Received 15 January 2022; revised 19 February 2022; accepted 2 March 2022. Date of publication 14 March 2022; date of current version 25 March 2022.

Digital Object Identifier 10.1109/OJCOMS.2022.3158417

Workflow Makespan Minimization for Partially Connected Edge Network: A Deep Reinforcement Learning-Based Approach

KAIGE ZHU[1,2], ZHENJIANG ZHANG[1,2], FENG SUN[1,2], AND BO SHEN[1,2]

[1]School of Electronic and Information Engineering, Beijing Jiaotong University, Beijing 100044, China

[2]Key Laboratory of Communication and Information Systems, Beijing Municipal Commission of Education, Beijing 100032, China

CORRESPONDING AUTHOR: Z. ZHANG (e-mail: zhangzhenjiang@bjtu.edu.cn)

This work was supported in part by the Fundamental Research Funds for the Central Universities under Grant 2020YJS020, and in part by the National Natural Science Foundation of China under Grant 62173026.

ABSTRACT The ever advances in wireless communication and mobile networks have brought novel workflow-formed applications, such as virtual reality and live-streaming, to our daily life. Arousing a growing need for workflow execution efficiency. An edge network is widely considered a promising way of bridging the gap between intensive resource demand and the limited computation capabilities of mobile terminals. However, when an edge network is partially connected, ordinary workflow scheduling algorithms suffer degradations as the data transmission time is prolonged. In this paper, we address the challenge of workflow makespan minimization in a partially connected edge network. Contrary to the general assumptions of a fully connected edge network, the edge servers under discussion are partly interconnected but can be reached within limited hops by using different paths. Here, the placement of interdependent tasks and selection of routing paths are two major factors that influence the makespan. We first propose a critical path analysis based dynamic task sorting algorithm to determine the scheduling order of tasks. Then the path quality is introduced as a reflection of path availability and is employed as the major indicator in selecting disjoint subpaths. We further model the workflow scheduling process into a Markov decision process and propose a reinforcement learning–based workflow embedding (RLWE) scheme to minimize the makespan of the workflow. With the fine-trained agent, the proposed scheme can coordinate the demand of computing resources and routing paths of interdependent tasks and provide a near-optimal makespan of the workflow. Numerical results validate the feasibility of our proposed scheme as its performance exceeds existing baselines with an improved quality of service in terms of makespan.

INDEX TERMS Workflow scheduling, edge computing, multipath routing, deep reinforcement learning.

I. INTRODUCTION

INTELLIGENT mobile edge devices have emerged in recent years. With the further development of the hardware, the computing capability of mobile devices increases exponentially according to Moore's law. These advances promotes new paradigms of mobile applications such as virtual reality (VR) [1] and live-streaming [2]. However, the computing capability of the mobile devices is still insufficient for these resource-intensive services [3], [4], thus further exacerbating the scheduling problems of interdependent functions known as workflows.

Offloading resource-intensive tasks to edge servers for further execution is now considered a promising solution as the computational capability of edge servers is much higher than that of mobile devices. As a supplement to cloud computing, edge computing is proposed to reduce the transmission delay by using task offloading. By sinking the service to the edge of the network, edge computing can provide computation service at the proximity to reduce the transmission delay, this reduction is critical to some delay-sensitive applications. In addition, the traffic of transmitted data in the core network decreases significantly as tasks are executed near the end-user. Currently, there are many task offloading and resource allocation algorithms designed for edge computing under different scenarios [1], [2], [5], but research on workflow scheduling in edge networks is still in its infancy [6]–[9].

Workflow scheduling earns its popularity alongside the widespread use of cloud computing. Chang *et al.* [10] conclude that cloud–based workflow scheduling scenarios have 4 major issues as follows: (1) dispatching order of tasks with dependency, (2) cloud resource allocation, (3) prediction of task execution time, and (4) dispatching order under deadline constraints. The authors propose a topological sort based algorithm to determine the dispatching order while employing rough set theory for execution time prediction. Calheiros *et al.* [11] consider scientific workflow scheduling with a time limit under a public cloud and propose an algorithm that minimizes the cost of renting virtual machines (VMs). Qin *et al.* [12], however, focus on the tradeoff between the makespan and the budget of the workflow in a cloud network. The authors propose a multiobjective reinforcement learning based scheme as the solution and employ the Chebyshev scalarization function to coordinate the weight selection problem. Owing to the resource heterogeneity, varied QoS constraints, and dynamic nature of the cloud, traditional list-based workflow scheduling algorithms can hardly be applied to the cloud environment without modification [13], [14]. Ambika *et al.* [13] resort to scheduling the multiworkflow in a cloud environment with multiple QoS parameters. The authors propose an enhanced metaheuristic optimization technique to maintain a tradeoff among these QoS parameters.

In regard to edge computing, the data transmission delay is further nonnegligible and is becoming the major bottleneck in optimizing the makespan of a workflow. The cost and makespan minimization problem of workflows in edge networks have been extensively studied in the recent years [7], [9], [15]–[17]. Stavrinides *et al.* [16] study the error propagation mechanism in the workflow in a fog computing environment: the authors emphasize that when imprecise evaluation of a task in the workflow exists, the error is likely to be propagated to the task's predecessor tasks and its descendants, thus resulting in error in predicting the actual makespan. The authors propose a partial computation and error propagation model and a dynamic scheduling heuristic to lower the deadline miss ratio. Liu *et al.* [38] developed a DAG scheduling algorithm in a homogeneous edge network. The scheduling decisions are made according to a dynamic-programming-based algorithm to minimize the makespan of a DAG. In the authors' work, tasks are placed on multiple edge servers repeatedly to find the optimal solution. Kanemitsu *et al.* [7] jointly optimize the energy consumption and the makespan of multiple workflows. In the authors' work, a pseudo task is added as the entry point of all workflows and the makespan is optimized in a centralized manner. Huang *et al.* [9] consider the security of the mobile edge network and schedule the tasks under the threat of malicious attacks. Here, the makespan and the encryption level are jointly optimized to achieve a tradeoff between efficiency and security. Xu *et al.* [15], focusing on workflow scheduling in the cloud-fog scenario, propose an extended PSO [18] algorithm by encoding each particle as a scheduling plan that consider the number of interdependent tasks in a workflow. An inertia weight updating mechanism is further designed to balance the global and local abilities of particles. In these works, the data transmission of a workflow is treated with the basic assumption that the network is fully connected. Thus, the communication cost is turned into an integer programming problem and is affected only by the placement of tasks.

However, things become more complicated when the network is partially connected. Edge servers that are not directly connected can be reached within several hops from multiple paths. Owing to the flexibility of the network topology, it is very likely that multiple routing paths exist as candidate transmission channels. Sun *et al.* [19] developed a workflow makespan minimization algorithm designed for partially connected networks with social groups where general edge servers belonging to different subnetworks can communicate only through joint servers. However, the authors' work ignores the data transmission on the edge network but focuses solely on the placement of tasks. With recent advancements in network management techniques such as network function virtualization (NFV) and software-defined networks (SDNs), multipath routing techniques have received growing attention both academically and industrially, not only because these techniques provide backup plans when link outages occur [20] but also because these techniques provide an emerging solution to improve network efficiency [21], [22]. In [23], an optimization algorithm was proposed to solve the task scheduling and multipath data stream routing of a workflow by using dynamic programming. However, the study failed to determine the dispatching order of the tasks, and ignoring the congestion of the same task during the multipath transmission further degraded the performance of the algorithm. In contrast to the aforementioned works, this paper emphasizes on solving the scheduling problem of tasks in the workflow and the multipath routing problem of the tasks' respective data streams by using deep reinforcement learning. In our work, the agent is trained to make decisions on task placement, routing path selection and data segmentation.

We hereby define the workflow embedding problem as follows. The workflow embedding on the edge is intended to dispatch the interdependent tasks over computing-enabled edge network infrastructure [24] and schedule their dataflow accordingly. An efficient embedding of workflow is considered to have a short makespan and to involve a reasonable usage of edge servers and network traffic [25].

In dealing with workflow embedding by multipath routing, the task scheduling order and the subpath selection are supposed to be determined beforehand. We propose a critical path analysis based dynamic task sorting algorithm to

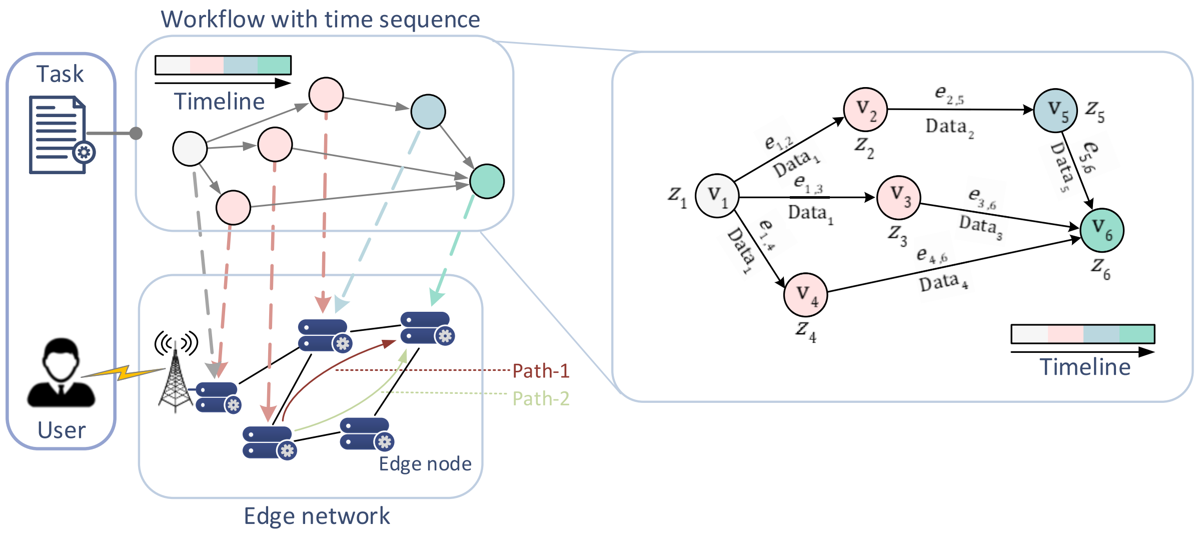
Workflow with time sequence
Task
Timeline
User
Path-1
Path-2
Edge node
Edge network
z_1 v_1 v_2 z_2 $e_{2,5}$ $Data_2$ v_5 z_5 $e_{1,2}$ $Data_1$ $e_{1,3}$ $Data_1$ v_3 z_3 $e_{3,6}$ $Data_3$ $e_{5,6}$ $Data_5$ v_6 z_6 $e_{1,4}$ $Data_1$ v_4 z_4 $e_{4,6}$ $Data_4$
Timeline

FIGURE 1. System model of workflow embedding.

determine the processing order of tasks. Here, the order of tasks is updated dynamically according to the actual finish time of the predecessor tasks. In regard to subpath selection, to avoid congestion of the data belonging to the same task, we introduce a path quality indicator to save the most valuable path when joint paths occur. The set of disjoint paths is then employed as the sub-paths to transmit the data in parallel. With this information clarified, the makespan optimization problem is then transformed into a Markov decision process (MDP). We propose a deep reinforcement learning-based scheme to handle the dispatch of tasks and the selection of subpaths.

In this paper, we propose a deep reinforcement learning based workflow embedding scheme named RLWE. Our primary objective is to train an agent that can schedule interdependent tasks of a workflow and coordinate the multipath data transmission in a partially connected edge network. Our contribution can be concluded as follows:

1) In this paper, we explore the workflow embedding problem in a partially connected edge network. We employ the multipath routing technique to improve the data stream transmission efficiency between interdependent tasks. A critical path analysis-based dynamic weighting scheme is introduced to keep track of the change in the critical path and to further instruct the multipath routing process. We further propose a path selection algorithm to determine the candidate routing paths.

2) We propose a deep reinforcement learning-based workflow embedding scheme that can coordinate the need for computing resources, routing paths, and data segmentation. By leveraging the multipath routing-based workflow embedding strategy, the proposed algorithm utilizes suboptimal paths for tasks with spare free floats and make way for tasks with a tighter budget.

3) We conduct extensive experiments to evaluate the performance of the proposed workflow embedding algorithm. Numerical results validate that the proposed RLWE can effectively reduce the makespan of a workflow by leveraging the multipath routing strategy.

The remainder of this paper is concluded as follows. Section II introduces the system model along with the formulation of the workflow embedding problem. The proposed workflow scheduling and data stream coordination is detailed in Section III. The numerical results are shown and discussed in Section IV. Finally, Section V concludes this paper.

II. SYSTEM MODEL OF WORKFLOW EMBEDDING

In this section, we present the system model shown in Fig. 1, where the workflow model and the edge network model are detailed and the workflow embedding problem is formulated.

A. WORKFLOW MODEL

The workflow is widely modeled as a directed acyclic graph (DAG); i.e., $G_{WF} \triangleq (\boldsymbol{V}, \boldsymbol{E})$, where $\boldsymbol{V} = \{v_1, \ldots, v_V\}$ is a set of interdependent tasks and $\boldsymbol{E} = \{e_{l,m} | \forall v_l, v_m \in \boldsymbol{V}\}$ is a set of edges that reflects data dependencies. For each task $v_l \in \boldsymbol{V}$, z_l denotes the required number of floating-point operations measured in FLOPs. An edge $e_{l,m}$ not only reflects the data dependency between two corresponding tasks v_l, v_m but also points in the direction of the data stream. $Data_{l,m}$ denotes the weight of an edge that reflects the data stream size from v_l to v_m and is measured in bits. Generally, for edge $e_{l,m}$, v_l is considered the predecessor task, as this task is the source of the data stream, while v_m denote the successor task. A successor task can be executed only after it receives the necessary data from all of its direct predecessor tasks. Here, two functions are defined for better illustration: $succ(v)$ denotes the set of the direct successor of task v, and $pred(v)$ denotes the set of task's direct predecessor tasks [12], [19].

Specifically, within a given workflow, the task with no predecessor is considered the entry task v_{entry}, while the task with no successor is called the exit task v_{exit}. For workflows with multiple-entry or multiple-exit tasks, a pseudo entry task v_{p-ent} and a pseudo exit task v_{p-exit} are added to the predecessor of v_{entry} and successor of v_{exit} respectively [19], [23], [26]. Since v_{p-ent} and v_{p-exit} are tasks with zero execution and transmission costs, this modification does not affect the scheduling process. In this paper, we also extend the interdependent task set $\boldsymbol{V}$ with the pseudo entry task and pseudo exit task, and the extended task set is denoted as $\boldsymbol{V'} = \boldsymbol{V} \cup \{v_{p-ent}, v_{p-exit}\}$. Such a method is also applied to extend the single-workflow algorithms to multi-workflow scenarios [7]; such an application is beyond our discussion.

B. NETWORK MODEL

Since the edge network under discussion is not fully connected, we model the partially connected network as an undirected graph $G_{Net} \triangleq (\boldsymbol{N}, \boldsymbol{\Phi})$ where $\boldsymbol{N} = \{n_1, \ldots, n_N\}$ denotes the set of edge servers and $\boldsymbol{\Phi} = \{\varphi_{i,j} | \forall n_i, n_j \in \boldsymbol{N}\}$ denotes a set of communication links that reflects the connectivity of servers. For each edge server $n_i \in \boldsymbol{N}$, f_i denotes the edge server's processing power measured in FLOP/s. For each link $\varphi_{i,j}$ where $n_i, n_j \in \boldsymbol{N}$, $B_{i,j}$ denotes the throughput in bits/s.

C. WORKFLOW EMBEDDING

In this paper, we refer to workflow scheduling and the multipath routing of the respective data stream as workflow embedding. The basic settings of the problem are detailed as follows.

$\mathcal{F}$ denotes the workflow scheduling function, and $\mathcal{F}_n^l$ indicates that task v_l is offloaded to edge server n. According to the workflow model G_{WF} and network model G_{Net}, the execution time $T^{exe}(\mathcal{F}_n^l)$ can be defined as Eq. (1).

$$T^{exe}\left(\mathcal{F}_n^l\right) = \frac{z_l}{f_n} \tag{1}$$

In special cases where $v_l = v_{p-ent}$ or $v_l = v_{p-exit}$, since the required computation resource of the task is 0, the execution time is also set as 0.

Even though the network is partially connected, multiple paths are likely to be found to reach the destination edge server through limited hops. However, in this paper, we roughly ignore the routing details and treat the subpaths as a whole. Denoting $\mathcal{P}_{i,j}$ as the set of paths from n_i to n_j. A subpath $p = \{\varphi_{i,x}, \ldots, \varphi_{y,j}\}$ is defined as a set of links arranged in the routing order. Obviously, when the data stream under transmission is divisible, we can split the data into segments and transmit them in parallel through multiple paths. There are already studies on router level multi-path routing techniques [27]–[29], and the advancement in segment routing further proves its practical feasibility [29]. Inconsistent with the basic assumptions in [23], we consider a scenario where task v_l is executed on n_i and one of the task's successor tasks v_m is scheduled to n_j, denoting $p \in \mathcal{P}_{i,j}$ as one of the subpaths, and α^p denotes the ratio of the respective segment. The segment of the data stream from task v_l to v_m in subpath p is hereby defined as Eq. (2).

$$seg_{l,m}^p = Data_{l,m} * \alpha^p \tag{2}$$

We define the equivalent bandwidth of a subpath B_{eq}^p as Eq. (3).

$$B_{eq}^p = 1 / \sum_{\varphi_{x,y} \in p_{i,j}} \left(\frac{1}{B_{x,y}}\right) \tag{3}$$

where x and y are the adjacent edge servers n_x and n_y, respectively, on the subpath; and where $\varphi_{x,y}$ represents the respective link.

Assuming a scheduled task v_l is executed on n_j with the task's predecessor task v_p executed on n_i, the transmission cost of the respective data stream $T^{trans}(\mathcal{F}_j^l)$ can be calculated as the maximum transmission time of the segments as shown in Eq. (4). Apparently, if the two tasks are scheduled on the same server, we have $p = \emptyset$. Specifically, when the scheduled task v_l is the pseudo entry task v_{p-ent}, the data in need of transmission are $Data_{p-ent,entry}$= 0, and the transmission time is 0. Similarly, when the predecessor task $pred(v_l) = v_{p-exit}$, its transmission time is also set as 0 for the same reason.

$$T^{trans}\left(\mathcal{F}_j^l\right) = \begin{cases} \max\limits_{\substack{\forall p \in \mathcal{P}_{i,j} \\ v_p \in pred(v_l)}} \left\lceil \frac{seg_{p,l}^p}{B_{eq}^p} \right\rceil, & if\ i \neq j \\ 0, & if\ i = j,\ v_l = v_{p-ent}\ or\ v_p = v_{p-exit} \end{cases} \tag{4}$$

The earliest finish time (FT) of task v_l is influenced by the earliest finish time of the task's predecessor tasks together with their respective transmission cost and the task's own execution time on the edge server.

$$FT(v_l) = \begin{cases} T^{exe}\left(\mathcal{F}_j^l\right), & if\ pred(v_l) = v_{entry} \\ \max_{\forall v_p \in \boldsymbol{pred}(v_l)} \left[FT\left(v_p\right) + T^{trans}(\mathcal{F}_j^l)\right] + T^{exe}\left(\mathcal{F}_j^l\right), & else \end{cases} \tag{5}$$

Hence, the makespan of the workflow is defined as the maximum earliest finish time of the extended task set $\boldsymbol{V'}$.

$$makespan = \max_{v_l \in \boldsymbol{V'}} FT(v_l). \tag{6}$$

D. PROBLEM DEFINITION

After all of the tasks in $\boldsymbol{V'}$ are scheduled, we can obtain the makespan of the workflow accordingly. The goal of this paper is to minimize the makespan of the workflow and the optimization problem can be formulated as follows:

$$\begin{aligned} \mathbf{P}: \min_{\mathcal{F}_n^v,\ p,\ \alpha^p} \quad & makespan \\ \text{s.t.} \quad & C1: 1 > \alpha^p > 0, \forall p \in \mathcal{P}_{i,j} \\ & C2: \sum_{p \in \mathcal{P}_{i,j}} \alpha^p = 1, \forall n_i, n_j \in \boldsymbol{N} \end{aligned} \tag{7}$$

where constraint $C1$ states that the ratio of a segment is between $[0, 1]$ and $C2$ ensures that all segments of a data stream are allocated to the respective subpath.

III. DEEP REINFORCEMENT LEARNING BASED WORKFLOW EMBEDDING

Apparently, the decision of task scheduling and data stream routing are closely coupled. To tackle the complicated NP-hard workflow embedding problem, we propose a deep reinforcement learning based scheme. Before tackling the intended workflow embedding problem, several preparation works of task scheduling order and subpath filtering should be made to improve the training efficiency.

A. CRITICAL PATH ANALYSIS BASED DYNAMIC TASK SORTING

Several algorithms are designed for the static scheduling of a workflow. List scheduling is considered the most classical approach. However, dynamic scheduling results may deviate from the original prediction, letting alone the transmission varies significantly with different routing subpath. However, the related concept can be absorbed to determine the task scheduling order.

The tasks of the workflow are first grouped based on the topology order. However, unlike the typical topological sorting algorithm, when multiple tasks with zero in-degree exist, these tasks will be stored as a set. Thus the result of such sorting is an ordered combination of task sets. Apparently, scheduling the tasks of the same set will not violate the dependency of tasks if all of the tasks in the former set are scheduled. However, the makespan can be further improved with the actual scheduling results.

To estimate the scheduling order of the tasks, the importance of each task will be evaluated statically. Let f_{avg} denote the average processing power of all edge servers and B_{avg} denote the average bandwidth between each link; then, the execution time and the single-path transmission time can be estimated as Eqs. (8) and (9) respectively.

$$\hat{T}^{exe}_{avg}(v_l) = \frac{z_l}{f_{avg}} \tag{8}$$

$$\hat{T}^{trans}_{avg}(v_l, v_m) = \frac{Data_{l,m}}{B_{avg}} \tag{9}$$

Then the estimated earliest start time and earliest finish time of v_l are defined as:

$$\widehat{ES}_l = \begin{cases} 0, & if\ pred(v_l) = \emptyset \\ \max\left[\widehat{EF}_p + \hat{T}^{trans}_{avg}(\boldsymbol{v_p}, v_l)\right], & otherwise \end{cases} \tag{10}$$

$$\widehat{EF}_l = \begin{cases} \hat{T}^{exe}_{avg}(v_l), & if\ pred(v_l) = \emptyset \\ \widehat{ES}_l + \hat{T}^{exe}_{avg}(v_l), & otherwise \end{cases} \tag{11}$$

where $\boldsymbol{v_p}$ denotes the set of predecessors of task v_l.

After all of the estimated earliest start and finish times are calculated, the estimated latest start and finish times can be derived as:

$$\widehat{LS}_l = \widehat{LF}_l - \hat{T}^{exe}_{avg}(v_l) \tag{12}$$

$$\widehat{LF}_l = \begin{cases} \widehat{EF}_l, & if\ succ(v_l) = \emptyset \\ min\left[\widehat{LS}_s - \hat{T}^{trans}_{avg}(v_l, \boldsymbol{v_s})\right], & otherwise \end{cases} \tag{13}$$

where $\boldsymbol{v_s}$ denotes the set of successors of task v_l.

Consequently, the free float, defined as the maximum delay before postponing the successor tasks, is calculated as Eq. (14). The total float [30], a crucial indicator of the maximum delay tolerance of a path is noted as Eq. (15)

$$\text{Free Float}(v_l) = \min\left(\widehat{ES}_s - \widehat{EF}_l\right), v_s \in \text{succ}(v_l) \tag{14}$$

$$\text{Total Float}(v_l) = \widehat{LF}_l - \widehat{EF}_l \tag{15}$$

Thus, the tasks in the same set can be further sorted according to the inverse order of their respective free float.

However, when the task is scheduled to an edge server, the execution time and data transmission time are changed accordingly. To track the influence of the scheduling consequence on actual execution time, we promptly update the free float and the order of tasks in the next batch by replacing the estimated time with the actual execution and transmission time.

A path of a DAG is often defined as a series of sequential nodes and edges that start with the entry node and sink to the exit node. Among these paths, only the one (or ones) with the longest execution and communication cost is defined as the critical path [25]. We can easily obtain the estimated critical path according to the total float of a path. Hence, some of the path's features can be utilized to improve the exploration efficiency of the proposed deep reinforcement learning scheme.

Proposition: Tasks on the same critical path are executed on the same edge server with the highest processing capability.

Proof: Let $T^*_{crit[entry,exit]}$ be the makespan of the critical path when all of the tasks on the critical path are executed on the edge server with the highest processing capability n_{max}. If there is an alternative decision to schedule task v_i, $v_i \in critical\ path$ and its successor tasks on n', the changed makespan is $T'_{critical} = T^*_{crit[entry,i]} + T^{trans}(\mathcal{F}^i_{n'}) + T'_{crit[i,exit]}$. If the makespan is reduced, then $T^{trans}(\mathcal{F}^i_{n'}) + T'_{crit[i,exit]} \leq T^*_{crit[i,exit]}$; because the transmission cost is $T^{trans}(\mathcal{F}^i_{n'}) > 0$, we have $T'_{crit[i,exit]} < T^*_{crit[i,exit]}$. This result conflicts with the condition that $f_{n_{max}} > f_{n'}$.Thus, the proposition stands. ■

Algorithm 1 shows the details of the proposed task sorting scheme.

B. PATH QUALITY-BASED ROUTING PATH SELECTION

When the placement of adjacent tasks is determined, the upcoming problem is to find proper routing paths for data stream transmission.

First, we use the RPF algorithm proposed in [23] to find all the possible paths from two edge servers. However, these paths are hardly to be disjoint, thus leading to the potential risk of congestion during the multipath routing phase. In this paper, we propose a path selection scheme and take path quality as the major indicator.

TABLE 1. Algorithm for dynamic task sorting.

```
Algorithm 1 Critical Path Analysis-Based Dynamic Task Sorting
Input: extended task set $\boldsymbol{V'}$, estimated condition of each task
$\widehat{ES}_v, \widehat{EF}_v, \widehat{LS}_v, \widehat{LF}_v$
1: Initialize the target task set $\boldsymbol{v}_{tgt} = [v_{psd-entry}]$ and sorting
   result $res = \emptyset$;
2: $res.append(\boldsymbol{v}_{tgt})$
3: While True:
4:   For $v_{curr}$ in $enumerate(\boldsymbol{v}_{tgt})$ do:
5:     Remove $v_{curr}$ from $\boldsymbol{v}_{tgt}$
6:     Remove the task $v_{curr}$ and its edges from $\boldsymbol{V'}$;
7:     If $\boldsymbol{V'} = \emptyset$:
8:       break
9:     For $v_i$ in $enumerate(\boldsymbol{V'})$:
10:      If $degree_{in}(v_i) = 0$ then:
11:        $\boldsymbol{v}'_{tgt}$.append($v_i$)
12:  Set $\boldsymbol{v}_{tgt} = \boldsymbol{v}'_{tgt}$
13:  Update result $res.append(\boldsymbol{v}_{tgt})$
14: Evaluate the free float and total float of the tasks according
    to Eqs. (8)-(15) and determine critical path
15: For task set $\boldsymbol{v}_{sub}$ in $enumerate(res)$:
16:   Sort the tasks in $\boldsymbol{v}_{sub}$ with the descending order of free
      float and rewrite $res$.
17: While True:
18:   Obtain the first element of $res$ and denote this element as $res[0]$
19:   While true:
20:     $v_{curr} = res[0].pop()$
21:     Schedule task $v_{curr}$ and respective data stream.
22:     If res[0].length = 0:
23:       Remove $res[0]$ from $res$
24:       Update the total float of all paths
25:       Update critical path
26:       If $res[1] = \emptyset$:
27:         break
28:       Else:
29:         Update the free float of $res[1]$ according to the
            scheduled results.
30:         Update the task order of $res[1]$
31:   If $res = \emptyset$:
32:     break
```

TABLE 2. Algorithm for routing path selection.

```
Algorithm 2 Path Quality-Based Routing Path Selection
Input: network model $G_{Net}$, edge server set $\boldsymbol{N}$
1: While True:
2:   $n_i = \boldsymbol{N}.pop()$
3:   For $n_j$ in $enumerate(\boldsymbol{N})$:
4:     Initialize the selected routing path set $\boldsymbol{P_{i,j}} = \emptyset$
5:     Obtain simple paths by using RPF [23]:
           $\boldsymbol{P}_{i,j}^{simp} = RPF(n_i, n_j)$
6:     Calculate path quality for paths in $\boldsymbol{P}_{i,j}^{simp}$ according to
       Eq. (16)
7:     While True:
8:       $idx_m = \underset{k}{\operatorname{argmax}}\{q_{i,j}(\wp_k)\}, \wp_k \in \boldsymbol{P}_{i,j}^{simp}$
9:       $\boldsymbol{P_{i,j}}.append(\wp_{idx_m})$
10:      Remove $\wp_{idx_m}$ from $\boldsymbol{P}_{i,j}^{simp}$
11:      Remove path with joint link $\varphi$ in $\wp_{idx_m}$ from $\boldsymbol{P}_{i,j}^{simp}$
12:      If $\boldsymbol{P}_{i,j}^{simp} = \emptyset$:
13:        break
14:  If $\boldsymbol{N}.length = 1$:
15:    break
16: return $\boldsymbol{P_{i,j}}$
```

A higher routing hop indicates that more links are included in the path, thus resulting in lower parallelism and a higher congestion probability. A lower bandwidth brings extra transmission costs. Thus, the path quality $q_{i,j}(\cdot)$ is designed as the reflection of the path length and equivalent bandwidth. Obviously, paths with a higher quality achieve a better tradeoff between throughput and link usage.

$$q_{i,j}(p) = \frac{1}{\sum_{\varphi_{x,y} \in p_{i,j}} \left(\frac{1}{B_{x,y}}\right) * \|p\|} \tag{16}$$

where $\|p\|$ denotes the number of links in path p.

For a data transmission task with multiple paths, the path with the best quality is considered as the major path and is treated as the benchmark to choose the rest of the disjoint paths. This process is executed recursively until we obtain all of the disjoint paths. The routing path selection is detailed in Algorithm 2.

C. DEEP DETERMINISTIC POLICY GRADIENT-BASED WORKFLOW EMBEDDING

In this paper, we consider a deep reinforcement learning approach to deal with the MDP featured optimization problem **P**. We construct a deep deterministic policy gradient (DDPG)-based workflow embedding scheme.

We believe deep reinforcement learning is an ideal approach that fits well in solving the proposed workflow embedding problem because the approach is model-free and relies lightly on accurate system models [27], thereby making it capable of dealing with systems with complex mechanisms and high dimensional behaviors.

A typical reinforcement learning architecture has basically three fundamental elements: state, action, and reward [31]. In this paper, these elements are defined as follows:

State Space: The state is defined as the reflection of the predecessor tasks $\boldsymbol{v_p}$and the current task v_l. ϱ_i^k denotes the in-degree of task v_k. Since the number of predecessor tasks varies with different tasks, to include the required information of the current task, the information of the predecessor task is defined as a $1 \times (\varrho_{in} + 1)$ vector, where $\varrho_{in} = \max_{v_k \in \boldsymbol{V}'}(\varrho_i^k)$ represents the maximum in-degree of all tasks. Thus, the predecessor information is defined as $I_{pred} = [(idx_{p_1}, EF_{p_1}, Data_{p_1,l})_1, \ldots, (idx_{p_{\varrho_i}}, EF_{p_{\varrho_i}}, Data_{p_{\varrho_i},l})_{\varrho_i}, idx_{\mathcal{C}}]$, were, $idx_{\mathcal{C}}$ is the index of the edge server; this index indicates where the last task on the critical path is scheduled. Specifically, if none of the predecessor tasks of the current task is on the critical path, $idx_{\mathcal{C}}$ is set to 0. Each tuple in I_{pred} corresponds to the information of a predecessor task; this information includes idx_l, which denotes the index of its location of the edge server; EF_l, which denotes the actual earliest finish time of the corresponding task; and $Data_l$, which denotes the size of data to be transmitted. Obviously, if the number of predecessor tasks of the current task is less than ϱ_i, the rest of the tuples are set as (0, 0, 0). The information on the current task is a combination of the required processing power z_l and a binary indicator $\mathcal{C}$ representing whether the current task belongs to the critical path. Thus the state space is a combination of predecessor and current task information and is given as $\boldsymbol{s} = [I_{pred}, z_l, \mathcal{C}]$.

Action Space: In this paper, the action consists of three parts, the placement of the current task, the selection of the routing paths from predecessor tasks, and the data segment ratio of subpaths. Denoting idx_l as the decision of placement of the current task; that is, the task is offloaded to n_{idx_l}. In regard to routing path and data segmentation, to make the output compatible with the input of predecessor tasks, we use the path selection result of algorithm 2. Let ξ denote the maximum number of sub-paths for all of the path sets in $\boldsymbol{P_{i,j}}$, then, the decision of the multipath routing phase is defined as a $\varrho_i \times \xi$ matrix and is given as $\Delta = [[\alpha_1^1, \ldots, \alpha_1^{\xi}]_1, \ldots, [\alpha_{\varrho_i}^1, \ldots, \alpha_{\varrho_i}^{\xi}]_{\varrho_i}]$. Each element in Δ is related to the predecessor in the corresponding position of the state. Thus, for tasks with an in-degree less than ϱ_i, the corresponding element in Δ is a full zero vector. Similarly, if the number of subpaths is less than ξ, zero padding is employed. Hence, the action is defined as $\boldsymbol{a} = [idx_l, \Delta]$.

Reward: During each step, a certain reward $r(\boldsymbol{s}, \boldsymbol{a})$ will be obtained under state $\boldsymbol{s}$ after jointly executing action $\boldsymbol{a}$. Since the objective of problem $\boldsymbol{P}$ is to minimize the makespan of the whole workflow while the goal of reinforcement learning lies in learning a policy for the agent that leads to maximum long-term reward, the reward should function as positively correlated with the actual improvement of the makespan. With $\widehat{EF_l}$ indicate the estimated earliest finish time of task v_l if all of the tasks are executed on a pseudo edge server with average processing ability and the data stream is transmitted on a pseudo link with the average bandwidth of the real network. The reward is defined as Eq. (17).

$$r_n = \begin{cases} e^{\left(\frac{\widehat{EF_l}}{EF_l}-1\right)}, & if\ EF_l < \widehat{EF_l} \\ -e^{\left(\frac{EF_l}{\widehat{EF_l}}-1\right)}, & otherwise \end{cases} \tag{17}$$

Notably, the order of the tasks mentioned in the state space follows the sorting result of Algorithm 1, Line 16. The scheduling and routing decision of the current task mostly affects the earliest finish time of the predecessor task of its successor tasks.

However, the number of possible state-action pairs is still unimaginably large, and the ratio of the data segment of subpaths is a continuous variable. It is impractical to enumerate all the possible conditions and store them in an immense Q-table. In this paper, we resort to using DDPG, a reinforcement learning scheme designed for continuous action.

The DDPG algorithm is based on the Q value and absorbs the experiences in actor-critic. Denoting μ as the current policy parameterized by θ^{μ}, the expected reward is defined as Eq. (18).

$$Q^{\mu}(s_t, a_t) = E\big[r(s_t, a_t) + \gamma Q^{\mu}\big(s_{t+1}, \mu(s_{t+1}, \theta^{\mu})\big)\big] \tag{18}$$

where s_{t+1} is the according state after executing action $\mu(s_t, \theta^{\mu})$, and γ is the discount factor ranging from [0, 1] to emphasize the influence of the long-term reward.

To evaluate the performance of the learned policy, a performance function $\mathcal{I}_{\beta}(\mu)$ is employed; this function is shown as Eq. (19).

$$\mathcal{I}_{\beta}(\mu) = E_{s\sim\rho^{\beta}}\big[Q^{\mu}(s, \mu(s))\big] \tag{19}$$

where ρ^{β} is defined as the probability density function of the state s.

Like most deep reinforcement learning frameworks, the goal of training is to find a policy μ to maximize $\mathcal{I}_{\beta}(\mu)$ while minimizing the loss function $\mathcal{L}$ [32].

$$\mathcal{L} = E_{s,a,r,s'}\left[\left(Q^{\mu}\left(s, a|\theta^{Q}\right) - y\right)^2\right] \tag{20}$$

After obtaining action from the target actor network μ', the reward obtained from the target network Q' can be derived as Eq. (21).

$$y = r + \gamma Q'\left(s_{t+1}, \mu'\left(s_{t+1}, \theta^{\mu'}\right)|\theta^{Q'}\right) \tag{21}$$

Hence, the policy gradient is calculated according to the sampled K experiences.

$$\nabla_{\theta^{\mu}}\mathcal{I}_{\beta}(\mu) = \frac{1}{K}\sum_i\left(\nabla_a Q\left(s, a|\theta^{Q}\right)|_{s=s_i, a=\mu(s_i)} * \nabla_{\theta^{\mu}}\mu\left(s|\theta^{\mu}\right)|_{s=s_i}\right) \tag{22}$$

Notably, the selection of routing paths is a major influence of transmission time and the earliest finish time of the corresponding task. In the context of multi-path routing, higher parallelism not only indicates less data transmission time, but also brings a latent risk of network congestion. Thus the network congestion level directly affects the designed reward. By leveraging deep reinforcement learning as an experience-driven congestion avoidance approach, the agent is expected to learn from the previously perceived history of feedback from the environment. Such history is considered to contain information about network utility and can be further exploited for routing path selection of the successor tasks [33].

Obviously, by modeling the workflow embedding operator as an agent, the proposed RLWE scheme still suffers from sparsity in experience because some of the edge servers have seldom been used. To handle this issue, we propose changing the exploration process by adding the OU noise after the action is taken according to the current policy. Thus, edge servers and subpaths that join the workflow less frequently are likely to be explored with probability ε. With the modified exploration process included, the proposed RLWE algorithm is supposed to be trained to explore and exploit with higher efficiency. Algorithm 3 shows the details of the proposed scheme.

By using the deep reinforcement learning-based workflow embedding scheme, the agent can coordinate the demand of computing resources and routing paths of interdependent tasks in a workflow.

TABLE 3. Deep Reinforcement Learning-Based Workflow Embedding

```
Algorithm 3 Deep Reinforcement Learning-Based Workflow Embedding
1:  Randomly initialize parameters of the actor network $\theta^{\mu}$ and critic
      network $\theta^{Q}$
2:  Initialize a random process $\mathcal{N}$ for action exploration
3:  Obtain routing paths according to Algorithm 2
4:  For episode in range(episodes):
5:    Initialize workflow ending indicator $\text{flag}_{exit} = 0$
6:    Receive initial state $s_1$
7:    Execute Algorithm 1, lines 1-14 to obtain the initial task
        Order $res$
8:    While True:
9:      For $v_{curr}$ in enumerate($res[0]$):
10:       If $v_{curr} = v_{psd-exit}$:
11:         $\text{flag}_{exit} = 1$
12:       Schedule $v_{curr}$ according to the current policy and
            add perturbation $\mathcal{N}$ with probability $\varepsilon$
13:       Store the transition$\{s_t, a_t, r_t, s_{t+1}, \text{flag}_{exit}\}$ in
            replay buffer $\mathcal{R}$
14:       If $\mathcal{R}$.length() $> threshold$:
15:         Sample a random mini-batch of $K$ transitions
16:         Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s_{i+1}, \theta^{\mu'})|\theta^{Q'})$
17:         Update critic by minimizing the loss
              $\mathcal{L} = E_{s,a,r,s'}[(Q^{\mu}(s,a|\theta^{Q}) - y)^2]$
18:         Update the actor policy by using the sampled policy
            gradient:
              $\nabla_{\theta^{\mu}} \mathcal{J}_{\beta}(\mu) = \frac{1}{K}\sum_i (\nabla_a Q(s,a|\theta^{Q})|_{s=s_i,a=\mu(s_i)} * \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s=s_i})$
19:         Update the target networks:
              $\theta^{\mu'} \leftarrow \eta\theta^{\mu} + (1-\eta)\theta^{\mu'}$
              $\theta^{Q'} \leftarrow \eta\theta^{Q} + (1-\eta)\theta^{Q'}$
20:     Update the task execution order of the next task batch
        according to Algorithm 1, lines 24-30
21:     Remove $res[0]$ from $res$
22:     If $res = \emptyset$:
23:       Break
```

IV. NUMERICAL RESULTS

In this section, we conduct a series of simulations to validate the feasibility of the proposed RLWE scheme.

A. EXPERIMENTAL SETUP

Inconsistent with the network setup in [23], in our simulation, we set 10 edge servers in the fully connected edge network with their respective processing power randomly sampled at [20, 40] Gflop/s, while the bandwidth is randomly sampled at [30, 80] Mbit/s as the benchmark network. All of the simulations are performed on a desktop with an I7-7700K CPU operating at 4.2GHz and 16GB memory. The proposed algorithms are implemented by Python 3.6 and TensorFlow 1.14.

B. PERFORMANCE EVALUATION

1) VERIFICATION OF SUBPATH SELECTION

To evaluate the feasibility of the proposed subpath selection algorithm, we create a new edge network based on the benchmark network with links of the edge servers randomly eliminated but ensure that the servers can communicate with each other within limited hops. We create four groups of workflows, each consecutively consisting of 10, 40, 60, and 120 task nodes. For each group, we randomly generate 50 workflows and calculate the average makespan of each subpath selection strategy. The required computational resources are uniformly distributed between [2, 6] gigaflops while the data under transmission are scaled to [5, 10] Mbits.

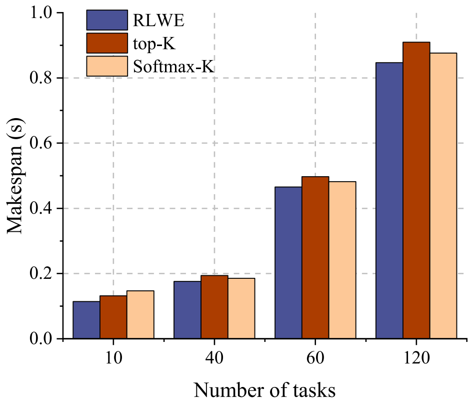

FIGURE 2. Average completion time of three different subpath selection strategies.

We consider the following two multipath routing rules as the baseline.

Top-k: Select the top $k = \xi/3$ subpaths in terms of their equivalent bandwidth.

Softmax-k [34]: Compute the softmax values $\sigma_i = \frac{e^{\omega_i}}{\sum_{j=1}^{\xi} e^{\omega_j}}$ for each subpath j; then, randomly choose $k = \xi/3$ subpaths according to the probability distribution $[\sigma_i(\omega)]_{i=1}^{\xi}$.

Fig. 2 is an illustration of the simulation result. All three agents are trained with the learning rate set as cosine annealing with warm restarts from 10^{-3} to 10^{-6}, and their upper bounds decrease exponentially.

Fig. 2 shows that the proposed RLWE outperforms the top-K and Softmax-K methods over all four batches of workflows, with average improvements of 8.97% and 8.48% respectively. Such improvement over the baseline increases when the workflow scales up. This is because the RLWE can effectively coordinate the routing paths when multiple tasks are scheduled, thus minimizing the influence of congestions during multipath data transmission. Notably, the top-K outperforms Softmax-K on the batch where the number of tasks is 10 but is at a disadvantage on the rest of the batches. This finding can also be analyzed from the data transmission perspective. With fewer tasks included, it is less likely to route data simultaneously, in such circumstances, selecting paths with the best quality is obviously a better choice compared with random selection. However, when the workflow scales up, the path selection strategy with perturbation is an effective way of avoiding the congestion.

2) PERFORMANCE EVALUATION OF THE PROPOSED WORKFLOW EMBEDDING ALGORITHM

We further evaluate the proposed RLWE algorithm by using real-world workflows. Three different structures of the workflow named CyberShake, Montage, and EpiGenomics are

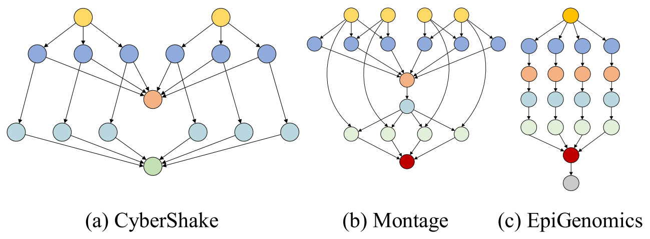

FIGURE 3. Overview of three workflow structures.

TABLE 4. Details of the benchmark workflows.

Workflows	# of Tasks	# of Edges	Avg. Data Size
CyberShake 50	50	188	864.74MB
Montage 50	50	206	3.36MB
EpiGenomics 46	46	148	104.81MB

implemented in our simulation. All of these workflows are extracted from real-world scientific applications [35]–[37]. CyberShake is a data-intensive application used to characterize earthquake hazards by generating probabilistic seismic hazard curves for a certain area. Montage is an input/output (I/O)-intensive application designed to generate custom mosaics of the sky by using input images in the FITS format. EpiGenomics is a CPU-intensive workflow for mapping the epigenetic state of human cells on a genome-wide scale. All of these applications are provided in the XML format and can be converted to the DAG-formed workflow with the help of workflow management system framework tools such as Pegasus [35], [37]. Fig. 3 and Table 4 are the illustration of these workflows and details of the workflow implemented in our simulation respectively.

To verify the effectiveness of our proposed RLWE scheme, we choose the following algorithms as the baseline: two single-path workflow scheduling algorithms named FixDoc and HEFT and a multipath workflow embedding algorithm named DPE.

DPE [23]: A workflow embedding algorithm designed for a partially connected edge network. Tasks with the same predecessor tasks are scheduled in random order, and all of the possible paths (noted as simple paths) are utilized for data transmission. The placement of tasks and the splitting ratio are determined by employing dynamic programming and matrix analysis.

FixDoc [38]: A DAG scheduling algorithm with fixed edge server configuration. Tasks are offloaded to homogeneous edge servers according to a dynamic-programming based algorithm to minimize the makespan of a DAG. Here, tasks are likely to be placed on multiple edge servers repeatedly to find the optimal solution.

HEFT (Heterogeneous Earliest-Finish-Time) [26]: A classic algorithm based on topological sorting of workflow. The average execution time of each task and the average communication time are used to estimate the earliest finish time of the tasks. Task priorities are further allocated statically according to the estimated earliest finish time and are assigned to the servers heuristically.

1) Analysis on different network connectivity: Four new networks are created based on benchmark networks with 25%, 50%, 75%, and 100% of their links reserved. The simulation result is shown in Fig. 4.

In Fig. 4, we find that the makespan of the workflows is negatively related to the connectivity of the network. This can be explained from the data transmission perspective. With the increment of the network connectivity, more routing subpaths are available during the multipath routing phase, thereby extending the throughput of the links and further avoiding the possibility of congestion. Notably, FixDoc and HEFT are single-path routing algorithms; thus, they benefit less when the connectivity of the network is above 50%. DPE and the proposed RLWE share a similar trend when the network connectivity increases because both of them can use multiple subpaths to further decrease the transmission time of the data stream. However, the advantage of RLWE over DPE lies in two parts, the task scheduling order and the congestion avoidance mechanism.

Concerning the performance evaluation on different characteristics of the workflows, as a data-intensive application, CyberShake (Fig. 4-a) is more sensitive to the change in connectivity of the network, while the I/O-intensive Montage (Fig. 4-b) and CPU-intensive EpiGenomics (Fig. 4-c) change slightly. These differences explain why the improvement of RLWE with the increment of the network connectivity is higher on CyberShake than that on Montage and EpiGenomics.

2) Analysis on the different number of edge servers: To evaluate the performance of the RLWE when the network scales up, workflows are implemented on edge networks consisting of 4, 7, 10, and 13 edge servers with 50% of their links reserved. The simulation result is shown in Fig. 5.

It is observable that for workflows with higher parallelism (CyberShake in Fig. 5-(a) and Montage in Fig. 5-(b)), the execution time becomes the major bottleneck. The makespan of both single-path and multipath algorithms are high. All of the algorithms share a similar decreasing trend when the network scales up. This is because, with more edge servers available, more tasks can be handled simultaneously. Moreover, an increasing number of edge servers leads to higher connectivity of the network, thus providing potential subpaths with higher efficiency. This explains the reason why multipath routing algorithms DPE and RLWE benefit more than single path algorithms. The results show that the proposed RLWE overwhelms the baseline in terms of makespan.

3) Analysis of different bandwidths: We further evaluate the effectiveness of the proposed approach on networks with different bandwidths. We change the average bandwidth as 35, 45, 55, and 65 Mbits/s respectively.

Fig. 6 reveals the influence of the bandwidth. An increasing bandwidth of links will save more transmission time, and the makespan of all algorithms is reduced consequently. Obviously, single-path algorithms FixDoc and HEFT benefit

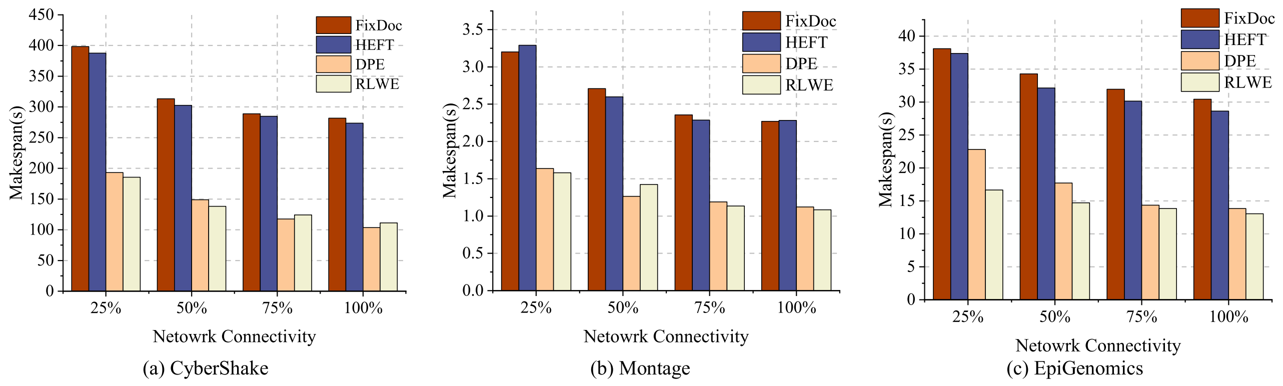

FIGURE 4. Makespan of workflows under different network connectivities. (a) CyberShake (b) Montage (c) EpiGenomics.

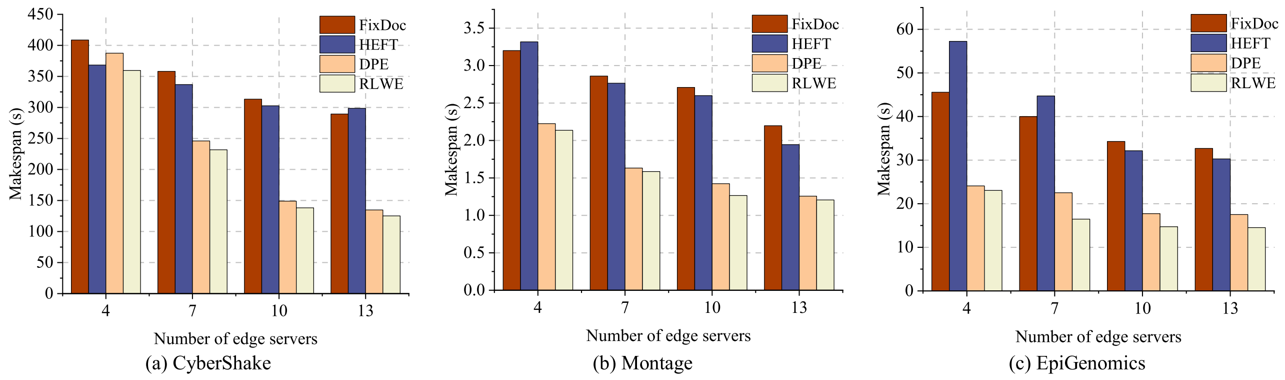

FIGURE 5. Makespan of workflows under different number of edge servers. (a) CyberShake (b) Montage (c) EpiGenomics.

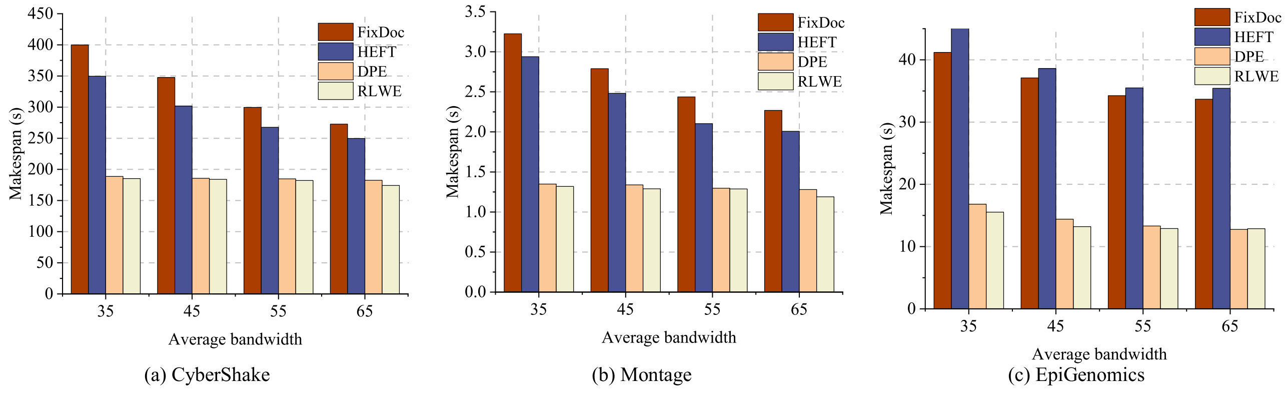

FIGURE 6. Makespan of workflows under different bandwidths. (a) CyberShake (b) Montage (c) EpiGenomics.

more than those multipath-based algorithms. Nonetheless, the proposed RLWE outperforms all the baselines.

V. CONCLUSION

This paper focuses on the workflow embedding problem with interdependent task scheduling and multipath routing in a partially connected edge network. We propose a free-float based topology sorting method to determine the scheduling order of tasks and a filtering algorithm for selecting routing paths between edge servers. We further propose a deep reinforcement learning-based workflow embedding algorithm RLWE to solve the problem. Experimental results on real-world workflows show that our proposed RLWE can coordinate the data stream of interdependent tasks and can effectively reduce the makespan of the workflow through multipath routing. The result also reflects that the RLWE outperforms baselines, i.e., DPE, FixDoc and HEFT in terms of makespan reduction.

REFERENCES

[1] L. Wang, L. Jiao, T. He, J. Li, and M. Mühlhäuser, "Service entity placement for social virtual reality applications in edge computing," in *Proc. IEEE INFOCOM*, 2018, pp. 468–476.

[2] F. Wang *et al.*, "Intelligent edge-assisted crowdcast with deep reinforcement learning for personalized QoE," in *Proc. IEEE INFOCOM*, 2019, pp. 910–918.

[3] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things: A primer," *Digit. Commun. Netw.*, vol. 4, no. 2, pp. 77–86, 2018.

[4] M. T. Beck, M. Feld, S. Werner, and T. Schimper, "Mobile edge computing: A taxonomy," in *Proc. 6th Int. Conf. Adv. Future Internet*, 2014, pp. 48–55.

[5] J. Xu and S. Ren, "Online learning for offloading and autoscaling in renewable-powered mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2016, pp. 1–6.

[6] T. Dhand, *A Unified Framework for Service Availability and Workflow Scheduling in Edge Computing Environment*, Thapar Univ., Patiala, Punjab, 2017.

[7] H. Kanemitsu, M. Hanada, and H. Nakazato, "Multiple workflow scheduling with offloading tasks to edge cloud," in *Proc. Int. Conf. Cloud Comput.*, 2019, pp. 38–52.

[8] Y. Zhang, Z. Zhou, Z. Shi, L. Meng, and Z. Zhang, "Online scheduling optimization for DAG-based requests through reinforcement learning in collaboration edge networks," *IEEE Access*, vol. 8, pp. 72985–72996, 2020.

[9] B. Huang, Y. Xiang, D. Yu, J. Wang, Z. Li, and S. Wang, "Reinforcement learning for security-aware workflow application scheduling in mobile edge computing," *Security Commun. Netw.*, vol. 2021, May 2021, Art. no. 5532410. [Online]. Available: https://doi.org/10.1155/2021/5532410

[10] Y.-S. Chang, C.-T. Fan, R.-K. Sheu, S.-R. Jhu, and S.-M. Yuan, "An agent-based workflow scheduling mechanism with deadline constraint on hybrid cloud environment," *Int. J. Commun. Syst.*, vol. 31, no. 1, 2018, Art. no. e3401.

[11] R. N. Calheiros and R. Buyya, "Meeting deadlines of scientific workflows in public clouds with tasks replication," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 7, pp. 1787–1796, Jul. 2014.

[12] Y. Qin, H. Wang, S. Yi, X. Li, and L. Zhai "An energy-aware scheduling algorithm for budget-constrained scientific workflows based on multi-objective reinforcement learning," *J. Supercomput.*, vol. 76, no. 1, pp. 455–480, 2020.

[13] A. Aggarwal, P. Dimri, A. Agarwal, M. Verma, H. A. Alhumyani, and M. Masud, "IFFO: An improved fruit fly optimization algorithm for multiple workflow scheduling minimizing cost and makespan in cloud computing environments," *Math. Problems Eng.*, vol. 3, pp. 1–9, Jun. 2021.

[14] A. Aggarwal, P. Dimri, and A. Agarwal, "Survey on scheduling algorithms for multiple workflows in cloud computing environment," *Int. J. Comput. Sci. Eng.*, vol. 7, no. 6, pp. 565–570, 2019.

[15] R. Xu *et al.*, "Improved particle swarm optimization based workflow scheduling in cloud-fog environment," in *Proc. Int. Conf. Bus. Process Manage.*, 2018, pp. 337–347.

[16] G. L. Stavrinides and H. D. Karatza, "Orchestrating real-time IoT workflows in a fog computing environment utilizing partial computations with end-to-end error propagation," *Clust. Comput.*, vol. 24, pp. 3629–3650, Jul. 2021.

[17] K. Matrouk and K. Alatoun, "Scheduling algorithms in fog computing: A survey," *Int. J. Netw. Distrib. Comput.*, vol. 9, no. 1, pp. 59–74, 2021.

[18] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proc. Int. Conf. Neural Netw.*, 1995, pp. 1942–1948.

[19] J. Sun, L. Yin, M. Zou, Y. Zhang, T. Zhang, and J. Zhou, "Makespan-minimization workflow scheduling for complex networks with social groups in edge computing," *J. Syst. Archit.*, vol. 108, Sep. 2020, Art. no. 101799. [Online]. Available: https://doi.org/10.1016/j.sysarc.2020.101799

[20] F. Aubry, S. Vissicchio, O. Bonaventure, and Y. Deville, "Robustly disjoint paths with segment routing," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol.*, 2018, pp. 204–216.

[21] Q. Wang, J. Xue, G. Shou, Y. Liu, Y. Hu, and Z. Guo, "Implementation of multipath network virtualization scheme with SDN and NFV," in *Proc. IEEE 28th Annu. Int. Symp. Personal Indoor Mobile Radio Commun. (PIMRC)*, 2017, pp. 1–6.

[22] T. Pham and L. M. Pham, "Load balancing using multipath routing in network functions virtualization," in *Proc. IEEE Int. Conf. Comput. Commun. Technol. Res. Innov. Vis. Future (RIVF)*, 2016, pp. 85–90.

[23] H. Zhao, S. Deng, Z. Liu, Z. Xiang, and J. Yin, "Placement is not enough: Embedding with proactive stream mapping on the heterogenous edge," 2020, *arXiv:2012.04158*.

[24] M. Melnik and D. Nasonov, "Workflow scheduling using neural networks and reinforcement learning," *Procedia Comput. Sci.*, vol. 156, pp. 29–36, Jan. 2019.

[25] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 506–521, May 1996.

[26] H. Topcuoglu, S. Hariri, and M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 260–274, Mar. 2002.

[27] Z. Xu *et al.*, "Experience-driven networking: A deep reinforcement learning based approach," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, 2018, pp. 1871–1879.

[28] T. Braud, P. Zhou, J. Kangasharju, and P. Hui, "Multipath computation offloading for mobile augmented reality," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, 2020, pp. 1–10.

[29] Y. Tian *et al.*, "Traffic engineering in partially deployed segment routing over IPv6 network with deep reinforcement learning," *IEEE/ACM Trans. Netw.*, vol. 28, no. 4, pp. 1573–1586, Aug. 2020.

[30] Y. Y. Yuan, X. Li, Q. Wang, and X. Zhu, "Deadline division-based heuristic for cost optimization in workflow scheduling," *Inf. Sci.*, vol. 179, no. 15, pp. 2562–2575, 2009.

[31] S. Thrun and M. L. Littman, "Reinforcement learning: An introduction," *AI Mag.*, vol. 21, no. 1, p. 103, 2000.

[32] K. Zhu, Z. Zhang, and M. Zhao, "Auxiliary-task based resource orchestration in mobile edge computing," *IEEE Trans. Green Commun. Netw.*, to be published.

[33] N. Jay, N. H. Rotman, P. B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on Internet congestion control," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 1–10.

[34] D. Kim *et al.*, "Learning to schedule communication in multi-agent reinforcement learning," 2019, *arXiv:1902.01554*.

[35] N. Anwar and H. Deng, "A hybrid metaheuristic for multi-objective scientific workflow scheduling in a cloud environment," *Appl. Sci.*, vol. 8, no. 4, p. 538, 2018.

[36] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientific workflows," *Future Gener. Comput. Syst.*, vol. 29, no. 3, pp. 682–692, 2013.

[37] E. Deelman *et al.*, "Pegasus, a workflow management system for science automation," *Future Gener. Comput. Syst.*, vol. 46, pp. 17–35, May 2015.

[38] L. Liu, H. Tan, S. H. Jiang, Z. Han, X.-Y. Li, and H. Huang, "Dependent task placement and scheduling with function configuration in edge computing," in *Proc. Conf. IEEE/ACM 27th Int. Symp. Qual. Service (IWQoS)*, 2019, pp. 1–10.

KAIGE ZHU received the M.S. degree in electrical and communication engineering from Beijing Jiaotong University, Beijing, China, in 2016, where he is currently pursuing the Ph.D. degree with the School of Electronic and Information Engineering. His research interests include edge computing and machine learning techniques.

ZHENJIANG ZHANG received the Ph.D. degree in communication and information systems from Beijing Jiaotong University (BJTU), where he was an Associate Professor from 2008 to 2013 and has been a Professor since 2013. He currently serves as the Director of the Institute of Intelligent Network and Information Security in BJTU. His research interests include cognitive radio, wireless sensor networks, and edge computing. He has been the Guest Editor of a number of journals, including IET COMMUNICATIONS, *Sensors*, and *International Journal of Distributed Sensor Networks*.

FENG SUN received the bachelor's degree from the School of Electronic and Information Engineering, Beijing Jiaotong University in 2017, where he is currently pursuing the Ph.D. degree in communication engineering. His research interests include edge computing and vehicular network.

BO SHEN received the B.S. degree in communication and control engineering from Northern Jiaotong University, Beijing, China, in 1995, and the Ph.D. degree in communication and information system from Beijing Jiaotong University, Beijing, in 2006. From 2006 to 2007, he was a Postdoctoral Researcher with the Institute of System Science, Beijing Jiaotong University, where he has been a Researcher and a Teacher with the School of Electronic and Information Engineering, Beijing Jiaotong University since 2008. He is currently a Professor with the School of Electronic and Information Engineering, Beijing Jiaotong University and the Director of the Key Laboratory of Communication and Information Systems. His research interests include complex system theory, opinion evolution, data mining, and NLP techniques.