Collaborative Clustering Parallel Reinforcement Learning for Edge-Cloud Digital Twins Manufacturing System

Fan Yang[1], Tao Feng[2], Fangmin Xu[1,*], Huiwen Jiang[1], Chenglin Zhao[1]

[1] School of Information and Communication Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2] China Tendering Center for Mechanical and Electrical Equipment, Ministry of Industry and Information Technology, Beijing 100142, China
[*] The corresponding author, email: xufm@bupt.edu.cn

Abstract: To realize high-accuracy physical-cyber digital twin (DT) mapping in a manufacturing system, a huge amount of data need to be collected and analyzed in real-time. Traditional DTs systems are deployed in cloud or edge servers independently, whilst it is hard to apply in real production systems due to the high interaction or execution delay. This results in a low consistency in the temporal dimension of the physical-cyber model. In this work, we propose a novel efficient edge-cloud DT manufacturing system, which is inspired by resource scheduling technology. Specifically, an edge-cloud collaborative DTs system deployment architecture is first constructed. Then, deterministic and uncertainty optimization adaptive strategies are presented to choose a more powerful server for running DT-based applications. We model the adaptive optimization problems as dynamic programming problems and propose a novel collaborative clustering parallel Q-learning (CCPQL) algorithm and prediction-based CCPQL to solve the problems. The proposed approach reduces the total delay with a higher convergence rate. Numerical simulation results are provided to validate the approach, which would have great potential in dynamic and complex industrial internet environments.

Keywords: edge-cloud collaboration; digital twins; job shop scheduling; parallel reinforcement learning

Received: Apr. 07, 2021
Revised: Jun. 21, 2021
Editor: Fuhong Lin

I. INTRODUCTION

With the increasing demands for manufacturing simulation, digital twin (DT) is proposed to provide more comprehensive analytical and predictive capabilities which plays an important role in manufacturing [1]. DTs are characterized by the seamless integration between physical and cyber spaces, which provide a real-time, bi-direction, and dynamic physical-cyber mapping pattern [2]. With the help of DTs, companies improve the efficiency in design [3], [4], manufacturing and services [2], [5], and DT-based production scheduling, supply chain optimization, fault diagnosis system are developed [6–8].

DT is a technology that requires high real-time performance, the interaction delay between physical entities and virtual entities will have a great impact on the accuracy of digital twin modeling. However, few existing studies have considered the delay optimization between physical entities and virtual entities. Traditional DTs modeling only consider three-dimensional virtual modeling and mechanism modeling but ignored the time-dimensional modeling [9, 10].

Tradition DTs are typically deployed in cloud computing servers. In ref. [5, 11, 12], cloud-based DTs is proposed which could guarantee global optimization when execution DTs system, while the interaction delay between cloud server and physical device is large which cannot meet the time-dimensional consistency requirement. Therefore, some studies deploy DTs in edge servers, which could improve the time-dimensional consistency compared with cloud-based

DTs [13, 14]. However, edge computing faces the problem of limited, heterogeneous and dynamic computing capacity, which thus brings extra execution delay and cannot support the operation of DT applications stably.

To guarantee the reliable DT consistency in time dimension, DT system need to reduce the delay caused by physical-virtual interaction and execution. Since DTs needs to collect and analyze huge amount of data with limited network communication and computation resources to support DT-based applications mentioned before. This lead to high data transmission and execution latency which has a great impact on the accuracy of the system. Therefore, reducing interaction and execution delay in DT system is a research problem.

In this work, we innovatively focus on the physical-virtual consistency of DT in time dimension. Inspired by efficient resource scheduling technology, a novel DTs application deployment and execution scheme is proposed to minimize both interaction and execution delay. To be specific, we construct a novel edge-cloud collaborative architecture to support the DT-based job shop scheduling (JSS) application, and optimize the running position of DT-based applications to minimize the total delay according to the application attributes and cloud-edge resources status. The optimization problem could be formulated as an adaptively dynamic programming problem, and deterministic and uncertainty scenarios are considered.

In deterministic scenario, we innovatively consider the problem of “struggle” workers in distributed system. By clustering edge servers according to their computing capacities, we enable the cooperation between edge servers and thus avoid the extra delay caused by strong nodes waiting for weak nodes. Therefore, a novel collaborative clustering parallel Q-learning (CCPQL) algorithm is presented, which not only minimizes the total delay, but also brings lower adaptive command decision latency due to the distributed characteristic.

In uncertainty scenario, the required computation capacity of the DT-based JSS application is difficult to evaluate. Since the computational complexity of JSS algorithm and underlying DT module is difficult to obtain. Moreover, the application has too many input dimensions and it is impossible to obtain the required computation capacity for each input. Therefore, in order to solve the uncertainty problem, a novel prediction based CCPQL is utilized to predict the incomplete collection of application attributes information.

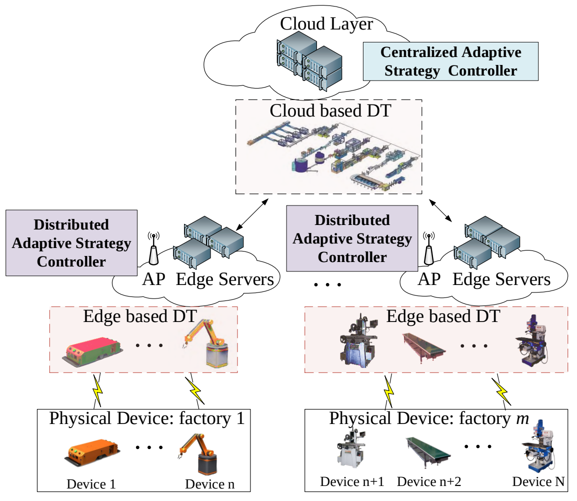
Cloud Layer
Centralized Adaptive Strategy Controller
Cloud based DT
Distributed Adaptive Strategy Controller
AP
Edge Servers
Distributed Adaptive Strategy Controller
AP
Edge Servers
Edge based DT
Edge based DT
Physical Device: factory 1
Device 1
Device n
Physical Device: factory m
Device n+1
Device n+2
Device N

Figure 1. *Edge-cloud collaboration digital twins system in smart factory.*

Simulation results demonstrate the proposed approaches achieve lower total delay with higher convergence rate compared with conventional approaches. Thus, the time-dimensional consistency between physical-virtual entities can be greatly improved.

II. EDGE-CLOUD COLLABORATION DIGITAL TWINS SYSTEM MODEL

2.1 System Architecture

In this paper, an edge-cloud collaboration DT-based JSS application system architecture is proposed, which is shown in Figure 1. The system architecture includes three layers: cloud layer, edge layer, and device layer. Cloud layer and edge layer are connected by wired channels, edge layer and device layer are connected by wireless channels. Three layers work collaboratively to minimize the total latency of DT-based JSS applications. As one of the decisive factors influencing manufacturing efficiency, JSS application assigns a set of jobs to the matched production devices in available time slots to optimize given objectives, such as delivery time or energy consumption [5, 15].

Device layer includes physical production devices and sensors which are used for executing production and sensing real time device status. Edge layer deploys the workshop-based DTs system and JSS application. The workshop-based DTs system is the virtual mapping of workshop scale production devices. Cloud layer includes order system, factory-level DTs system and JSS application. The factory-based DTs system is the virtual mapping of factory scale production devices. Based on this mode, the workshop level JSS application can be executed both in cloud server and edge servers. The cloud-based JSS has a global view, which allows both large-scale factory level scheduling and workshop level scheduling. Thus, the adaption strategy controllers make adaptive decisions based on the real-time resources status and application attributes. To be specific, the adaption strategy controllers choose a server from candidate servers (including cloud server and edge servers) with the minimum interaction and execution delay.

In deterministic environment, the adaption strategy controller acquires complete decision related real-time status information to make adaptive decisions which includes cloud and edge communication and computing resources states, as well as application attributions. However, in uncertainty environment, it is difficult to obtain the required computation capacity of JSS applications. Thus, the adaption strategy controller needs to predict the required computation capacity of DT-based JSS application. Then, the adaption decision is made according to the prediction and actual data.

2.2 Intelligence Job-Shop Scheduling Application

JSS application needs to get huge amount of data which includes production system states and the order information from the customers, such as delivery time, delivery quantity (product type, quantity, etc.). Then, the order system converts the order information as the input of JSS application, which includes the workpieces, the required production devices, production processes, and working hours of each process, and maximum delivery tolerance of the order. The JSS application schedules production processes according to the above information.

DT-based JSS application could be described as follows. We consider that there are n independent work pieces $W=\{W_1, W_2, ..., W_n\}$, m production devices $D = \{D_1, D_2, ..., D_m\}$, i processes $P = \{P_1, P_2, ..., P_i\}$ which are arranged according to a certain order. The computing capacity required for running JSS application tasks u could be represented as o_u, which mainly related to the input size (including numbers of workpieces and devices) of the application when the JSS application algorithm is fixed. Meanwhile, real-time devices data should be transmitted for running the application, and we use $n_{u,m}(t)$ to represent the size of data collected by sensors on production device D_m in time slot t. Thus the total transmission data size is $n_u(t) = \sum_{d=1}^{m} n_{u,d}(t)$.

2.3 Edge Computing Based Digital Twins Application

Considering the wireless channels between production devices and edge servers are block time varying channels, i.e., the channels remain constant during a time slot and change independently among different time slots, which can be modeled as finite state Markov channels. According to the reciprocity of uplink and downlink channels, we assume the quality of the channel between production device D_m and edge server e can be represented by the received SNR $\gamma_{m,e}(t)$, $t \in \{0, 1, 2, ..., T-1\}$ of production devices D_m [16]. Thus, the transmission rate between production device D_m and edge server e at time slot t is:

$$r_{m,e}(t) = b_{m,e}(t)\log_2(1 + \gamma_{m,e}(t)), \tag{1}$$

where $b_{m,e}$ is the allocated bandwidth of edge server e to production device D_m.

The edge server will fuse the data after receiving all data transmitted in time slot t. Thus, the total transmission latency between production devices and edge server at time slot t equals to the maximum device-edge transmission latency which could be denoted as:

$$t_{M,e}(t) = \max_{m\in M}\{n_{u,m}(t)/r_{m,e}(t)\}, \tag{2}$$

Edge servers receive all data at t_1, where $t_1 = t + t_{M,e}(t)$. After receiving the data, we then consider the edge computation model. Each edge server deploys a part of DTs, and edge servers are connected via wired channel which means the interaction delay among edge servers could be ignored. Since all edge servers deploy applications, all edge servers can execute JSS applications. We consider that the computation capacity of edge server e at t_1 is $f_e(t_1)$.

Thus, the latency for running the DT-based JSS application in edge layer could be denoted as l_{edge}. The JSS results are sent back at time t_2 where $t_2 = t_1 + t_{comp\_e}(t_1)$. Note that, the JSS results send back delay is small and thus could be ignored.

$$\begin{aligned} l_{edge}(t) &= t_{M,e}(t) + t_{comp\_e}(t_1) \\ &= \max_{m \in M}\{\frac{n_{u,m}(t)}{b_{m,e}(t)\log_2(1+\gamma_{m,e}(t))}\} + \frac{o_u(t)}{f_e(t_1)}. \end{aligned} \tag{3}$$

2.4 Cloud Computing Based Digital Twins Application

We consider the wired channels between cloud server c and edge servers are relatively stable channels. We assume the transmission rate between cloud and servers at time slot t as $r_{e,c}(t)$. Considering the edge servers send the data to cloud server after receiving and preprocessing. The edge servers collect all data at t_1, then collected data are preprocessing at the edge servers, and the size of the preprocessed data could be denoted as $kn_u(t)$, where $k \in (0,1)$ is compression coefficient [17]. The preprocessing delay could be denoted as:

$$t_{pre}(t_1) = \alpha k n_u(t)/f_e(t_1), \tag{4}$$

where α is the correlation coefficient of data size and required computation capacity for preprocessing task. The preprocessed data are transmitted to cloud server at time t_3, where $t_3 = t_1 + t_{pre}(t_1)$. The transmission latency between edge server e and cloud server c at time slot t_3 could be denoted as:

$$t_{e,c}(t_3) = kn_u(t)/r_{e,c}(t_3). \tag{5}$$

Then, we consider the cloud computation model. We consider that the computation capacity of cloud server at t_4 is $f_c(t_4)$, where $t_4 = t_3 + t_{e,c}(t_3)$. Since the computation capacity used for preprocessing data is provided by edge servers, then the required cloud capacity for running the rest part of the application could be denoted as $\beta o_u(t)$, where $\beta \in (0,1)$. Thus, the total required computation capacity of JSS application includes data preprocessing capacities and application execution capacities, $\alpha k n_u(t) + \beta o_u(t) = o_u(t)$. And the computation delay could be denoted as:

$$t_{comp\_c}(t_1) = \beta o_u(t)/f_c(t_4). \tag{6}$$

Thus, the cloud application execution delay could be denoted. Note that, the JSS results send back delay is small and thus could be ignored.

$$\begin{aligned} l_{cloud}(t) &= t_{M,e}(t) + t_{pre}(t_1) + t_{e,c}(t_3) + t_{comp\_c}(t_1) \\ &= \max_{m \in M}\{\frac{n_{u,m}(t)}{b_{m,e}(t)\log_2(1+\gamma_{m,e}(t))}\} + \frac{\alpha k n_u(t)}{f_e(t_1)} \\ &+ \frac{kn_u(t)}{r_{e,c}(t_3)} + \frac{\beta o_u(t)}{f_c(t_4)}. \end{aligned} \tag{7}$$

III. PROBLEM FORMULATION

Based on the above analysis, we aim to minimize the long term total interaction and execution delay of all JSS tasks CT_{total} in T time slots under the constraints of servers maximum bandwidth and computation capacities. Thus the optimization problem could be formulated as:

$$\min_{\mathcal{A}} CT_{total} = \min_{\mathcal{A}} \sum_{t=0}^{T-1}\sum_{i=1}^{I}(l_{edge_i}(t), l_{cloud_i}(t)), \tag{8}$$

$$s.t. \sum\nolimits_{d=1}^{D} b_{d,n}(t) \le \mathrm{B} \tag{8a}$$

$$0 \le f_e(t) \le f_e^{\max} \tag{8b}$$

$$0 \le f_c(t) \le f_c^{\max} \tag{8c}$$

where $\mathcal{A}= \{a_c, a_{e_1}, a_{e_2}, ..., a_{e_E}\}$ denotes action space which includes the choice of cloud server and edge servers, $i \in \{1, ...I\}$ denotes the JSS tasks. Note that, constraint (8a) represents the bandwidth limits, where the bandwidth allocated to the sensors cannot exceed the total bandwidth of the edge server e. Constrains (8b) and (8c) denote the maximum computation capacities.

3.1 Deterministic Scenario

The complete information are collected for making adaptive decisions, which includes communication and computation resources states of edge and cloud servers, and the application attributes. Thus, the adaptive controller makes adaptive decisions based on the deterministic information. When there are order changes and other emergency events, the adaptive controller dynamically update the adaptive results.

3.2 Uncertainty Scenario

The algorithmic complexity of DT-based JSS application is difficult to obtain directly in practice. This is because the operation of the application needs to call different part of the DT system each time, and the production scheduling application needs to be run based on the DTs and order information. Moreover, the whole process needs to be intelligently analyzed and visually expressed. Therefore, instead of analyzing the code for obtaining the algorithmic complexity, it is better to use fitting and prediction method to estimate the required computation capacity of the application in real time. Due to the non-linear characteristic of this fitting and prediction problem, neural network is used in solving this problem.

Thus, we learn the relationship between the required computation capacity (algorithm complexity) and input dimension of JSS application to predict the required computation capacities. Although there are some errors when fitting and predicting, the experiment proves that the error of the estimation based on the NN prediction is small, and the impact on subsequent calculations is negligible. Finally, the adaptive strategy could be made based on the prediction data and real-time data. When there are order changes and other emergency events, the adaptive controller dynamically update the adaptive results.

IV. PROPOSED METHOD

The optimization problem defined in Eq.(8) is a multi-stage decision making problem, the decisions of the next stage are often affected by the previous state. Thus, the activity route of a process needs to be completely determined, and the globally optimal solution of which is difficult to obtain. The exploration characteristic in reinforcement learning (RL) is suitable for finding this optimal solution. In this work, we develop two approaches to solve the above problems in two scenarios.

Algorithm 1. *Edge server cluster algorithm.*

```
Adaptive Controller Executes:
    Input: $K, \mathcal{X}, \mathcal{T}$
    Output: $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_K\}$
 1: Sort $\mathcal{X}_{sort} = sortDescent(\mathcal{X})$.
 2: The first $K$ elements of $\mathcal{X}_{sort}$ are the initial elements of $K$ clusters $\mathcal{C} = \{\mathcal{C}_1, , \mathcal{C}_2, ..., \mathcal{C}_K\}$.
 3: for $i = (k+1) : E$ do
 4:     for $j = 1 : K$ do
 5:         $f_{\mathcal{C}_j} = \sum_{e \in \mathcal{C}_j} f_e$
 6:     end for
 7:     Sort $sortAscend(f_{\mathcal{C}})$.
 8:     Choose the cluster with smaller $f_{\mathcal{C}_j}$ until $\mathcal{T}_{e_j, e_i} = 1, \forall e_j \in \mathcal{C}_j$.
 9:     Assign $e_i$ to cluster $\mathcal{C}_j$.
10: end for
11: return $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_K\}$.
```

Algorithm 2. *Deterministic adaptive algorithm with collaborative clustering PQL (CCPQL).*

```
Adaptive Controller Executes:
 1: Initialize global $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily expect that $Q(terminal, \cdot) = 0$,
 2: $clusters \leftarrow EdgeServerCluster(K, \mathcal{X}, \mathcal{T})$
 3: for each round $t = 1, 2, ...$ do
 4:     $Q(s,a) = \sum_{k=1}^{K} \frac{n_k}{n} Q^k(s,a)$
 5: end for
 6: $a(t) = \arg\min_a Q(s,a)$.
Edge layer client cluster $k$ executes:
 7: Initialize local $Q^k(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily expect that $Q^k(terminal, \cdot) = 0$.
 8: for each $episode = 1, 2, ...$ do
 9:     for each step of episode do
10:         Choose $A$ from $S$ using $Q^k(s,a)$, observe $R$, $S'$.
11:         Update $Q^k(s,a)$.
12:         $S \leftarrow S'$.
13:         Until $S$ is terminal.
14:     end for
15: end for
```

4.1 Reward Function

The reward R_{total} in each step is defined as the latency reduction rate compared to traditional cloud computing approach, which could be denoted as:

$$R_{total} = (CT_{cloud} - CT)/CT_{cloud}, \quad (9)$$

where CT is the total interaction and execution delay of adaptive approach in each step, and CT_{cloud} is the total delay of cloud based approached in each step. Thus, the problem can be denoted as $\max_{\mathcal{A}} R_{total}$.

4.2 Deterministic Adaptive Algorithm

Traditional centralized adaptive controller deployed in cloud layer brings extra round trip decision delay compared with edge based adaptive controllers. PRL could reduce the learning time of single-agent RL algorithms (SARL) by increasing the number of agents solving the task and sharing their experiences. And PRL could be a good solution to make distributed adaptive strategies. However, distributed PRL algorithms face a synchronization problem due to the heterogeneity of workers capabilities.

The resource allocation algorithm based on PRL needs to use application attributes, resource state and other environmental information as samples for training. Due to the privacy of these samples, samples can only be shared within a factory. Therefore, instead of collecting all samples and then redistributed according to the load and capacity of edge computing servers, we adopt the clustering method based on server capability and load to ensure synchronization of parameter update and avoid the influence of “struggle” node on global performance.

Thus, we propose a novel CCPQL algorithm for large-scale distributed network environment which could solve the above synchronize problem. The proposed algorithm clusters the workers according to the computation capacities and trust value, and the aim is to balance the computation capacities of different clusters. In this way, the problem of inefficient training caused by the “struggle” node can be solved. Note that, the workers in one cluster trust each other and can share data.

We assume that there are E edge servers ($E = e_1, ..., e_i, ..., e_E$) in the system. Trust values $SD_{e_i,e_j} = 1$ if the edge server e_i and e_j trust each other, otherwise $SD_{e_i,e_j} = 0$. The trust values could be defined in matrix $\mathcal{T}$.

$$\mathcal{T} = \begin{pmatrix} SD_{e_1,e_1} & ... & SD_{e_1,e_j} & ... & SD_{e_1,e_E} \\ ... & ... & ... & ... & ... \\ SD_{e_i,e_1} & ... & SD_{e_i,e_j} & ... & SD_{e_i,e_E} \\ ... & ... & ... & ... & ... \\ SD_{e_E,e_1} & ... & SD_{e_E,e_j} & ... & SD_{e_E,e_E} \\ . & & & & \end{pmatrix} \quad (10)$$

In each clustering period, all edge servers form a set $\mathcal{X}$. K edge servers with larger computation capacities are chosen as the initial element of each cluster sets. These elements are taken out of $\mathcal{X}$ and put into cluster set $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_K\}$. Each time the largest element of the remaining elements in $\mathcal{X}$ is put into a trusted cluster with minimum total computing capacities until the $\mathcal{X}$ is empty. Note that, edge servers that do not trust each other cannot be grouped into a cluster. The detail of edge server cluster algorithm is shown in Algorithm 1.

After clustering, each cluster optimize the adaptive policies by updating local Q-tables. And after a period of time, all clusters converge the local policies to form a global optimal policy to make adaptive strategies. In this way the algorithm could synchronize the training pace of different clusters, and obtain a relatively consistent update of the global algorithm. Details of the proposed algorithm is shown in Algorithm 2.

4.3 Prediction Based Uncertainty Adaptive Algorithm

Since the DT-based JSS application is complex and the relationship between application required computation capacity and the input parameter scale is nonlinear. Thus, in this work, neural network (NN) is used for fitting the relationship of the input scale and required computation capacity. Thus, $o_u(t)$ could be predicted in real time, and the uncertainty adaptation decision can be made based on the predicted results and real time data. The parameters of the neural network is determined by θ which includes weights and bias $w_{ij}^{(l)}, b_i^{(l)}$. And $h_\theta(\cdot)$ is the hypothesis of the fitting problem, the prediction problem thus could be determined as $y_{pred} = h_\theta(x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)})$, y_{pred} is the execution delay of DT-based JSS application on a test server. And in order to get accurate prediction results, we update the parameters of the neural network

Algorithm 3. *Prediction based uncertainty adaptive algorithm with CCPQL.*

```
Adaptive Controller Executes:
    Input: Training set $\{(x_n{}^{(1)}, y_n{}^{(1)}), ..., (x_n{}^{(i)}, y_n{}^{(i)})\}$.
    Output: adaptive action decision.
 1: Initialize weights and biases $w_{ij}^{(l)}, b_i^{(l)}$.
 2: while not converge do
 3:     Sample batch data from training set.
 4:     Calculate $\mathcal{L}_{loss}$ on data set to update $w_{ij}^{(l)}, b_i^{(l)}$.
 5: end while
 6: $y_{pred} = h_{w,b}x$.
 7: Compute $o_{u,pred}(t)$ via Eq. (11).
Adaptive controller, edge layer client clusters execute:
 8: For the system, run deterministic adaptive algorithm with collaborative clustering PQL by Algorithm 1.
```

by minimizing the loss function. Thus the required computation capacity of JSS with certain scale of input could be defined as:

$$o_{u,pred}(t) = f_{ser} y_{pred}, \tag{11}$$

where f_{ser} is the computation capacity of the test server used for generating the data sets. After predicting, the required computation capacities of DT-based JSS application could be obtained in real time. And the details of the algorithm is shown in Algorithm 3.

Obviously, the accuracy of the NN-based prediction algorithm will greatly affect the accuracy of the adaptive decision. Therefore, we will further analyze the influence of the prediction accuracy on the accuracy of adaptive decision. We uniformly discrete the value of o_u into N levels, and use the average level $\bar{o}_{u,n}$ of each level to present the required computation capacities of the tasks. After predicting the required computation capacity $o_{u,pred}(t)$ of task $o_{u,n}(t)$. The algorithm will automatically set the nearest level value as the required computation capacity of the task. Therefore, there will be no influence on the adaptive accuracy if $o_{u,pred}(t) - o_{u,n}(t) < o_{u,pred}(t) - o_{u,n'}(t)$, where $o_{u,n'}(t)$ is the required computational capacity of the other tasks. And if $o_{u,pred}(t)$ is in the range of the corresponding level, then there will be no influence on adaptive algorithm. The specific numerical analysis will be presented in the simulation section.

4.4 Algorithm Complexity Analysis

The proposed deterministic adaptive algorithm with CCPQL algorithm and prediction based uncertainty adaptive algorithm with CCPQL has the main objective to improve the convergence rate, avoid the impact of 'struggler worker' and task uncertainty on system adaptive performance. We will analyze the computational complexity, space complexity and communication complexity of the algorithms.

Computational complexity: Algorithm 2 runs in $O\{[E \log E + K^2 \log K(E-K)] + n_r K + n_e m_s K\}$ in training process, where E is the number of edge servers, K is the number of clusters, n_r is the number of parameter aggregations, n_e denotes the number of training episodes, m_s denotes the number of steps in each episode. Since n_e and n_r are large, thus the computational complexity could be simplified to $O(n_r + n_e m_s)$. Traditional QL and PRL runs in $O(n_q m_s)$ and $O(n_r + n_e m_s)$ in training process respectively, where n_q is the number of training episodes of QL [18].

Algorithm 3 runs in $O(n_r + n_e m_s + n_l B_l \sum_{j=1}^{L} M_l N_l)$, where n_l is the number of training episodes, B_l is the batch size, and M_l is the input size of layer l, N_l is the output size of layer l [19] [20]. And in inference process, Algorithm 3 runs in $O(\sum_{l=1}^{L} M_l N_l)$.

Space complexity: The space complexity of Algorithm 2 and Algorithm 3 are given by $O(\sum_{i=1}^{K} m_s |S| |A|)$ and $O(\sum_{l=1}^{L} T_l + \sum_{i=1}^{K} m_s |S| |A|)$, where T_l is the number of weights and bias in layer l.

Communication complexity: It is measured by the number of messages exchanged between agents during learning and inference.

By assuming that one message can contain an array or Q-table, Algorithm 2 runs in $O(n_r + 2Kn_e)$ in training process. Since n_r and n_e are large enough, and thus the communication complexity is simplified to $O(n_r + n_e)$. The communication complexity of Algorithm 3 is also given by $O(n_r + n_e)$ in training process. And the communication complexity of traditional PRL is given by $O(n_r + n_e)$ [18].

In summary, the proposed algorithms improve convergence rate and avoid the impact of 'struggler worker' on system adaptive performance with lower space complexity, the same communication and computation capacity compared with traditional PRL algorithm.

V. SIMULATION RESULTS

In this section, we evaluate the total delay and convergence rate of our proposed CCPQL and prediction-based CCPQL algorithms. Firstly, the simulation parameters setting are introduced. We assume that there are 9 edge servers and cloud server c in the system. The computation and communication capacities of the edge servers could be weak and strong, the specific value are $f_e \in \{3, 1.7\}$GHz, and $r_{m,e} \in \{3, 9\}$ Gbps, the communication and computation capacities of cloud server are relatively static which are defined as $f_c = 3$ GHz, and $r_{e,c} = 0.9$ Gbps. And we divide the servers into 3 clusters of edge servers $\{ec_1, ec_2, ec_3\} \in EC$. After analyzing the application of intelligent scheduling, we preliminarily assume that the application running delay is mainly related to the number of production devices and workpieces. DT-based JSS application is run based on generic algorithm [21], and we obtain the application input dimension changes from 1 to 20, and the delay various from 0.421 sec to 49.955 sec. Thus, the required computation capacity o_u various from 0.653G CPU cycles to 77.43G CPU cycles. To simplify calculation, we define that o_u various from 1G CPU cycles to 75G CPU cycles, and $N = 16$. Intel Core i5, 3.1GHz server is used for running the JSS application, the CPU occupancy of the application is around 48.9%. The amount of data needs to be transmitted is related to devices number, we define the number of devices various from 5 to 9, and the amount of data each production device generated various from 0.5Gb to 0.9Gb. We use a three-layer neural network for fitting and prediction. Finally, the basic parameters of the proposed CCPQL algorithm are defined as $\gamma = 0.9$, $\alpha = 0.1$, and dynamic greedy ε is used which changes from 0.1 to 0.9 in the whole training process.

Figure 2 shows the fitting performance of the NN based prediction approach. The proposed NN based fitting algorithm shows good fitting performance. Thus, the predicted required computation capacity of the application could be used for adaptive decision making in uncertainty scenario. Mean Square Error of the NN-based prediction approach is 0.0019. Considering a task with running delay of 9.6774s, thus $o_u = 14.98G$ CPU cycles. We use NN-based prediction algorithm to predict the required computation capacity for 2000 times. All predicted values are in the range of the corresponding level when $N < 63$. And

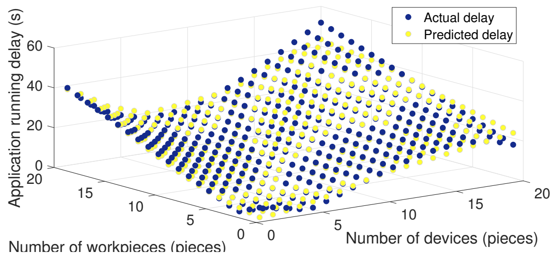

Figure 2. *Application required computation capacity versus input size.*

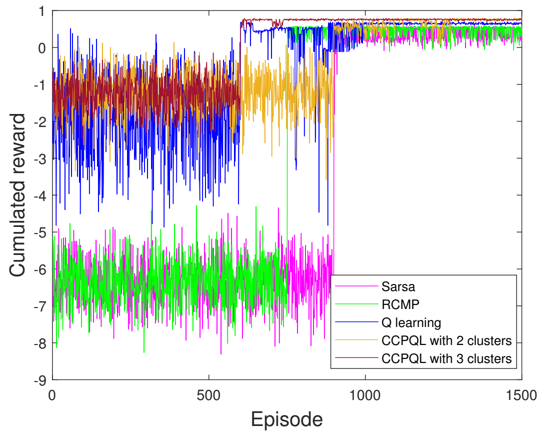

Figure 3. *Converge curve of proposed CCPQL.*

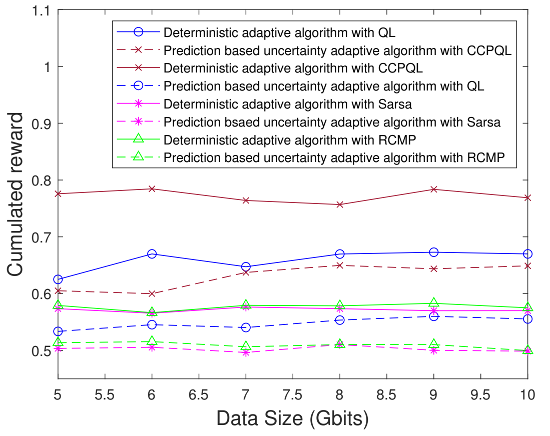

Figure 4. *Cumulated reward of different schemes versus data size.*

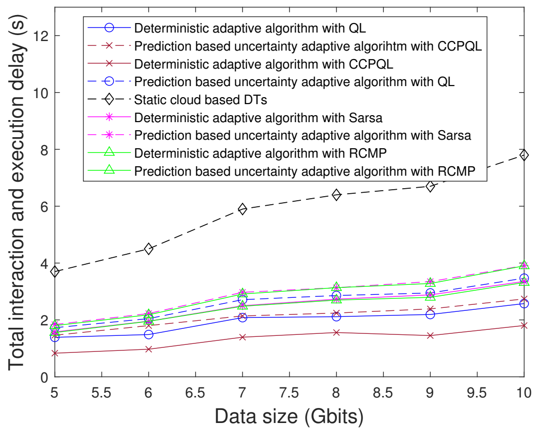

Figure 5. *Cumulated CT_{total} of different schemes versus data size.*

there are 1.1% predicted values out of the range when $N > 63$, which will affect the accuracy of adaptive algorithm. Thus, it is obvious that the finer-grained task division, the higher the accuracy requirement on the NN-based prediction algorithm. However, when N=62, the CCPQL-based adaptation algorithm could already get a fine-grained adaptive decision. Thus, we still choose the current algorithm, although we can predict required computation capacity by using a prediction algorithm with higher complexity and obtain more accurate predicted values.

Figure 3 compare the convergence performance of tradition QL, Sarsa, state-of-art RCMP [22], proposed CCPQL with 2 clusters, and CCPQL with 3 clusters algorithms. It is obvious that the distributed adaptive algorithm converges faster compared with centralized way. Compared with traditional QL, and Sarsa algorithm, the proposed CCPQL has faster convergence rate, and the convergence rate increases with the cluster number. RCMP improves the learning efficiency by means of supervision, which makes RCMP converge fastest among the centralized adaptation algorithms. When the cluster number of CCPQL is greater than three, the convergence rate of CCPQL is much faster than that of RCMP. Thus, it can be seen that the proposed CCPQL has good convergence performance.

Figure 4 and Figure 5 shows the cumulated reward and cumulated interaction and execution delay versus data size of different algorithms. The performance of traditional QL, Sarsa, static approaches and state-of-art RCMP approaches are also shown as contrasts. The proposed algorithms show better latency and reward performance compared with the other approached when the data size changes. This shows that the CCPQL and prediction based CCPQL have good performances with the constant change of data size, which reflects the universality of the proposed approaches.

VI. CONCLUSION

In this paper, we have proposed a novel DTs deployment and execution pattern, which achieved lower interaction delay and analyze delay with higher convergence rate. Firstly, the cloud-edge collaborative DTs application deployment architecture is constructed. Based on the proposed architecture, deterministic and uncertainty application adaptive strategies are introduced. Finally, to solve the adaptive problems, distributed CCPQL and prediction based CCPQL algorithms are presented. Simulation results show that the proposed algorithm can achieve better performances compared with conventional algorithms, which is promising in improving efficiency in actual manufacturing.

ACKNOWLEDGEMENT

This research work is supported by 2019 Industrial Internet Innovation Development Project of Ministry of Industry and Information Technology of P.R. China “Comprehensive Security Defense Platform Project for Industrial/Enterprise Networks”, and Research on Key Technologies of wireless edge intelligent collaboration for industrial internet scenarios (L202017), Natural Science Foundation of China, No.61971050, and BUPT Excellent Ph.D. Students Foundation (CX2020214).

References

[1] M. Grieves, “Digital twin: manufacturing excellence through virtual factory replication,” *White Paper*, vol. 1, no. 2014, 2014, pp. 1–7.

[2] F. Tao, H. Zhang, *et al.*, “Digital twin in industry: State-of-the-art,” *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, 2018, pp. 2405–2415.

[3] C. Zhuang, J. Liu, *et al.*, “Connotation, architecture and trends of product digital twin,” *Computer Integrated Manufacturing Systems*, vol. 23, no. 4, 2017, pp. 753–768.

[4] F. Tao, F. Sui, *et al.*, "Digital twin-driven product design framework," *International Journal of Production Research*, vol. 57, no. 12, 2019, pp. 3935–3953.
[5] M. Schluse, M. Priggemeyer, *et al.*, "Experimentable digital twins—streamlining simulation-based systems engineering for industry 4.0," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 4, 2018, pp. 1722–1731.
[6] Y. Fang, C. Peng, *et al.*, "Digital-twin-based job shop scheduling toward smart manufacturing," *IEEE transactions on Industrial Informatics*, vol. 15, no. 12, 2019, pp. 6425–6435.
[7] F. Ameri and R. Sabbagh, "Digital factories for capability modeling and visualization," in *IFIP International Conference on Advances in Production Management Systems*. Springer, 2016, pp. 69–78.
[8] Y. Xu, Y. Sun, *et al.*, "A digital-twin-assisted fault diagnosis using deep transfer learning," *IEEE Access*, vol. 7, 2019, pp. 19 990–19 999.
[9] R. Minerva, G. M. Lee, *et al.*, "Digital twin in the iot context: a survey on technical features, scenarios, and architectural models," *Proceedings of the IEEE*, vol. 108, no. 10, 2020, pp. 1785–1824.
[10] T. Mukherjee and T. DebRoy, "A digital twin for rapid qualification of 3d printed metallic components," *Applied Materials Today*, vol. 14, 2019, pp. 59–65.
[11] K. M. Alam and A. El Saddik, "C2ps: A digital twin architecture reference model for the cloud-based cyber-physical systems," *IEEE access*, vol. 5, 2017, pp. 2050–2062.
[12] C. Qiu, X. Wang, *et al.*, "Networking integrated cloud–edge–end in iot: A blockchain-assisted collective q-learning approach," *IEEE Internet of Things Journal*, vol. 8, no. 16, 2020, pp. 12 694–12 704.
[13] Y. Lu, X. Huang, *et al.*, "Low-latency federated learning and blockchain for edge association in digital twin empowered 6g networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, 2020, pp. 5098–5107.
[14] X. Xu, B. Shen, *et al.*, "Service offloading with deep q-network for digital twinning-empowered internet of vehicles in edge computing," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, 2020, pp. 1414–1423.
[15] M. Zhang, F. Tao, *et al.*, "Digital twin enhanced dynamic job-shop scheduling," *Journal of Manufacturing Systems*, vol. 58, 2021, pp. 146–156.
[16] C. Qiu, H. Yao, *et al.*, "Deep q-learning aided networking, caching, and computing resources allocation in software-defined satellite-terrestrial networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, 2019, pp. 5871–5883.
[17] T. Alfakih, M. M. Hassan, *et al.*, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on sarsa," *IEEE Access*, vol. 8, 2020, pp. 54 074–54 084.
[18] C. Jin, Z. Allen-Zhu, *et al.*, "Is q-learning provably efficient?" *Advances in Neural Information Processing Systems*, vol. 31, 2018.
[19] B. Li, P. Chen, *et al.*, "Random sketch learning for deep neural networks in edge computing," *Nature Computational Science*, vol. 1, no. 3, 2021, pp. 221–228.
[20] M. Camelo, M. Claeys, *et al.*, "Parallel reinforcement learning with minimal communication overhead for iot environments," *IEEE Internet of Things Journal*, vol. 7, no. 2, 2019, pp. 1387–1400.
[21] L. Xixing, D. Baigang, *et al.*, "Production scheduling optimization method for textile machinery manufacturing enterprise based on improved bee algorithm," in *2017 36th Chinese Control Conference (CCC)*. IEEE, 2017, pp. 2805–2811.
[22] F. L. Da Silva, P. Hernandez-Leal, *et al.*, "Uncertainty-aware action advising for deep reinforcement learning agents," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 04, 2020, pp. 5792–5799.

Biographies

Fan Yang received the B.S. degree in e-commerce engineering with law from the Queen Mary University of London, in 2017, and the B.Admin. degree in e-commerce engineering with law from the Beijing University of Post and Telecommunications (BUPT), in 2017. She is currently pursuing the Ph.D. degree with the Key Laboratory of Universal Wireless Communications, Ministry of Education, BUPT, China. Her research interests include edge computing, digital twins and blockchain.

Tao Feng currently working in China Tendering Center for Mechanical and Electrical Equipment, Ministry of Industry and Information Technology. His research interests include industrial internet of things (IIoT), digital twins, and Internet of Things (IoT).

Fangmin Xu received the M.S. and Ph.D. degrees in communication engineering from the Beijing University of Posts and Telecommunication (BUPT), China, in 2003 and 2008, respectively. He is currently an Associate Professor with the School of Information and Communication Engineering, BUPT, China. From 2008 to 2014, he was with Samsung Electronics, where he actively contributed to 3GPP LTE/LTE-A and IEEE 802.16m. He has authored two books, 20 peer-reviewed international research papers, and 50 standard contributions and the Inventor of 15 issued or pending patents among which four have been adopted in the specifications of 4G (3GPP LTE/LTE-A and IEEE 802.16m) standards. His research interests include advance technologies in wireless networks, especially the IoT field.

Huiwen Jiang received the B.S. degree from Beijing University of Post and Telecommunications (BUPT). He is currently pursuing the M.S. degree with the Key Laboratory of Universal Wireless Communications, Ministry of Education, BUPT, China. His research interests include edge computing and digital twins.

Chenglin Zhao received the bachelor's degree in radio-technology from Tianjin University, in 1986, and the master's degree in circuits and systems and the Ph.D. degree in communication and information system from the Beijing University of Posts and Telecommunications, Beijing, China, in 1993 and 1997, respectively, where he is currently a Professor. His research is focused on emerging technologies of short-range wireless communication, cognitive radios, and industrial internet.