IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 31 March 2023, accepted 6 May 2023, date of publication 16 May 2023, date of current version 23 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3276859

RESEARCH ARTICLE

A Deep Reinforcement Learning Approach for Competitive Task Assignment in Enterprise Blockchain

GAETANO VOLPE, AGOSTINO MARCELLO MANGINI, (Senior Member, IEEE), AND MARIA PIA FANTI, (Fellow, IEEE)
Department of Electrical and Information Engineering, Polytechnic University of Bari, 70125 Bari, Italy
Corresponding author: Gaetano Volpe (gaetano.volpe@poliba.it)

ABSTRACT With the advent of Industry 4.0, the demand of high computing power for tasks such as data mining, 3D rendering, file conversion and cryptography is continuously growing. To this extent, distributed and decentralized environments play a fundamental role by dramatically increasing the amount of available resources. However, there are still several issues in the existing resource sharing solutions, such as the uncertainty of task running time, the renting price and the security of transactions. In this work, we present a blockchain-enabled task assignment platform by performance prediction based on Hyperledger Fabric, an open-source solution for private and permissioned blockchains in enterprise contexts that outperforms other technologies in terms of modularity, security and performance. We propose a model-free deep reinforcement learning framework to predict task runtime in agents current load state while the agent is engaged in multiple concurrent tasks. In addition, we let clients choose between prediction accuracy and price saving on each request. This way, we implicitly give inaccurate agents a chance to get assignments by competing in price rather than in time, allowing them to collect new experiences and improve future predictions. We conduct extensive experiments to evaluate the performance of the proposed scheme.

INDEX TERMS Blockchain, cloud, deep reinforcement learning (DRL), resource sharing.

I. INTRODUCTION

The migration from on-premise to Cloud-based platforms has been rising over the past years. The public Cloud *infrastructure as a service (IaaS)* spending forecast for 2023 is of 156 billions U.S. dollars, whereas in 2019 it was only 45 [1]. Consequently, due to the high requirements of several application categories, the rapid increase of company investments in the Cloud leads to a continuous growing of computing power, memory and storage demands.

In this context, scientific workflows, data mining algorithms, file format conversion (e.g. video and audio file), 3D rendering and cryptography-based solutions are only some examples. All of those applications are resource consumption intensive in terms of CPU and memory and it is often required in enterprise environments that they are completed in the least time possible, especially when they are part of a production flow.

However, due to the high volatility of resources availability in the Cloud, application runtime is not always static and strictly depends on the assigned environment current workload, therefore it needs to be somehow estimated. This problem is well known in literature as performance prediction and it has been addressed since 2005 with different techniques based on the continuous collection of resource consumption data such as CPU and memory usage. In [2], a systematic review of performance prediction of parallel applications is provided. The authors observe that in the 81.7% of considered papers, the estimation is based on analytic methods in which the corresponding equations are either manually or automatically derived mostly by stochastic linear and non-linear approaches. On the other hand, in the remaining 18.29%,

The associate editor coordinating the review of this manuscript and approving it for publication was Abderrahmane Lakas.

This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 License.
For more information, see https://creativecommons.org/licenses/by-nc-nd/4.0/

non-analytic methods, such as *Support Vector Machines* or *Artificial Neural Networks* are used. Although most of these approaches are quite accurate, they cannot be easily used in modern Cloud environments for at least three reasons:

i) they are too application-specific or platform-dependent and, therefore, they have a limited field of application;
ii) they are intended for single-core processors or for a specific class of multi-core architectures and they do not consider the vast class of available platforms;
iii) they assume a free workload context in which the underlying resources are not being used by other concurrent processes, which is unrealistic in a typical Cloud environment.

Indeed, these limitations are reflected in some recent works. For example, in [3], the proposed performance prediction approach is restricted to scientific applications on *HPC Cluster* platforms. In contrast, the method proposed in [4] is not application-specific but it is restricted to *Grid* environments. Moreover, many of the existing works, such as [5] and [6], suggest an offline approach, whereas all modern applications require an online continuous incremental learning.

As a result, due also to the heterogeneity of the applications and the volatility of configuration and availability of the underlying resources, the existing methods are not generally applicable in a Cloud environment.

In alternative, the adoption of Reinforcement Learning-based (RL) techniques is a valid approach. In [7] and [8], the authors propose RL-based methods for solving specific resource management and task mapping problems and minimizing application runtime by prediction. However, [7] is restricted to the problem of optimizing the performance of communication bound applications on parallel computing systems. In contrast, [8] proposes a mapping problem for generic tasks in a multi-resource cluster but, in its prediction approach, it assumes that the resource demand for each task is known in advance. When it comes to Cloud environments, this is certainly a strong assumption as the task demands are not necessarily known upon arrival on the system.

In addition, decentralization is another key factor in Cloud environments. In fact, in a typical scenario multiple resources are available from different locations on a variable time-based fee. The resulting resources trading problem among nodes poses several security and integrity issues for transactions that have been addressed in recent literature with auction systems implemented on Blockchain, a popular secure distributed ledger technology, and combined with RL-based techniques for various optimization objectives such as maximizing participants payoff, minimizing energy consumption or adjusting block size [9].

In this paper, we overcome the limitations of the existing methods and we formulate a task assignment problem by performance prediction for Cloud applications in a multi-agent environment that is based on an incremental online learning process. Diversely from the approaches proposed in [3] and [4], which are restricted to only some classes of platforms and applications, the novel method is neither application-specific or platform-dependent and is able to work in a generic cloud environment in which, for each agent, a certain number of concurrent different processes are running. The main objective of the proposed approach is to find, according to the client preferences, the agent that completes the execution in least time or, in alternative, the cheapest agent.

Differently from [9], which is focused mostly on the use of Blockchain for the implementation of optimal auction and bidding strategies, we leverage on Hyperledger Fabric to manage both the agent selection process by auction and the task execution. In addition, we adopt a DRL incremental learning approach [10] to enable each agent to predict task runtime and therefore place a bid for a submitted task. The proposed DRL-based algorithm overcomes the limitations of [8] as it does not assume any preliminary consumption-related information in advance.

In more detail, the paper contributions are listed as follows:

1) A Blockchain-based trading platform is designed on top of Hyperledger Fabric to orchestrate clients requests, agents bids and resulting task assignment. We formulate a double bidding strategy according to client preferences. On the one hand, agents provide both a runtime estimation and a price. On the other hand, clients choice between the least price and the least estimated time according to their current setting.
2) A model-free DRL framework for task runtime estimation to support the agent selection process. The proposed algorithm enables the agent to incrementally learn how to do predictions for generic tasks represented by only two parameters, considering its current load in terms of resources consumption and already running tasks.

To the best of our knowledge, it is the first time that Hyperledger Fabric Blockchain and DRL are combined together for task assignment orchestration and performance prediction. In this regard, in our previous works [11] and [12], we propose a similar framework, which is based on Ethereum [13] and implements a traditional offline deep learning-based prediction strategy, rather than the incremental online approach proposed in [13].

We conduct extensive experiments to evaluate the performance of the proposed DRL algorithm with different training factors. In particular, we show how it works by varying discount factor, number of episodes and policy exploration vs. exploitation probability.

The rest of the paper is organized as follows: Section II gives some preliminaries on Blockchain, Hyperledger Fabric and DRL. Section III describes some works about performance prediction, DRL-based task assignment algorithms and resources trading systems in Blockchain environments combined with DRL. Section IV introduces the proposed system model. The DRL-based task assignment approach is

described in section V, whereas in Section VI experimental results are presented in detail. Finally, Section VII concludes the paper.

II. BACKGROUND

A. BLOCKCHAIN

Blockchain is a distributed ledger technology which is used to record transactions without the need of a central authority. Each transaction is signed at least by the issuer and it is verifiable by the nodes participating in the peer-to-peer network [14]. Transactions are also grouped in *blocks* and each block is connected to the previous one by including its Secure Hash Algorithm (SHA) *SHA-256* in the header, thus creating a *tamper-proof* chain [15]. Moreover, all nodes must agree to which transactions are valid and the order in which they have to be stored in the ledger. This is the goal of the so-called *consensus* mechanism and different methods exist to reach such *consensus* in a Blockchain [16].

A *Smart Contract* is a decentralized program code deployed in the Blockchain that enforces the terms and conditions of a specific agreement between two or more parties [17]. It is executed automatically when triggering conditions are satisfied.

B. HYPERLEDGER FABRIC

Hyperledger Fabric is an enterprise-grade Blockchain platform which has great advantages, compared to other platforms such as Ethereum, in terms of transactional privacy, flexibility and data-query capabilities [18]. Since it is mainly designed for permissioned networks, a new node must be authorized to join the network by another node with appropriate permissions. In the same way, a user can be granted access to all of part of the data according to its role. Moreover, in order to ease deployment, it leverages on a series of single micro-services based on Docker [19] containers. The list of services includes *peer*, *Certification Authority (CA)* and *orderer*, each with a specific role. A CouchDB [20] no-sql database is often used to deploy the ledger, while *Smart Contracts* are called *Chaincodes*. Some of the most popular languages are supported to develop chaincodes including Java, NodeJs and GoLang. Finally, on consensus layer, Hyperledger Fabric supports practical leader-based algorithms, such as RAFT [21], in which a recognized leader node publishes the blocks while all other nodes verify and validate transactions.

C. DEEP REINFORCEMENT LEARNING

DRL is a technique that combines traditional RL and deep learning. One of the first approaches of DRL was DQN for Atari games [10], in which a deep neural network (DNN) was used as the function approximator in the place of traditional Q-learning [22]. A more recent application is Deep Deterministic Policy Gradient (DDPG) [23], which is designed for scenarios with exponentially large continuous action space.

III. RELATED WORK

The prediction of performance for a vast class of tasks with machine learning (ML) techniques has been extensively studied over the past years. Some of the earliest works suggest how to predict resource consumption over time, such as CPU utilization, amount of used memory, disk space and network bandwidth, given a set of environment attributes.

In [24], the suitability of several machine learning techniques for predicting spatio-temporal utilization of resources by two bioinformatics applications, BLAST and RAxML, is studied. Some of the investigated methods are: k-nearest neighbor, linear regression, decision table, Radial Basis Function network, Predicting Query Runtime and Support Vector Machine. The authors conclude that different algorithms perform better in different situations and that including as many attributes as available, even from monitoring systems, can improve prediction performance.

In [25], a novel approach for learning ensemble prediction models of tasks runtime is presented. The authors use bagging techniques to produce ensembles of regression trees to be used in scientific workflows, such as gene expression analysis, made of different sub-tasks, and show that their method leads to significant prediction-error reductions when compared with standalone models.

Still referring to scientific workflows, an online incremental learning approach to predict the runtime of tasks in cloud environments is suggested in [26]. The incremental approach enables to capture some typical cloud features, such as the continuous environmental changes and the heterogeneity of the different platforms whereas. Recurrent Neural Network (RNN) and K-Nearest Neighbors (KNN) are used for estimation. To improve predictions, fine-grained time-series resources monitoring data related to CPU utilization, memory usage, and I/O activities are collected for each unique task in the form of time-series records. The authors show that their approach significantly outperforms state-of-the-art solutions in terms of estimation error.

A similar approach, called two-stage prediction, is proposed in [27]. In this work, first the resources consumption in terms of CPU utilization, memory, storage, bandwidth, I/O is estimated for a single task instance in a cloud environment and then the result is used for runtime prediction. However, that is an offline machine learning approach and has some limitations compared to online approach in which data are processed as soon as they arrive in a near real-time fashion and it does not reflect the streaming nature of workloads in cloud environments.

Furthermore, in [2] a systematic literature review of performance prediction methods for parallel applications is presented that includes analytic and non-analytic methods and indicates future research trends and some unsolved issues. In particular, this work shows that performance prediction has been applied on a wide range of domains and reviews 82 different approaches developed between 2005 and 2020. However, the authors conclude that most of these methods

focus on a specific application type and platform and therefore there's a lack of independent solutions able to work in unknown environments.

Resource management and task assignment are two other problems that have received great attention in the related literature. In this context, the DRL approach, that is often used for intelligent-robot related problems such as optimal path planning and obstacle avoidance [28], has recently become very popular. For instance, in [8], a multi-resource cluster scheduler named *DeepRM* is presented that is able to learn how to manage resources directly from experience in an online fashion and to optimize various objectives such as minimizing average job slowdown or completion time. The authors show that their method performs comparably to state-of-the-art heuristics, adapts to different conditions and converges relatively quickly.

In [7], a DRL approach for solving task mapping problems with dynamic traffic on parallel systems is discussed. The algorithm explores better task mappings by using a network simulator that predicts performance and runtime communication behaviors. Since communication pattern are often changing and unknown, network performance is difficult to be accurately estimated, therefore the authors claim that DRL is an efficient solution in this dynamic context and show that their method performs comparably or better than previous approaches.

Diversely from the aforementioned works, the DRL approach proposed in this paper does not rely on a specific class of application, but it is intended to be an abstract framework combined with modern container based technologies, such as Docker, with the purpose to learn the behavior and estimate the execution time of a generic software task that can be packed in a container image and can be monitored by Docker resources metrics in a competitive environment.

Blockchain-enabled solutions for a vast range of problems have been recently arising. For example, in our previous works [11] and [12], a Smart Contracts-based platform is proposed for improving digital processes in a Cloud manufacturing environment. In detail, we combine Blockchain with Docker and Cloud Storage and introduce a deep learning approach in a task mapping framework. In [29], a blockchain-based two-stage secure spectrum intelligent sensing and sharing auction mechanism for mobile devices is designed in a consortium blockchain to guarantee secure and efficient spectrum auction with low complexity. In [30], the authors introduce a novel scalable and multi-layer blockchain-based energy trading framework for cooperative microgrid systems that considers the issue of of perceiving the status of block generation over temporary network disruption and improves the consensus and the reliability of energy trading.

Moreover, several recent works have been proposed addressing security, data integrity and optimization problems with a combination of Blockchain and DRL. For instance, in [31], a reliable data collection and a secure sharing scheme for smart mobile terminals are proposed. The authors introduce an Ethereum-based Blockchain to safely manage data sharing with a DRL approach to achieve the maximum amount of collected data, geographic fairness and minimize energy consumption. Several simulations experiments show that their method outperforms traditional database based approaches in terms of reliability and security.

In [32], a smart grid blockchain combined with fog computing is suggested. This work includes a Hyperledger Fabric Blockchain in which the nodes are part of a fog computing environment. A verifiable random function is proposed to ensure randomness and increase safety in the selection of the primary node while keeping the probability proportional to the computing power provided by each member. Based on storage cost and security constraints, a DRL scheme is implemented to adjust the block size and the block interval in the proposed Blockchain. By conducting extensive simulations, the authors show the superiority of their scheme in terms of throughput and latency. A similar approach, though intended to decrease energy consumption and to improve the efficiency of the consensus process in Blockchain-enabled Industrial Internet of Things systems by adjusting block size and offloading some tasks to computing servers, is suggested in [33]. The problem is formulated as a high-dynamic and high-dimensional Markov Decision Process for whom a DRL approach is used to converge to an optimal solution.

A peer-to-peer energy trading problem among microgrids is investigated in [34]. In this work a multi-agent deep deterministic policy gradient, based on energy trading algorithm, is proposed to enable each microgrid to maximize its own utility in a local market. Given the uncertainties and the constraints in renewable energy and power demand, the authors claim that the DRL-based approach is suitable to help each microgrid to find its optimal policy. An Ethereum Blockchain is adopted to ensure the integrity of transaction data.

A Blockchain-enabled computing resource trading system is proposed in [9]. This system takes into account pricing and bidding strategies to enable providers and customers to trade computing resources on a safe and tamper-proof environment. A decision-making problem in the continuous double auction is formulated with the goal of maximizing each participant payoffs, while a DRL approach is adopted to help them building their optimal bidding strategies. The authors conduct extensive simulations and show that their scheme outperforms other existing methods.

Finally, in [35], a DRL approach is used to solve a joint optimization problem to enhance adaptivity and scalability in Blockchain environments. The proposed approach considers the optimal selection of consensus protocols and the allocation of computation and bandwidth resources. The authors show through extensive simulations the effectiveness of their scheme.

IV. SYSTEM MODEL

In this section, we introduce a new system that combines a novel runtime estimation algorithm based on DRL in a competitive task assignment framework safely managed by

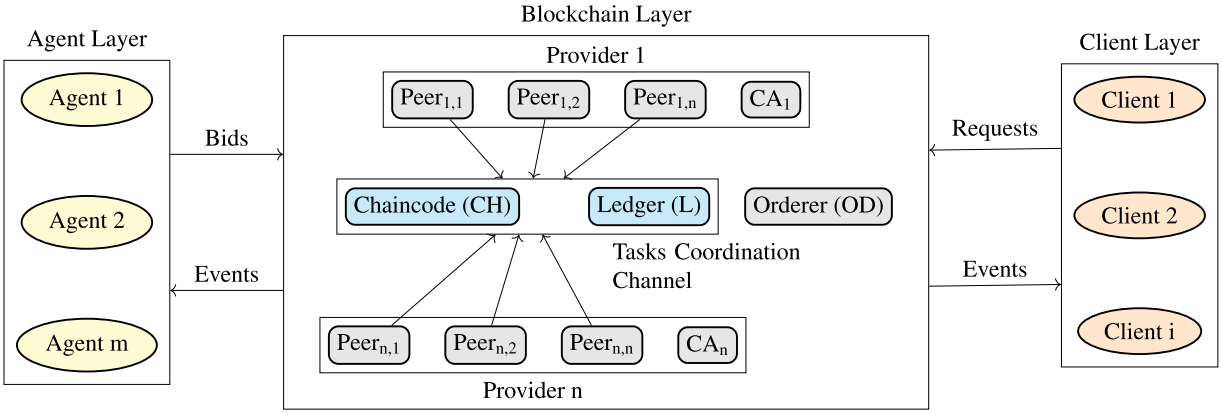


FIGURE 1. System architecture.

a Hyperledger-based Enterprise Blockchain. The proposed architecture is depicted in Fig. 1. The system is made up of three layers: the Blockchain layer in the middle, and the Agent and Client layers on the sides.

A. BLOCKCHAIN LAYER

Let us consider a set of pre-authorized organizations $O = \{o_1, o_2, o_n, \ldots, o_{|O|}\}$, called providers, that constitute the nodes of the permissioned Hyperledger Blockchain. An organization in Hyperledger Fabric is simply a firm that decides to join a network and that is authorized by other existing members. Note that symbol $|A|$ denotes the cardinality of the generic set A. Moreover, each provider o_n delivers a set of peers $P_n = \{p_{n,1}, \ldots, p_{n,i}, \ldots, p_{n,|P_n|}\}$ joining the network, where $p_{n,i}$ is the *i-th* peer of provider o_n. Each provider o_n has one Certification Authority CA_n to provide and renew certificates for Agents, Clients and other components. All peers in the set P_n hold a copy of the chaincode CH and of the ledger L. The chaincode CH is the main Smart Contract that regulates the requests of the clients, the bids received by the agents and coordinates the task assignment process. The ledger L stores the current state of the tasks and the related transactions. Finally, the Orderer node OD is responsible for packaging transactions into blocks and distributing them to the peers in the sets P_n across the network.

B. CLIENT LAYER

In the proposed system, we define the set of clients $C = \{c_1, \ldots, c_i, \ldots, c_{|C|}\}$, where c_i implements a step of a generic process flow, for which it is required to run computationally intensive tasks. Since c_i may not have enough resources to run tasks locally, it interacts with the Blockchain to submit execution requests. For each successful task execution, a variable fee is charged.

For each c_i, we define the set

$$R_i = \{r_{i,1}, \ldots, r_{i,j}, \ldots, r_{i,|R_i|}\}$$

where $r_{i,j}$ represents the *j-th* execution request of client c_i. In addition, for each request $r_{i,j}$, the client c_i has to specify the strategy and the task details. Considering the used strategy, we define two alternative options: *time-sensitive* and *price-sensitive*. In the first case, we assume that the client objective is to complete the task in the least time possible, regardless of the charged fee. In the second case, the requirement is to pay the lowest fee, regardless of the execution time. In order to specify task details, we define the set $T = \{t_1, \ldots, t_k, \ldots, t_{|T|}\}$, where t_k represents a generic task that can be requested by the clients with an arbitrary input parameter.

Now, each request $r_{i,j}$ can be defined by a triple $r_{i,j} = \langle g_{i,j}, t_{i,j}, p_{i,j} \rangle$, where $g_{i,j} \in [0, 1]$ is the strategy, with 0 representing *time-sensitive* and 1 *price-sensitive*, $t_{i,j} = t_k \in T$ is the required task and $p_{i,j}$ is the input argument.

C. AGENT LAYER

Each request $r_{i,j}$ submitted by a client requires an Agent to execute the associated task. In this fully distributed environment, we define the set of agents $A = \{a_1, \ldots, a_m, \ldots, a_{|A|}\}$, where a_m is a node that joins the network and provides its computational resources, such as CPU and memory, to the system. For the sake of generalization, we use Docker container-based technology to embed the tasks in self-dependent images. In this way, since everything is packed in the Docker image, the nodes do not have any constraint in terms of operating system and dependencies.

Now, at each request $r_{i,j}$, we associate an agent $a_m \in A$ that is denoted $a_{i,j}$. Such an agent is paid a variable fee upon successful execution of the related task. Moreover, since for each agent a_m we allow the execution of concurrent processes, the execution time strictly depends on its current load state.

As illustrated in section IV-B, the client can choose between *time-sensitive* and *price-sensitive* strategies. More specifically, when the former strategy is chosen, the client goal is to minimize the run-time of its processes and thus each agent a_m is required to provide a reliable prediction of the execution time for each request $r_{n,i}$ based on its current state. Conversely, in case the latter strategy is selected, only the proposed price is relevant for the assignment. The main

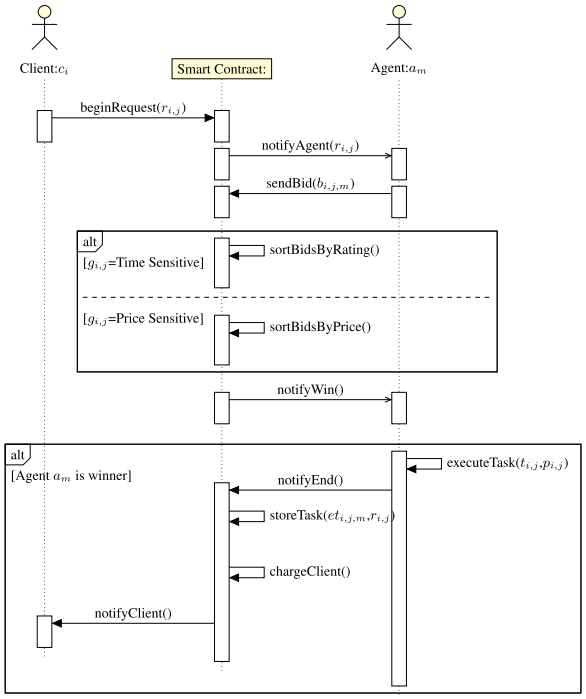


FIGURE 2. Agent selection UML sequence diagram.

scope of the *price-sensitive* option is to allow an agent that has recently joined the network and it is not able to provide a reliable execution time prediction to compete in price rather than in time and collect new experiences to progressively improve its predictions.

V. DRL-BASED TASK ASSIGNMENT PROCESS

In this section, we introduce a competitive task assignment process by expected runtime prediction that implements the two client strategies: *time-sensitive* and *price-sensitive*. The agents leverage on a novel DRL approach to provide predictions.

A. TASK ASSIGNMENT PROCESS

The main process is designed in the chaincode *CH* and is depicted in the UML sequence diagram in Fig. 2. Both the clients C and the agents A continuously interact with the Blockchain for requests and tasks management. More specifically, in the first part of the diagram, the agent selection is performed and in the second part, post-assignment actions are implemented.

The agent selection process is detailed in Algorithm 1, which is implemented in the chaincode in *GO* [36] programming language. The input is the client request $r_{i,j}$ and the output is the winner agent $a_{i,j} \in A$.

In Step 1, a request $r_{i,j}$ for a $b_{i,j,m}$ is sent to each available agent $a_m \in A$, where $b_{i,j,m}$ is the bid of agent m for the request j of client c_i. In addition, $b_{i,j,m}$ is defined as $b_{i,j,m} = \langle et_{i,j,m}, pr_{i,j,m} \rangle$ where $et_{i,j,m}$ is the estimated completion time provided by the agent a_m and $pr_{i,j,m}$ is the proposed price. All received bids are collected in the set $B = \{b_{i,j,1}, \ldots, b_{i,j,m}, \ldots, b_{i,j,|A|}\}$. In Step 2, given the strategy $g_{i,j}$ in $r_{i,j}$, the members of B are sorted by $et_{i,j,m}$ or $pr_{i,j,m}$ accordingly. Finally, in Step 3, the first available agent $\overline{a_m}$ associated to $b_{i,j,m}$ in the sorted rank is elected as the winner and therefore $a_{i,j} = \overline{a_m}$.

Algorithm 1 Agent Selection Algorithm

Input: A Client Request $r_{i,j} = \langle g_{i,j}, t_{i,j}, p_{i,j} \rangle$
Output: Winner Agent $a_{i,j} \in A$
Step 1:
  foreach $a_m \in A$ **do**
    Forward request $r_{i,j}$ to Agent a_m;
    Collect bid $b_{i,j,m} = \langle et_{i,j,m}, pr_{i,j,m} \rangle$;
    $B \leftarrow B \cup \{b_{i,j,m}\}$;
  end
Step 2:
  if $g_{i,j}$ *is time-sensitive* **then**
    Sort set B by estimated time $et_{i,j,m}$;
  else if s *is price-sensitive* **then**
    Sort set B by bid price $pr_{i,j,m}$;
Step 3:
  Set $h \leftarrow 1$;
  while $B \neq \varnothing$ **do**
    Take bid $b_{i,j,m}(h) \in B$;
    if *Agent* $\overline{a_m}$ *is available* **then**
      Agent $\overline{a_m}$ is the winner;
      $a_{i,j} \leftarrow \overline{a_m}$;
      return $a_{i,j}$;
    else
      $B \leftarrow B \setminus \{b_{i,j,m}\}$;
      Set $h = h + 1$;
  end

The estimated time $et_{i,j,m}$ leverages on the DRL approach that is described in Section V-B. On the contrary, the bid price $pr_{i,j,m}$ is determined autonomously by the agent to compete in the current race.

After a request $r_{i,j}$ has been successfully assigned, the winner agent $a_{i,j}$ executes the task $t_{i,j}$ with the parameter $p_{i,j}$ and notifies the completion to the chaincode *CH*. Finally, the instance metrics, such as the effective runtime, are stored in the tamper-proof Blockchain and the client c_i is charged according to the offset between the actual execution time and the prediction.

B. DRL APPROACH FOR RUNNING-TIME PREDICTIONS

In this section, we propose a DRL approach for task runtime prediction. In particular, the Deep-Q-Network (DQN) algorithm proposed in [10] is applied to the considered problem. The triple (S, A, R) defines a RL deterministic model, where S is the State Space, A is the Action Space and R is the Reward. In the considered application the triple (S, A, R) is specified in the following.

1) **State Space (S):** We assume that each agent is able to run a fixed number k of concurrent tasks as individual containers. As a consequence, execution time is potentially impacted by three combined parameters to be considered on a new incoming request $r_{i,j}$:

   I) the type of task $t_{i,j} \in T$, its associated input parameter $p_{i,j}$ and submission time $ts_{i,j}$;
   II) total available resources for each agent a_m: cpu_speed_m, number of cores n_cores_m, maximum memory max_memory_m, hdd_type_m (rotational or high-speed SSD), available network bandwidth bw_m;
   III) current resources consumption of previous $k-1$ processes running on a_m: $cpu_time_{m,n}$, $mem_{m,n}$, $net_i/o_{m,n}$, $block_i/o_{m,n}$, number of processes $n_pids_{m,n}$ where $n = 1, 2, \ldots, k-1$.

   Now, we introduce the following three tuples:

   I) $s_{m,1} = \langle t_{i,j}, p_{i,j}, ts_{i,j} \rangle$;
   II) $s_{m,2} = \langle cpu_speed_m, n_cores_m, max_memory_m, hdd_type_m \rangle$;
   III) $s_{m,3} = \langle cpu_time_{m,n}, mem_{m,n}, net_i/o_{m,n}, block_i/o_{m,n}, n_pids_{m,n} \rangle$, where $n = 1, 2, \ldots, k-1$.

   Moreover, we define $S_m = \{s_{m,1}, s_{m,2}, s_{m,3}\}$ that represents the current state of agent a_m in the proposed DQN framework. The structure of $s_{m,2}$ reflects the current node configuration. Those features are assumed to be static and are described in Table 1. In addition, $s_{m,3}$ denotes the current running containers resources usage and is continuously updated. Since in the proposed system, each task is represented by a Docker image and, consequently, each instance is essentially a container based on that image, we can leverage on live metrics exposed by Docker to collect live resources usage data. In Table 2, all those metrics are described.
2) **Action Space (A):** Given the current state of the candidate agent a_m, the target is to predict the expected runtime in seconds for the incoming client request $r_{i,j}$. In the proposed DRL approach, the bid estimated value $et_{i,j,m}$ constitutes the action. In order to ease the training, we consider a discrete set of execution times in steps of 1 second with a predefined upper bound value that constitutes a global time out for each instance processed by the system.
3) **Reward (R):** Once task $t_{i,j}$ has been executed, each state S_m and action $et_{i,j,m}$ produce the actual elapsed time as observation and contribute to the reward by assessing the level of prediction accuracy. Thus, if we set a constant token TK_{max} as a maximum reward for each successful calculation, we can calibrate the reward value $RD_{i,j,m}$ by considering the relative error between actual execution time $at_{i,j,m}$ and estimated time $et_{i,j,m}$.

TABLE 1. Node configuration features.

Feature	Description	UM
cpu_{speed}	CPU speed	Mhz
n_{cores}	Number of cores	n/a
max_{memory}	Memory	MB
hdd_{type}	Storage type	(0=spinning,1=solid state)
bw	Network bandwidth	Kbps

TABLE 2. Resources usage metrics.

Metrics	Description	UM
cpu_{time}	The amount of cpu usage accumulated by the process	1/100th seconds
mem	The total memory the container is using	MiB
$net_{i/o}$	The amount of data the container has sent and received over its network interface	bytes
$block_{i/o}$	The amount of data the container has read to and written from block devices on the host	bytes
n_{pids}	The number of processes or threads the container has created	n/a

In detail, $RD_{i,j,m}$ is defined by the following formula:

$$RD_{i,j,m} = TK_{max} \begin{cases} \dfrac{at_{i,j,m}}{et_{i,j,m}} & at_{i,j,m} \leq et_{i,j,m} \\ \dfrac{et_{i,j,m}}{at_{i,j,m}} & otherwise. \end{cases} \quad (1)$$

The adopted DQN scheme is described by Algorithm 2. Firstly, according to the approach in [10], a number of episodes E for training and the condition for a state S_m to be terminal are defined. In the proposed approach, the number of episodes is arbitrary and is strictly related to the number and types of tasks in the set T. Basically, it should be as long as to guarantee an accurate prediction of execution runtime $et_{i,j,m}$ in every agent load condition. Similarly, a maximum number of iterations I is set after which the state S_m is terminal and single episode is completed.

Step 1 initializes the replay memory set D with capacity N and the Q network that will be trained.

Moreover, in Step 2, for each iteration of each episode, whenever a new request $r_{i,j}$ is submitted to agent a_m, a new state S_m is built and a value for $et_{i,j,m}$ is randomly selected or predicted with probability ϵ. Subsequently, price $pr_{i,j,m}$ is arbitrarily determined by agent a_m and the bid $b_{i,j,m}$ is definitely set. As shown in Fig. 2, $b_{i,j,m}$ is sent to the Smart Contract on which the Algorithm 1 is executed after all bids from all agents are collected.

In Step 3, if the agent a_m is the winner, $t_{i,j}$ is run locally and the actual execution time $at_{i,j,m}$ is observed from Docker metrics to calculate the step reward $RD_{i,j,m}$. At the same time, a new training step begins.

In Step 4, on next incoming request $r'_{i,j}$, the future state S'_m is determined. Then, the whole transition tr_h is stored in experience replay memory set D. Subsequently, M transitions are randomly selected from set D in the subset D_h. Then, a new value y_n that considers maximum future reward as

Algorithm 2 Deep Q Learning

Step 1:
Initialize replay memory set D with capacity N
Initialize $Q(S_m, \widehat{et})$ arbitrarily
Step 2:
Set $e = 1$
while $e \leq E$ **do**
Set $\hat{i} = 1$
while $\hat{i} \leq I$ **do**
Wait for request $r_{i,j}$
With probability ϵ select a random action $\widehat{et}$
otherwise select $\widehat{et} = \max_{\widehat{et}} Q(s, \widehat{et}; \theta)$
Set $et_{i,j,m} = \widehat{et}$
Set arbitrary price $pr_{i,j,m}$
Set $b_{i,j,m} = \langle et_{i,j,m}, pr_{i,j,m} \rangle$
Send $b_{i,j,m}$ to Smart Contract and wait for Algorithm 1
Step 3:
if *agent* a_m *is the winner* **then**
Take action $et_{i,j,m} = \widehat{et}$, observe $RD_{i,j,m}, S'_m$
Step 4:
Wait for next request $r'_{i,j}$
Store transition $tr_h = \langle S_m, et_{i,j,m}, RD_{i,j,m}, S'_m \rangle$ in D
Sample random minibatch of M transitions $D_h = \{tb_1, tb_2, tb_n, \ldots, tb_M\}$ where $tb_n \in D$
foreach $tb_n \in D_h$ **do**
Set $y_n \leftarrow$
$RD_{i,j,m,n}$ for terminal $S'_{m,n}$
$RD_{i,j,m,n} + \gamma \max_{et_{i,j,m}} Q(S'_{m,n}, et'_{i,j,m}; \theta)$ for non-terminal $S'_{m,n}$
end
Step 5:
Perform gradient descent step on $(y_n - Q(S_{m,n}, et_{i,j,m,n}; \theta))^2$ where $n = 1, 2, \ldots, M$
$S_m \leftarrow S'_m$
Set $\hat{i} = \hat{i} + 1$
end
Set $e = e + 1$
end
end

predicted from target network Q and discounted by γ is calculated as updated reward for each transition $tb_n \in D_h$.

Finally, in Step 5, a new Q network is trained by performing a gradient descent step on $(y_n - Q(S_{m,n}, et_{i,j,m,n}; \Theta))^2$, in order to get new θ for next iteration $h + 1$. The target network is updated every w iterations to stabilize learning, where w must be preliminary assigned.

TABLE 3. Tasks details.

ID	Algorithm	Complexity	Min	Max
1	Sort	$\mathcal{O}(n \log n)$	2×10^4	5×10^4
2	NumPy sort	$\mathcal{O}(n^2)$	10^6	2×10^6
3	Dijkstra SPS	$\mathcal{O}((\lvert V\rvert + \lvert E\rvert) \log \lvert V\rvert)$	10^3	5×10^3

VI. PERFORMANCE EVALUATION

In this section, we evaluate the performance of the proposed task runtime prediction DQN driven algorithm in a simulated environment.

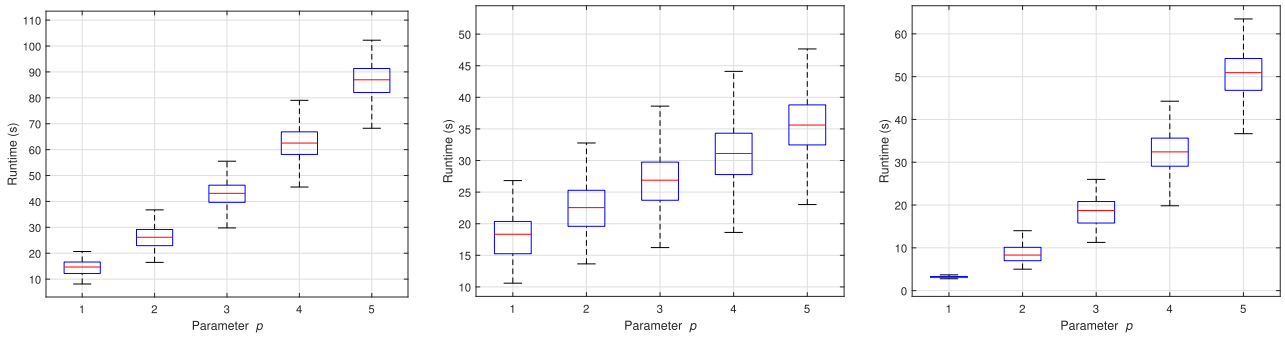
A. SIMULATION SETTINGS

In the proposed case study, we identify three common software algorithms with different complexities to evaluate the proposed system, therefore we set $T = \{t_1, t_2, t_3\}$. We code the tasks using *Python* language in a single Docker image that requires two parameters on launch: task $t_{i,j} \in T$ and parameter $p_{i,j}$. In the considered case, the value of $p_{i,j}$ is restricted to the members of the set $P = \{1, \ldots, 5\}$ with $p_{i,j} \in P$.

In particular, we consider the following three well known algorithms:

1) **Standard Array Sorting** (t_1): it builds a random big *Python* integer list whose number of elements is based on $p_{i,j}$. After building the list, the *sort* method is called that implements the *Timsort* algorithm [37]. This algorithm has a runtime complexity of $\mathcal{O}(n \log n)$ in the worst case;
2) **Fast Array Sorting** (t_2): instead of constructing a list, a random *NumPy* [38] integer array is built that also implements the *sort* method. However, in *NumPy* library the *quicksort* algorithm [39] is adopted which has a runtime complexity of $\mathcal{O}(n^2)$ in the worst case;
3) **Dijkstra Shortest Path Search** (t_3): The *Dijkstra*'s algorithm [40] is an algorithm for finding the shortest paths between nodes in a graph. In this implementation, firstly we build a graph with a large number of vertices V determined by parameter $p_{i,j}$. Secondly, we find the shortest path from the first vertex to all other vertices. The *Dijkstra* standard implementation has a complexity of $\mathcal{O}((|V| + |E|) \log |V|)$ in the worst case.

The proposed algorithms are currently implemented in scientific and industrial applications. For example, sorting tasks are used in operations research to implement both the *Shortest Processing Time First* and the *Longest Processing Time First* rules for optimal jobs scheduling and load balancing [41]. Indeed, *Dijkstra*'s algorithm is currently been used for a vast class of problems including vehicle path



(a) Sort t_1 (b) NumPy sort t_2 (c) Dijkstra SPS t_3

FIGURE 3. (a), (b), (c), tasks average runtime for different values of parameter $p_{i,j}$.

planning [42] and optimal packet routing in software defined network environments [43].

In Table 3, we summarize the three tasks, their runtime complexities and the minimum and maximum value of elements as determined by their linear combination with parameter $p_{i,j}$.

We set maximum concurrent tasks $k = 4$ and we run a random sequence of simultaneous instances on a designated agent. On each new submitted instance, we collect current Docker metrics from the node as described in section V-B.

As agent, we use a *t2.medium* cloud instance from *Amazon AWS* equipped with 2x Intel Xeon 3.3Ghz vCPU and 4GB of memory. Since the number of available CPUs is less than k, we simulate an elevated competition between tasks during runtime and therefore we observe high variability in execution time, which is typical of real cloud environments.

In Fig. 3, the average task $t_{i,j}$ runtime, observed for each $p_{i,j}$ value over a total of 9995 metrics collected in two days simulation, is depicted.

We use the metrics set for testing the performance of the DRL approach described in Section V. In particular, on each iteration a new state from current item is built, then either a random time is selected or the expected runtime is predicted from current Q-function according to the exploration vs. exploitation probability. Finally, the reward is calculated against the actual runtime as described in 1. The current transition is stored in experience replay, then the rest of the steps in Algorithm 2 update the current Q-function.

In this experiment, the Q-function is approximated by a dense DNN made up of three layers:

1) The input layer represents the state and consists of 21 neurons. In this layer, we combine some elements of the state to reduce the dimension and we use One Hot Encoding techniques [44] to represent some categorical variables, such as the tasks IDs.
2) The intermediate layer consists of 70 neurons, as the average between input and output dimensions
3) The output layer represents the discretized action space and consists of 120 neurons, where 1 second is the minimum expected runtime and 120 seconds is the last accepted value before raising a timeout error.

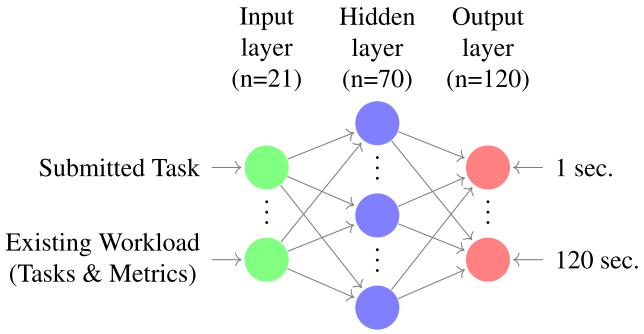


FIGURE 4. Deep neural network for Q-function.

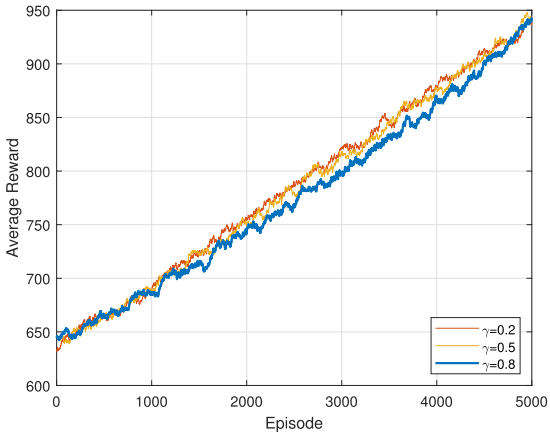
The deep neural network approximating the Q-function is depicted in Fig. 4.

To evaluate the performance of our DRL strategy, we use a *t3.2xlarge* cloud instance from *Amazon AWS* equipped with 8 x Intel Xeon 3.1Ghz vCPU and 32GB of memory.

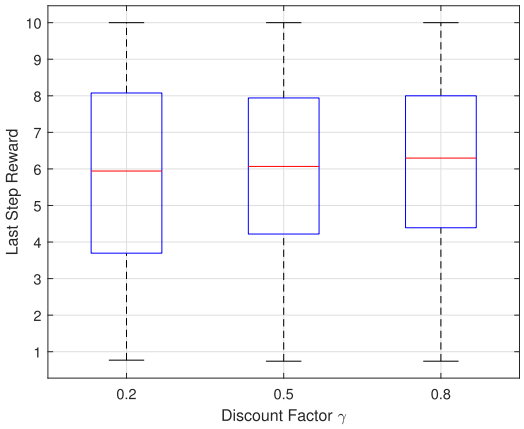
B. PERFORMANCE COMPARISON

This section reports a performance comparison between two different scenarios tested in the performed experiment. The first scenario consists of 5000 episodes of 150 iterations each, whereas the second scenario consists of 3000 episodes of 300 iterations each. Since we are using a DRL approach, we have two more critical hyper-parameters to consider:

1) *Discount Factor* γ: in DRL environments, setting the discount factor is part of the problem [45]. The discount factor essentially determines how much the RL agent cares about rewards in the distant future relative to those in the immediate future. If $\gamma = 0$, the agent will be completely myopic and only learn about actions that produce an immediate reward. If $\gamma = 1$, the agent will evaluate each of its actions based on the sum total of all of its future rewards. In the simulations, we continuously feed the system with tasks and we are interested in evaluating how the DRL approach performs with different values of γ. Therefore, we test the algorithm for both scenarios with $\gamma = 0.2$, $\gamma = 0.5$ and $\gamma = 0.8$.
2) *Exploration vs. Exploitation Probability* ϵ: in RL, exploration means that the agent explores randomly

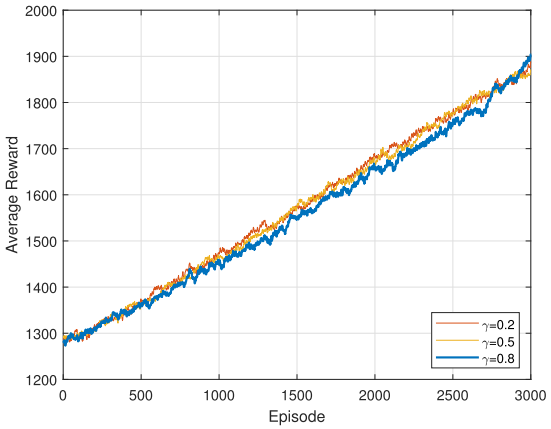


(a) Average Reward

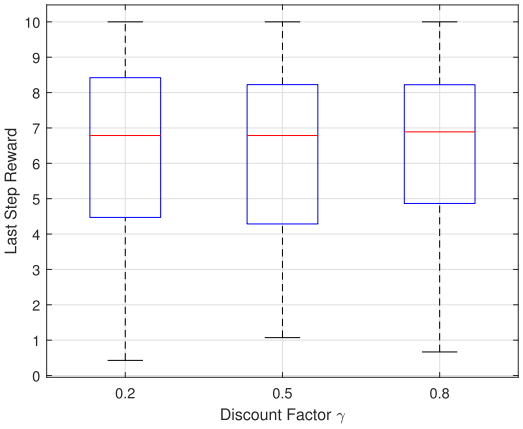


(b) Prediction Accuracy

FIGURE 5. First scenario performance evaluated for different values of γ: (a) Average reward over episodes, (b) Runtime prediction accuracy summary statistics.



(a) Average Reward



(b) Prediction Accuracy

FIGURE 6. Second scenario performance evaluated for different values of γ: (a) Average reward over episodes, (b) Runtime prediction accuracy summary statistics.

the whole action space to improve its knowledge about each action for a long-term benefit. On the other hand, exploitation means that the agent uses only its current knowledge to get the most reward. In Algorithm 2, the choice between exploration and exploitation is made with a probability coefficient ϵ that usually varies at each episode. Normally, this value starts from 1, meaning that, since the agent doesn't know anything at the beginning about the actions, it must explore all available actions for each state. Successively, it slowly decays over future episodes until the training end, when it is very close to 0, meaning that it fully levarages on its knowledge. For the performed evaluations, we start from $\epsilon = 1$ and we implement a linear decaying strategy over episodes till a minimum value of $\epsilon = 0.001$.

Considering the aforementioned hyper-parameters, we are interested in comparing the following two metrics for both scenarios.

1) **Cumulative Reward** *CR* is defined for single episode of n steps as:

$$CR = \sum_{n} R(n); \;\; CR \in [0, C_{max}n]. \tag{2}$$

Figures 5a and 6a compare the average values *CR* vs. episodes for all three considered discount factors in both scenarios. For all values of γ, as the agent explores new actions and trains its Q-function, the *CR* value increases almost linearly and results in 46% performance improvement at the end of the training, compared to the first exploration-only episode. Moreover, though $\gamma = 0.2$ appears to learn faster than $\gamma = 0.5$ and $\gamma = 0.8$, in the end the performance of the highest discount factors outperforms the smallest one. In the last part of the training, the value of ϵ becomes very small and let the algorithm leverage only on exploitation for task runtime prediction. In this case, the comparison shows that for higher values of γ, the accuracy of the Q-function slightly improves proving that the proposed DRL approach is able to catch a sort of correlation between successive submitted tasks. There are no major differences between the performance of the two scenarios. It can only be observed that in the second scenario, the learning speed for $\gamma = 0.8$ is higher than in the first scenario, compared to other values of γ. This metric can be influenced by the different number of steps in the episode and the resulting value

of exploration vs. exploitation probability coefficient ϵ in each step. In fact, in the first scenario, ϵ decreases linearly every 150 steps, whereas in the second scenario every 300 steps. As a consequence, although the evaluated performance shows that a higher number of steps per episode improves the algorithm, it is evident that episode length also affects the training time. However, the simulations point out that different values of γ and ϵ may help in minimizing total computational effort.

2) **Runtime Prediction Accuracy** is measured in the last training step, in which the algorithm only exploits the approximated Q-function. We set $C_{max} = 10$ as highest step reward in case of exact runtime prediction, otherwise the value is determined in a linear fashion as defined in (1). Figures 5b and 6b show summary step reward statistics for both scenarios and for different values of discount factor γ. As for the previous metric, in all the cases a similar performance is observed among the compared schemes, having $\gamma = 0.8$ case the best median $\tilde{x} = 6.30$ and $\tilde{x} = 6.89$ for the first and second scenario, respectively. The latter case with $\gamma = 0.8$ shows the overall best performance, having the 50% of the steps a prediction accuracy $\geq \approx 69\%$, whereas only a residual 25% of the steps show an accuracy $\leq \approx 49\%$. Finally, the upper quartile shows a prediction accuracy $\geq \approx 82\%$ for 25% of the steps.

Given the dimensions of state space and action space, the evaluated performance is acceptable and confirms the effectiveness of the proposed DRL approach.

VII. CONCLUSION

In this paper, we introduce a Hyperledger Fabric Blockchain-based resources trading platform to orchestrate clients requests, agents bids and provide a solution to a common task assignment problem. A double bidding strategy is proposed that enables client to choose either the least time among estimations provided by the agents or the cheapest price. To support node selection and task assignment, we propose a model-free DRL framework for task runtime estimation in agent current load state. In the proposed algorithm, both the submitted and the existing tasks are represented by only two parameters combined with some resources usage metrics collected at runtime. As the agent receives new tasks requests, it incrementally learns how to do predictions in an online fashion.

Simulations show that DRL is suitable for task runtime prediction and provides similar or better performance, compared to other more complex existing solutions. Moreover, it is important to enlighten that the proposed approach is not tied to a specific application and can be adopted on any type of tasks in a modern cloud environment.

In future work, we plan to integrate this solution on *Kubernetes*, an open-source container-orchestration system, and evaluate the performance of the DRL approach for workload operations such as deployment and scaling.

REFERENCES

[1] *Public Cloud IaaS Market Worldwide 2023*, Gartner, USA, Sep. 2022. [Online]. Available: https://www.statista.com/statistics/505251/worldwide-infrastructure-as-a-service-revenue/
[2] J. Flores-Contreras, H. A. Duran-Limon, A. Chavoya, and S. H. Almanza-Ruiz, "Performance prediction of parallel applications: A systematic literature review," *J. Supercomput.*, vol. 77, no. 4, pp. 4014–4055, Apr. 2021.
[3] A. Mohammed, A. Eleliemy, F. M. Ciorba, F. Kasielke, and I. Banicescu, "An approach for realistically simulating the performance of scientific applications on high performance computing systems," *Future Gener. Comput. Syst.*, vol. 111, pp. 617–633, Oct. 2020.
[4] Y. Cho, S. Oh, and B. Egger, "Performance modeling of parallel loops on multi-socket platforms using queueing systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 31, no. 2, pp. 318–331, Feb. 2020.
[5] P. Altenbernd, J. Gustafsson, B. Lisper, and F. Stappert, "Early execution time-estimation through automatically generated timing models," *Real-Time Syst.*, vol. 52, no. 6, pp. 731–760, Nov. 2016.
[6] T. Doan and J. Kalita, "Predicting run time of classification algorithms using meta-learning," *Int. J. Mach. Learn. Cybern.*, vol. 8, no. 6, pp. 1929–1943, Dec. 2017.
[7] Y.-C. Wang, J. Chou, and I.-H. Chung, "A deep reinforcement learning method for solving task mapping problems with dynamic traffic on parallel systems," in *Proc. Int. Conf. High Perform. Comput. Asia–Pacific Region*, Jan. 2021, pp. 1–10.
[8] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. 15th ACM Workshop Hot Topics Netw.*, Nov. 2016, pp. 50–56.
[9] Z. Xie, R. Wu, M. Hu, and H. Tian, "Blockchain-enabled computing resource trading: A deep reinforcement learning approach," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, May 2020, pp. 1–8.
[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing Atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, pp. 1–9, Dec. 2013.
[11] G. Volpe, A. M. Mangini, and M. P. Fanti, "An architecture combining blockchain, Docker and cloud storage for improving digital processes in cloud manufacturing," *IEEE Access*, vol. 10, pp. 79141–79151, 2022.
[12] G. Volpe, A. M. Mangini, and M. P. Fanti, "An architecture for digital processes in manufacturing with blockchain, Docker and cloud storage," in *Proc. IEEE 17th Int. Conf. Autom. Sci. Eng. (CASE)*, Aug. 2021, pp. 39–44.
[13] (Mar. 2023). *Intro to Ethereum*. Accessed: Mar. 29, 2023. [Online]. Available: https://ethereum.org
[14] F. Casino, T. K. Dasaklis, and C. Patsakis, "A systematic literature review of blockchain-based applications: Current status, classification and open issues," *Telematics Informat.*, vol. 36, pp. 55–81, Mar. 2019.
[15] K. Christidis and M. Devetsikiotis, "Blockchains and smart contracts for the Internet of Things," *IEEE Access*, vol. 4, pp. 2292–2303, 2016.
[16] G.-T. Nguyen and K. Kim, "A survey about consensus algorithms used in blockchain," *J. Inf. Process. Syst.*, vol. 14, no. 1, pp. 101–128, Feb. 2018.
[17] R. Koulu, "Blockchains and online dispute resolution: Smart contracts as an alternative to enforcement," *SCRIPTed*, vol. 13, no. 1, pp. 40–69, May 2016.
[18] T. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," *J. Netw. Comput. Appl.*, vol. 177, Mar. 2021, Art. no. 102857.
[19] C. Anderson, "Docker [software engineering]," *IEEE Softw.*, vol. 32, no. 3, pp. 102, May 2015.
[20] (Aug. 2019). *CouchDB: The Definitive Guide*. Accessed: Dec. 22, 2021. [Online]. Available: http://guide.couchdb.org/editions/1/en/index.html
[21] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Tech. Conf. (USENIX ATC)*. Philadelphia, PA, USA: USENIX Association, Jun. 2014, pp. 305–319.
[22] C. J. C. H. Watkins and P. Dayan, "*Q*-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2019, *arXiv:1509.02971*.
[24] A. Matsunaga and J. A. B. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *Proc. 10th IEEE/ACM Int. Conf. Cluster, Cloud Grid Comput.*, May 2010, pp. 495–504.
[25] D. A. Monge, M. Holec, F. Železný, and C. G. Garino, "Ensemble learning of runtime prediction models for gene-expression analysis workflows," *Cluster Comput.*, vol. 18, no. 4, pp. 1317–1329, Dec. 2015.

[26] M. H. Hilman, M. A. Rodriguez, and R. Buyya, "Task runtime prediction in scientific workflows using an online incremental learning approach," in *Proc. IEEE/ACM 11th Int. Conf. Utility Cloud Comput. (UCC)*, Dec. 2018, pp. 93–102.
[27] T. Pham, J. J. Durillo, and T. Fahringer, "Predicting workflow task execution time in the cloud using a two-stage machine learning approach," *IEEE Trans. Cloud Comput.*, vol. 8, no. 1, pp. 256–268, Jan. 2020.
[28] L. Jiang, H. Huang, and Z. Ding, "Path planning for intelligent robots based on deep *Q*-learning with experience replay and heuristic knowledge," *IEEE/CAA J. Autom. Sinica*, vol. 7, no. 4, pp. 1179–1189, Jul. 2020.
[29] R. Zhu, H. Liu, L. Liu, X. Liu, W. Hu, and B. Yuan, "A blockchain-based two-stage secure spectrum intelligent sensing and sharing auction mechanism," *IEEE Trans. Ind. Informat.*, vol. 18, no. 4, pp. 2773–2783, Apr. 2022.
[30] H. Huang, W. Miao, Z. Li, J. Tian, C. Wang, and G. Min, "Enabling energy trading in cooperative microgrids: A scalable blockchain-based approach with redundant data exchange," *IEEE Trans. Ind. Informat.*, vol. 18, no. 10, pp. 7077–7085, Oct. 2022.
[31] C. H. Liu, Q. Lin, and S. Wen, "Blockchain-enabled data collection and sharing for industrial IoT with deep reinforcement learning," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3516–3526, Jun. 2019.
[32] W. Zheng, W. Wang, G. Wu, C. Xue, and Y. Wei, "Fog computing enabled smart grid blockchain architecture and performance optimization with DRL approach," in *Proc. IEEE 8th Int. Conf. Comput. Sci. Netw. Technol. (ICCSNT)*, Nov. 2020, pp. 41–45.
[33] L. Yang, M. Li, P. Si, R. Yang, E. Sun, and Y. Zhang, "Energy-efficient resource allocation for blockchain-enabled industrial Internet of Things with deep reinforcement learning," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2318–2329, Feb. 2021.
[34] Y. Xu, L. Yu, G. Bi, M. Zhang, and C. Shen, "Deep reinforcement learning and blockchain for peer-to-peer energy trading among microgrids," in *Proc. Int. Conf. Internet Things (iThings) IEEE Green Comput. Commun. (GreenCom) IEEE Cyber, Phys. Social Comput. (CPSCom) IEEE Smart Data (SmartData) IEEE Congr. Cybermatics (Cybermatics)*, Nov. 2020, pp. 360–365.
[35] C. Qiu, X. Ren, Y. Cao, and T. Mai, "Deep reinforcement learning empowered adaptivity for future blockchain networks," *IEEE Open J. Comput. Soc.*, vol. 2, pp. 99–105, 2021.
[36] A. A. Donovan and B. W. Kernighan, *The Go Programming Language*, 1st ed. Reading, MA, USA: Addison-Wesley, 2015.
[37] N. Auger, V. Jugé, C. Nicaud, and C. Pivoteau, "On the worst-case complexity of TimSort," in *Proc. 26th Annu. Eur. Symp. Algorithms (ESA)*, vol. 112, Y. Azar, H. Bast, and G. Herman, Eds. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 4:1–4:13.
[38] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, and R. Kern, "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020.
[39] C. A. Hoare, "Quicksort," *Comput. J.*, vol. 5, no. 1, pp. 10–16, 1962.
[40] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Math.*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
[41] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Reading, MA, USA: Addison-Wesley, 2011.
[42] D.-D. Zhu and J.-Q. Sun, "A new algorithm based on Dijkstra for vehicle path planning considering intersection attribute," *IEEE Access*, vol. 9, pp. 19761–19775, 2021.
[43] A. Buzachis, A. Celesti, A. Galletta, J. Wan, and M. Fazio, "Evaluating an application aware distributed Dijkstra shortest path algorithm in hybrid cloud/edge environments," *IEEE Trans. Sustain. Comput.*, vol. 7, no. 2, pp. 289–298, Apr. 2022.
[44] D. Harris and S. Harris, *Digital Design and Computer Architecture*, 2nd ed. Oxford, U.K.: Morgan Kaufmann, 2012.
[45] F. S. Perotto and L. Vercouter, "Tuning the discount factor in order to reach average optimality on deterministic MDPs," in *Artificial Intelligence XXXV*, M. Bramer and M. Petridis, Eds. Cham, Switzerland: Springer, 2018, pp. 92–105.

GAETANO VOLPE received the M.Sc. degree in computer and automation engineering from eCampus University, Novedrate, Italy, in 2020. He is currently pursuing the Ph.D. degree with the Laboratory of Control and Automation, Polytechnic University of Bari. He is also an independent consultant for private companies in the field of cybersecurity, IT infrastructures, and software architectures. His research interests include cybersecurity topics, blockchains in manufacturing environments, discrete-event systems, and petri nets.

AGOSTINO MARCELLO MANGINI (Senior Member, IEEE) received the Laurea degree in electronics engineering and the Ph.D. degree in electrical engineering from the Polytechnic University of Bari, Bari, Italy, in 2003 and 2008, respectively. He has been a Visiting Scholar with the University of Zaragoza, Zaragoza, Spain. He is currently an Associate Professor with the Department of Electrical and Information Engineering, Polytechnic University of Bari. He has authored or coauthored over 90 printed publications. His current research interests include modeling, simulation, control of discrete-event systems, petri nets, supply chains and urban traffic networks, distribution and internal logistics, the management of hazardous materials, the management of drug distribution systems, and healthcare systems. He was on the Program Committees of the 20072015 IEEE International SMC Conference on Systems, Man, and Cybernetics and the 2009 IFACWorkshop on Dependable Control of Discrete Systems. He was on the Editorial Board of the 2017 IEEE Conference on Automation Science and Engineering.

MARIA PIA FANTI (Fellow, IEEE) received the Laurea degree in electronic engineering from the University of Pisa, Pisa, Italy, in 1983. She was a Visiting Researcher with the Rensselaer Polytechnic Institute of Troy, New York, in 1999. Since 1983, she has been with the Department of Electrical and Information Engineering, Polytechnic University of Bari, Italy, where she is currently a Full Professor in system and control engineering and the Chair of the Laboratory of Automation and Control. Her research interests include management and the modeling of complex systems, such as transportation, logistics and manufacturing systems, discrete event systems, petri nets, consensus protocols, and fault detection. She has published more than 315 papers and two textbooks on her research topics. She was a Senior Editor of the IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING and an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS. She was a Member at Large of the Board of Governors of the IEEE Systems, Man, and Cybernetics Society. She is currently a member of the AdCom of the IEEE Robotics and Automaton Society and the Chair of the Technical Committee on Automation in Logistics of the IEEE Robotics and Automation Society. She was the General Chair of the 2011 IEEE Conference on Automation Science and Engineering, the 2017 IEEE International Conference on Service Operations and Logistics, and Informatics and the 2019 IEEE Systems, Man, and Cybernetics Conference.

Open Access funding provided by 'Politecnico di Bari' within the CRUI CARE Agreement