it is more varied for the more capable participants, which indicates that more computing tasks are allocated to them. In general, the edge node is offloading (i.e., distributing) tasks to the end-user device with the aim of having minimal execution time but in a manner that does not overload any, (as overloading might be reflected as more excessive energy usage or even causes the device to freeze/crash). This policy is fully data-driven and learned from online experience without directly modeling each device's expected load increments or specific demands pattern. Lastly, in Fig. 12 we show the execution time taken in the system for a single assignment with respect to the algorithms. It can be seen clearly that the MPC 2 has the highest

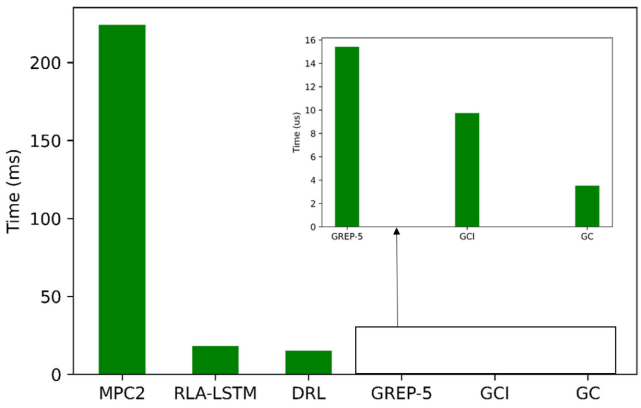

Fig. 12. Execution time comparison between the algorithms.

execution time, since it has exponential complexity. Whereas the RLA-LSTM and the DRL have similar execution times, but much lower than the MPC 2. As for the greedy algorithms, they have the shortest execution time in the system, as they are considered light-weight algorithms.

IX. CONCLUSION

In this paper, we investigated the design of smart agents for service provisioning SCs. We showed that heuristic-based and planning agents can have good performance in minimizing the overall cost, but are unreliable to execute in run-time, since their performance suffers when a shift in the system occurs. The proposed DRL-based method is able to achieve the highest utility and adapt to variations in the blockchain network, maintaining its superior performance. Future work can aim to quantify the speed of adaptability to different types of possible changes as well as the monetary execution cost of these adaptive agents in specific Blockchains. Overall, intelligent SCs are essential to the realization of Decentralized Autonomous Organizations (DAO) being sought after by academia and industry to achieve secure and intelligent applications.

ACKNOWLEDGMENTS

The findings achieved herein are solely the responsibility of the authors.

REFERENCES

[1] J. Feng, F. Richard Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6214–6228, Jul. 2020.
[2] O. Novo, "Blockchain meets IoT: An architecture for scalable access management in IoT," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1184–1195, Apr. 2018.
[3] C. Shen and F. Pena-Mora, "Blockchain for cities - A systematic literature review," *IEEE Access*, vol. 6, pp. 76787–76819, 2018.
[4] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[5] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data*, Jun. 2017, pp. 557–564.
[6] D. G. Wood, "ETHEREUM: A secure decentralized generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[7] S. Rouhani and R. Deters, "Security, performance, and applications of smart contracts: A systematic survey," *IEEE Access*, vol. 7, pp. 50 759–50 779, 2019.
[8] T. T. A. Dinh, R. Liu, M. Zhang, G. Chen, B. C. Ooi, and J. Wang, "Untangling blockchain: A data processing view of blockchain systems," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 7, pp. 1366–1385, Jul. 2018.
[9] S. E. Chang, Y.-C. Chen, and M.-F. Lu, "Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process," *Technological Forecasting Social Change*, vol. 144, pp. 1–11, Jul. 2019.
[10] D. Bumblauskas *et al.*, "A blockchain use case in food distribution: Do you know where your food has been?," *Int. J. Inf. Manage.*, vol. 52, 2020, Art no. 102008.
[11] P. Helo and Y. Hao, "Blockchains in operations and supply chains: A model and reference implementation," *Comput. Ind. Eng.*, vol. 136, pp. 242–251, Oct. 2019.
[12] M. Andoni *et al.*, "Blockchain technology in the energy sector: A systematic review of challenges and opportunities," *Renew. Sustain. Energy Rev.*, vol. 100, pp. 143–174, Feb. 2019.
[13] E. Mengelkamp *et al.*, "Designing microgrid energy markets: A case study: The brooklyn microgrid," *Appl. Energy*, vol. 210, pp. 870–880, 2018.
[14] H. Yao, T. Mai, J. Wang, Z. Ji, C. Jiang, and Y. Qian, "Resource trading in blockchain-based industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3602–3609, Jun. 2019.
[15] J. Abou Jaoude and R. George Saade, "Blockchain applications - Usage in different domains," *IEEE Access*, vol. 7, pp. 45 360–45 381, 2019.
[16] Y. Xu, G. Wang, J. Yang, J. Ren, Y. Zhang, and C. Zhang, "Towards secure network computing services for lightweight clients using blockchain," *Wireless Commun. Mobile Comput.*, vol. 2018, p. 12, 2018, Art. no. 2051693. [Online]. Available: https://doi.org/10.1155/2018/2051693
[17] Y. Shoham and K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. New York, NY, USA: Cambridge Press, 2008.
[18] G. Ciatto *et al.*, "Towards agent-oriented blockchains: Autonomous smart contracts," in *Proc. Adv. Practical Appl. Survivable Agents Multi-Agent Syst*. Cham: Springer, 2019, pp. 29–41.
[19] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[20] N. Zhao, Y.-C. Liang, D. Niyato, Y. Pei, M. Wu, and Y. Jiang, "Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks," *IEEE Trans. Wireless Commun.*, vol. 18, no. 11, pp. 5141–5152, Nov. 2019.
[21] M. Cheong, H. Lee, I. Yeom, and H. Woo, "SCARL: Attentive reinforcement learning-based scheduling in a multi-resource heterogeneous cluster," *IEEE Access*, vol. 7, pp. 153 432–153 444, 2019.
[22] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11 158–11 168, Nov. 2019.
[23] A. Asheralieva and D. Niyato, "Distributed dynamic resource management and pricing in the IoT systems with blockchain-as-a-service and UAV-enabled mobile edge computing," *IEEE Internet Things J.*, vol. 7, no. 3, pp. 1974–1993, 2019.
[24] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 4, pp. 2536–2549, Dec. 2020.
[25] C. Savaglio *et al.*, "Agent-based Internet of Things: State-of-the-art and research challenges," *Future Gener. Comput. Syst.*, vol. 102, pp. 1038–1053, Jan. 2020.
[26] M. Wu, K. Wang, X. Cai, S. Guo, M. Guo, and C. Rong, "A comprehensive survey of blockchain: From theory to IoT applications and beyond," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8114–8154, Oct. 2019.
[27] B. Shala, U. Trick, A. Lehmann, B. Ghita, and S. Shiaeles, "Blockchain and trust for secure, end-user-based and decentralized IoT service provision," *IEEE Access*, vol. 8, pp. 119 961–119 979, 2020.
[28] S. Wang, W. Ding, J. Li, Y. Yuan, L. Ouyang, and F.-Y. Wang, "Decentralized autonomous organizations: Concept, model, and applications," *IEEE Trans. Comput. Social Syst.*, vol. 6, no. 5, pp. 870–878, Oct. 2019.
[29] R. Dennis and G. Owen, "Rep on the block: A next generation reputation system based on the blockchain," in *Proc. 10th Int. Conf. Internet Technol. Secured Trans.*, 2015, pp. 131–138.
[30] E. Bellini, Y. Iraqi, and E. Damiani, "Blockchain-based distributed trust and reputation management systems: A survey," *IEEE Access*, vol. 8, pp. 21 127–21 151, 2020.
[31] R. Ismail and A. Jøsang, "The beta reputation system," in *Proc. 15th Bled Electron. Commerce Conf.*, 2002, pp. 2502–2511.
[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[33] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.
[34] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.
[35] Z. Wang *et al.*, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.
[36] T. Salman, M. Zolanvari, A. Erbad, R. Jain, and M. Samaka, "Security services using blockchains: A state of the art survey," *IEEE Commun. Surv. Tut.*, vol. 21, no. 1, pp. 858–880, Jan.–Mar. 2019.
[37] Y. Huang, Y. Bian, R. Li, J. L. Zhao, and P. Shi, "Smart contract security: A software lifecycle perspective," *IEEE Access*, vol. 7, pp. 150 184–150 202, 2019.
[38] D. P. Bertsekas, *Reinforcement Learning and Optimal Control*. Belmont, MA, USA: Athena Scientific, 2019.
[39] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. Accessed: Oct. 31, 2021. [Online]. Available: http://arxiv.org/abs/1412.6980
[40] P. Mach and Z. Becvar, "Mobile edge computing: A survey on architecture and computation offloading," *IEEE Commun. Surv. Tut.*, vol. 19, no. 3, pp. 1628–1656, Jul.–Sep. 2017.

Naram Mhaisen received the B.Sc. degree in computer engineering with excellence, from Qatar University (QU), Doha, Qatar, in 2017 and the M.Sc. degree in computing from the same university, in 2020. Since 2017, he has been a Research Assistant with the Department of Computer Science and Engineering, College of Engineering, QU. His research interests include the design and optimization of distributed and multiagent (learning) systems with applications to IoT.

Mhd Saria Allahham received the B.Sc. degree in computer engineering with excellence, from Qatar University, Doha, Qatar, in 2020. He is currently working toward the master's degree with Queen's University, Kingston, ON, Canada. He then was a Research Assistant with the Department of Computer Science and Engineering, Qatar University. His research interests include reinforcement learning, edge learning, and edge computing.

Amr Mohamed (Senior Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from the University of British Columbia, Vancouver, BC, Canada, in 2001 and 2006, respectively. From 1998 to 2007, he was an Advisory IT Specialist with IBM Innovation Centre, Vancouver, BC, Canada, taking a leadership role in systems development for vertical industries. He is currently a Professor with the College of Engineering, Qatar University, Doha, Qatar. He has more than 25 years of experience in IoT, edge computing, pervasive AI, and wireless networking research and industrial systems development. He has authored or coauthored more than 200 refereed journal and conference papers, textbooks, and book chapters in reputable international journals, and conferences. His research interests include wireless networking and edge computing for IoT applications. He holds three awards from IBM Canada for his achievements and leadership, and four best paper awards from IEEE conferences. He is serving as a technical editor for three international journals, has served as a guest editor in several special issues, and has served as a technical program committee (TPC), and co-chair for many IEEE conferences and workshops.

Aiman Erbad (Senior Member, IEEE) received the B.Sc. degree in computer engineering from the University of Washington, Seattle, WA, USA, in 2004, the master of computer science degree in embedded systems and robotics from the University of Essex, U.K., in 2005, and the Ph.D. degree in computer science from the University of British Columbia, Vancouver, BC, Canada, in 2012. He is currently an Associate Professor and ICT Division Head of the College of Science and Engineering, Hamad Bin Khalifa University (HBKU), Qatar. Prior to this, he was an Associate Professor with the Computer Science and Engineering (CSE) Department and the Director of research planning and development with Qatar University, Doha, Qatar, until May 2020. He was also the Director of research support responsible for all grants and contracts during 2016–2018 and as the Computer Engineering Program Coordinator during 2014–2016. His research interests include cloud computing, edge intelligence, Internet of Things (IoT), private and secure networks, and multimedia systems. He was the recipient of the Platinum Award from H.H. The Emir Sheikh Tamim bin Hamad Al Thani at the Education Excellence Day 2013 (Ph.D. category). He was also the recipient of the 2020 Best Research Paper Award from Computer Communications, the IWCMC 2019 Best Paper Award, and the IEEE CCWC 2017 Best Paper Award. His research received funding from the Qatar National Research Fund, and his research outcomes were published in respected international conferences and journals. He is the Editor of the *KSII Transactions on Internet and Information Systems*, the Editor of the *International Journal of Sensor Networks* (IJSNet), and the Guest Editor of the IEEE NETWORK. He also served as a Program Chair of the International Wireless Communications Mobile Computing Conference (IWCMC 2019), as a Publicity Chair of the ACM MoVid Workshop 2015, as a Local Arrangement Chair of NOSSDAV 2011, and as a Technical Program Committee (TPC) Member in various IEEE and ACM international conferences (GlobeCom, NOSSDAV, MMSys, ACMMM, IC2E, and ICNC). He is a Senior Member of ACM.

Mohsen Guizani (Fellow, IEEE) received the B.S. (with distinction), M.S., and Ph.D. degrees in electrical and computer engineering from Syracuse University, Syracuse, NY, USA, in 1984, 1986, and 1990, respectively. He is currently a Professor with Computer Science and Engineering Department, Qatar University, Doha, Qatar. Previously, he was with different institutions, which include the University of Idaho, Moscow, ID, USA, Western Michigan University, Kalamazoo, MI, USA, University of West Florida, Pensacola, FL, USA, University of Missouri-Kansas City, Kansas City, MO, USA, University of Colorado Boulder, Boulder, CO, USA, and Syracuse University. He is the author of nine books and more than 800 publications. His research interests include wireless communications and mobile computing, applied machine learning, cloud computing, security and its application to healthcare systems. He was listed as a Clarivate Analytics Highly Cited Researcher in Computer Science in 2019 and 2020. Dr. Guizani has won several research awards, including the 2015 IEEE Communications Society Best Survey Paper Award and four best paper awards from ICC and Globecom Conferences. He was also the recipient of the 2017 IEEE Communications Society Wireless Technical Committee (WTC) Recognition Award, the 2018 AdHoc Technical Committee Recognition Award, and the 2019 IEEE Communications and Information Security Technical Recognition (CISTC) Award. He was the Editor-in-Chief of the IEEE NETWORK and is currently serves on the Editorial Boards of many IEEE Journals/Transactions. He was the Chair of the IEEE Communications Society Wireless Technical Committee and the Chair of the TAOS Technical Committee. He served as the IEEE Computer Society Distinguished Speaker and is currently the IEEE ComSoc Distinguished Lecturer.