Temporal Fusion Pointer network-based Reinforcement Learning algorithm for Multi-Objective Workflow Scheduling in the cloud

1st Binyang Wang
Laboratory of intelligent information processing and control
Beijing Institute of Technology
Beijing, China
wbybit@163.com

2nd Huifang Li*
Laboratory of intelligent information processing and control
Beijing Institute of Technology
Beijing, China
huifang@bit.edu.cn

3rd Zhiwei Lin
Laboratory of intelligent information processing and control
Beijing Institute of Technology
Beijing, China
linzhiwei@bit.edu.cn

4th Yuanqing Xia
Laboratory of intelligent information processing and control
Beijing Institute of Technology
Beijing, China
xia_yuanqing@bit.edu.cn

***Abstract*—Cloud computing is emerging as a deployment promising environment for hosting exponentially increasing scientific and social media applications, but how to manage and execute these applications efficiently depends mainly on workflow scheduling. However, scheduling workflows in the cloud is an NP-hard problem and its existing solutions have certain limitations when applied to real-world scenarios. In this paper, a Temporal Fusion Pointer network-based Reinforcement Learning algorithm for multi-objective workflow scheduling (TFP-RL) is proposed. Through adopting reinforcement learning, our algorithm can discover its heuristics over time by continuous learning according to the rewards resulting from good scheduling solutions. To make more comprehensive scheduling decisions as the influence of historical actions, a novel temporal fusion pointer network (TFP) is designed for the reinforcement learning agent, which can improve the quality of our resulting solutions and the ability of our algorithm in dealing with versatile workflow applications. To decrease convergence time, we train the proposed TFP-RL model independently by the Asynchronous Advantage Actor-Critic method and use its resulting model for scheduling workflows. Finally, under a multi-agent reinforcement learning framework, a Pareto dominance-oriented criterion for reasonable action selection is established for a multi-objective optimization scenario. We first train our TFP-RL model by taking randomly generated workflows as inputs to validate its effectiveness in scheduling , then compare our trained model with other existing scheduling approaches through practical compute- and data-intensive workflows. Experimental results demonstrate that our proposed algorithm outperforms the benchmarking ones in terms of different metrics.**

***Index Terms*—Multi-objective workflow scheduling, Reinforcement Learning, Cloud computing, Neural networks**

This work was supported by the National Key Research and Development Program of China under grant 2018YFB1003700.

*Corresponding author. E-mail address:huifang@bit.edu.cn(Huifang Li).

I. Introduction

The amount of data is increasing exponentially with more and more scientific as well as social media applications. To store, access, analyze and process such a great amount of data, cloud computing is explored, which provides users with types of computational services by the means of internet [1]. The performance of cloud systems is mainly influenced by task scheduling and resource allocation. Any kind of computational applications, such as data calculation and analysis, can be described by workflows. Therefore, cloud workflow scheduling becomes an important issue needed to be solved. However, it faces great challenges: First, it is widely acknowledged that scheduling workflows on distributed platforms is an NP-hard problem [2]; Second, there are multiple objectives to be optimized when scheduling, such as minimizing makespan and maximizing resource utilization; Third, the elasticity and heterogeneity of cloud resources make it more complex.

Plenty of heuristic and meta-heuristic algorithms are proposed to produce approximate optimal solutions [3]. However, heuristics are always restricted by problem statements while meta-heuristics own high computational cost due to its large number of iterations during evolutionary processes. Moreover, scheduling should be dynamic, but heuristics and meta-heuristics can only make assignments of tasks from a given workflow [4]. As machine learning algorithms are becoming increasingly versatile and powerful, reinforcement learning (RL) is discovered and adopted into scheduling. It is a kind of novel method where agents can continuously learn to promote its scheduling performance concerning to defined objectives. But the generalization and solution quality of current RL based

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

scheduling approaches still have great promotion space.

To tackle the issues mentioned above, in this paper, we proposed a temporal fusion pointer network-based reinforcement learning algorithm for multi-objective workflow scheduling (TFP-RL). The main contributions can be demonstrated as follows: First, by exploring cloud workflow scheduling mechanisms from an RL perspective, our method can discover its own heuristics by continuous learning according to the rewards received from good scheduling decisions over time. Second, based on the pointer network (Ptr-net), a novel temporal fusion pointer (TFP) network is developed for the RL agent, which can help our algorithm make more informed scheduling decisions under considering the historical effects, thus improving its solution quality and ability to deal with versatile workflow applications. Third, our model is trained independently adopting the Asynchronous Advantage Actor-Critic (A3C) method, and then the resulting model is applied to schedule workflows, which can dramatically improve the time efficiency of our algorithm; Finally, a Pareto dominance-oriented criterion is established for selecting reasonable actions under multi-objective optimization situations.

The rest of the content is organized as follows. Section II outlines the formulation of the cloud resource, workflow and scheduling problems. Section III demonstrates the TFP-RL approach in detail, including its agent network design and training methods. Section IV displays the analysis and discussion about experimental results. Section V investigates related researches in recent years. Finally, Section VI concludes the paper and discusses some future work.

II. Scheduling Problem Formulation

Cloud Service Providers (CSPs) offer users with computational services through virtual machines (VMs) over internet [5]. Thus, we define a resource set $R = \{r_0, r_1, r_2, \cdots, r_{|R|}\}$ to represent available VMs that can be availed to meet the client requirements. Besides, the CSP provides VMs with different performances like CPU computational capacity and bandwidth, which can be denoted by $cu(r_f)$ and $bw(r_f)$ respectively. In addition, there are different kinds of pricing models applied by CSPs, such as charging all partial hours in Amazon EC2 [6] and counting used minutes in Microsoft Azure [7]. Here, we calculate the cost of leasing r_f as follow:

$$Cost(r_f) = \mu(r_f) \times [LDT(r_f)/tu] \tag{1}$$

where $\mu(r_f)$ is the price of leasing r_f per unit time, $LDT(r_f)$ is the leasing duration of r_f and tu represents the minimum leasing time duration.

A workflow is always depicted as a Directed Acyclic Graph (DAG) where nodes represent tasks $T = \{t_0, t_1, t_2, \cdots, t_{|T|}\}$ and edges represent data and control dependencies $D = \{(t_i, t_j) \,|t_i, t_j \in T, i \neq j \ or \ i < j\}$ between tasks [8]. An example of a workflow model is given in Fig. 1.

Based on the cloud resource and workflow models mentioned above, the execution time (ET) of task t_i can be calculated as:

$$ET(t_i) = \frac{fpo(t_i)}{cu(r_f)} \tag{2}$$

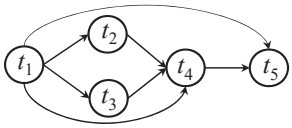


Fig. 1: A DAG sample with five tasks.

where $fpo\,(t_i)$ represents the computational size of t_i and r_f is the VM for executing t_i. The transmission time (TT) between two tasks t_i and t_j can be computed as:

$$TT(t_i, t_j) = \begin{cases} \frac{dataSize(t_i,t_j)}{\min\{bw(r_f),bw(r_g)\}} & (r_f \neq r_g) \\ 0 & (r_f = r_g) \end{cases} \tag{3}$$

where $dataSize(t_i, t_j)$ is the size of data to be transferred, while r_f and r_g are the VMs for executing t_i and t_j respectively.

Definition 1: Suppose we have n_{wa} workflow applications (modeled as DAGs) and a cloud data center consisting of n_c VMs. The goal of multi-objective workflow scheduling is to find a solution which can assign all tasks to VMs such that both makespan and economic cost of the whole workflow applications are minimized. It can be formulated as:

$$Minimize \quad F = (Makespan, Cost)$$

To estimate the makespan, we first define the start time (ST) and finish time (FT) for each task t_i as:

$$ST(t_i) = \max \left\{ \begin{array}{l} avail[r_f], \\ \max\limits_{t_j \in parent(t_i)} \left\{ \begin{array}{l} FT(t_j) + \\ TT(t_j, t_i) \end{array} \right\} \end{array} \right\} \tag{4}$$

$$FT(t_i) = ST(t_i) + ET(t_i). \tag{5}$$

where $avail[r_f]$ is the available or ready time of VM r_f on which task t_i is scheduled to execute. Note that when $t_i = t_{entry}$) (entry task t_{entry} is a task without any predecessors), $ST(t_{entry})$ = 0. Otherwise, $ST(t_i)$ can be computed as Eq. 4. Then, makespan can be obtained as:

$$Makespan = \max_{t_i \in T} \{FT(t_i)\} \tag{6}$$

For cost, we define the start lease time (SLT), finish lease time (FLT) and lease duration (LD) of r_f as:

$$SLT(r_f, t_i) = \begin{cases} 0 & (t_i = t_{entry}) \\ \min\limits_{t_j \in parent(t_i)} \{FT(t_j)\} & (t_i \neq t_{entry}) \end{cases} \tag{7}$$

$$FLT(r_f, t_i) = \begin{cases} FT(t_i) & (t_i = t_{exit}) \\ FT(t_i) + \max\limits_{t'_j \in children(t_i)} \{TT(t_i, t'_j)\} & (t_i \neq t_{exit}) \end{cases} \tag{8}$$

$$LD(r_f, t_i) = FLT(r_f, t_i) - SLT(r_f, t_i) \tag{9}$$

where t_{exit} denotes a task without any successors. Then, the accumulated total lease time (TLT) of r_f and the whole

workflow execution cost under a specific scheduling scheme can be computed as:

$$TLT(r_f) = \sum_{t_i \in T} LD(r_f, t_i) \tag{10}$$

$$Cost = \sum_{r_f \in R} Cost(r_f) = \sum_{r_f \in R} \mu(r_f) \times [TLT(r_f)/tu] \tag{11}$$

III. TFP-RL

A. Reinforcement learning framework

To handle the above formulated problems, we choose RL based structure and make several novel changes so as to get better performance. The basic framework of RL usually consists of two parts: an agent and environment [9]. The interaction between them can be depicted as follows: Through observations, the agent obtains current state s containing environmental information and takes an action a. Influence by a, the environment changes its state which needs to be re-observed. To solve multi-objective optimization problems, in this paper, we apply a multi-agent RL architecture shown as Fig. 2. There are two sub-agents and each of them is used to optimize one objective, i.e., makespan and cost respectively. RL problems are always modeled as Markov Decision Processes (MDPs) which can be defined as:

Definition 2: A Markov Decision Process is a tuple $<S, A, P, r, \gamma>$ [10] where:

- S is a finite set of states s
- A is a finite set of actions a
- P is a state transition probability matrix, $P^a_{ss'} = P[S_{t+1} = s' | S_t = s,\ A_t = a]$
- r is a reward function, $r^a_s = E[r_{t+1} | S_t = s,\ A_t = a]$
- γ is a discount factor, $\gamma \in [0, 1]$

In this study, a state at time step t consists of the features for alternative scheduling assignments at time step t, which can be depicted as:

$$s_t = (p_1, p_2, .., p_i, ., p_{n_t}) \tag{12}$$

where n_t is the size of s_t, representing the number of alternative scheduling schemes in the current scheduling stage. For time and cost sub-agent, each element p_i of s_t is a characteristic value of the scheduling scheme i, and its value equals corresponding time and cost resulting from the schemes i respectively. For instance, given that there is one task to be executed and two VMs, $n_t = 2$. p_1 and p_2 in s_t for time sub-agent represent current makespan if the task is executed on VM1 and VM2 respectively.

The action a_t at time step t is a selection of alternative scheduling schemes. In the above example, the ready task will be assigned to VM2 if the action value equals 2. The reward r_t for each sub-agent is derived from their objectives.

In the following parts of this section, the network model of sub-agents, the interaction processes and the RL training methodology will be presented in more detail.

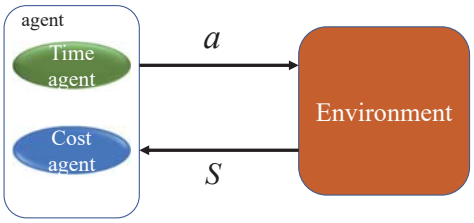


Fig. 2: Multi-agent reinforcement learning architecture.

B. Agent design

In this paper, the agent is designed for selecting eligible tasks (i.e.tasks whose dependencies have been satisfied) and assigning them to available VMs at any time step, which can be modeled as a seq2seq problem. Recurrent Neural Networks (RNNs), such as Long Short Term Memory (LSTM) [11], are usually employed in this kind of problems due to its capability of learning a probability distribution over a sequence [12]. Considering the number of eligible tasks is variable due to different dependency relationships among tasks, we choose pointer network (Ptr-Net) from [13], [14] and [15] as the baseline of our problem with discrete outputs and size changing inputs. In Ptr-Net (depicted in Fig. 3), vectors u^i_j are firstly calculated by the alignment model as:

$$u^i_j = \nu^T \tanh(W_1 e_i + W_2 d_{j-1}),\ j \in (1, ...,\ n_s) \tag{13}$$

where i is time step, ν, W_1 and W_2 are trainable parameters, e_i and d_j are the elements in hidden states of the encoder and the decoder respectively, n_s equals the number of input vectors. u^i_j scores the matching degree between the i^{th} output and the j^{th} input. Then, Ptr-Net utilizes vector u^i_j as a pointer to input elements and the conditional probability can be estimated as:

$$P(y^p_i |\ y^p_1,\ ..., y^p_{i-1},\ X^p) = softmax(u^i) \tag{14}$$

where X^p is the input of Ptr-net, y^p_i is the output at time step i. The softmax function normalizes u^i (of length n_s), which outputs a distribution over input vectors. Considering recurrent models are normally designed for solving problems associated with neural machine translation, it would be inappropriate to apply original Ptr-net directly to handle scheduling problems. Therefore, in this paper, Ptr-net is ameliorated so as to tailor for workflow scheduling and obtain better results. The details about our network (depicted in Fig 4) designed for sub-agents (time and cost sub-agents use the same network) are demonstrated as follows:

Our TFP mainly includes an inner pointer network (IPtr) and an outer layer LSTM network (OLstm). The hidden layer of IPtr is composed of basic RNN units. At each time step, IPtr reads the input vector $s_t = (p_1, p_2, ..., p_{n_t})$ as its order, where t represents actual time step of the current scheduling stage. The output vectors $y_{p_1},\ ..., y_{p_{n_t}}$ of IPtr are obtained through the RNN layer as follows:

$$y_{p_i} = W_{fo} h_{p_i} + b_{fo} \tag{15}$$

$$h_{p_i} = v^T \tanh(W^d_{fr} p_i + W^e_{fr} h^e_{p_i}), i \in (0,\ n_t) \tag{16}$$

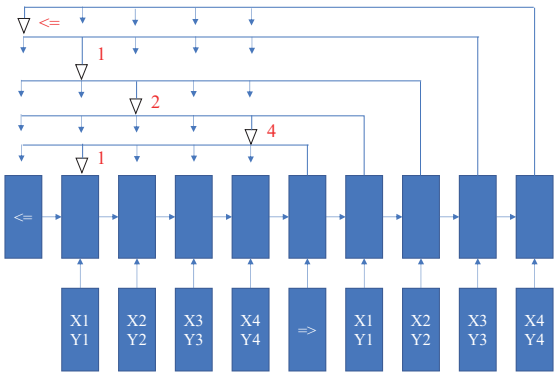


Fig. 3: Pointer network.

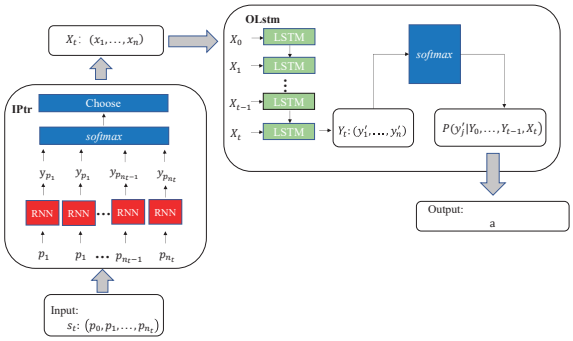


Fig. 4: Temporal fusion pointer (TFP) network.

where v , W_{fo}, b_{fo}, W^d_{fr} and W^e_{fr} are corresponding linear weight matrixes, $h^e_{p_i}$ is an RNN hidden state of IPtr encoder. h_{p_i} is the state of augmented decoder, which contains much information about the effects among alternative scheduling assignments than $h^e_{p_i}$. Then, through the softmax layer of IPtr, the conditional probability is generated as:

$$p(h_{p_i}|s_t) = soft\max(y_{p_i}), i \in (0,\ n_t) \tag{17}$$

It represents the initial selecting probability of alternative scheduling schemes i at state s_t. After that, the first n_x h_{p_i}s are selected according to $p(h_{p_i}|s_t)$ and concatenated to form the initial determining condition X_t.

Then in Olstm, we define the finial determining state Y_t as:

$$Y_t = g(c_{t-1},\ H_t,\ X_t) \tag{18}$$

where H_t and X_t are a hidden and cell state of LSTM in Olstm respectively. g is the model transformation which can be depicted as follows:

$$f(t) = \sigma(W_f[H_{t-1},\ X_t] + +b_f) \tag{19}$$
$$\eta(t) = \sigma(W_\eta[H_{t-1},\ X_t] + b_\eta) \tag{20}$$
$$\tilde{c}_t = \tanh(W_c H_{t-1} + b_c) \tag{21}$$
$$c_t = c_{t-1} \odot f(t) + i(t) \odot \tilde{c}_t \tag{22}$$
$$o(t) = \sigma(W_o[H_{t-1},\ X_t] + b_o) \tag{23}$$
$$H_t = o(t) \odot \tanh(c_t) \tag{24}$$
$$Y_t = W_{bo} H_t + b_{bo} \tag{25}$$

where σ is the sigmoid activation function, $\odot$ is expressed as Hadamard product, W and b in each formula are linear weight matrixes. From the above Eq. (19)-(25) we can see, Y_t contains not only the effects among current alternative scheduling schemes, but also the information about historical decisions, which means the assignments engineered by our agent are much reasonable.

Finally, through the outer softmax layer, the final selection probability of an alternative scheduling scheme at time step t is obtained as:

$$P(y'_j|Y_0, ..., Y_{t-1}, X_t) = soft\max(Y_t), j \in [1,\ n_x] \tag{26}$$

where y'_j is an element of Y_t. Compared with the Ptr-net used in [16], it is obvious that our model can generate better scheduling schemes due to its deeply considering the information both from the current and historical time stages. Throughout the whole network, one notice is that IPtr is composed of basic RNN cells since there is no temporal correlation among each p_i.

C. Interaction

Based on the agent model mentioned above, in our problem, the interaction between the agent and scheduling environment can be demonstrated in Fig. 5. First, the environment is reset to an initial state where the executed task set $\xi_e = \emptyset$ and available task (tasks can be executed) set $\xi_a = \emptyset$. Then, the agent observes time state s^m_t and cost state s^c_t, and delivers them to its sub-agents. Each sub-agent outputs probabilities of alternative schemes which will be taken as its attribute values, and the final action a_t is selected according to Pareto dominance. The criterion for action selection can be defined as:

Definition 3: Suppose P^m_i and P^c_i are final time and cost selection probability associated with p_i, then the action related to p_i will be selected as a_t if:

$$\forall\ j \in (0,\ n_x)\ P^m_i \geq P^m_j \wedge P^c_i \geq P^c_j$$

After that, a_t is mapped to its related scheme which will be implemented immediately and the environmental system evolves to next stage. In addition, the time reward r^m_t and cost reward r^c_t are obtained, and the MDP processes $e^m_t = (s^m_t, a_t, r^m_t, s^m_{t+1})$ and $e^c_t = (s^c_t, a_t, r^c_t, s^c_{t+1})$ are stored to a sample which will be pool used to train agent network models. The above processes (except the first step) will be repeated until all tasks are assigned. When the whole scheduling process is finished, agent network models will be trained by using samples in the pool.

D. Training

In this study, A3C is adopted to train our agent model due to its good performance in researches. It is a class of policy gradient algorithms in RL, which has two main features [17]: First, its training and updating processes are asynchronous between its two kinds of agents: a global agent for storing and updating parameters of sub-agent networks, and local agents for interacting with the environment and recording samples;

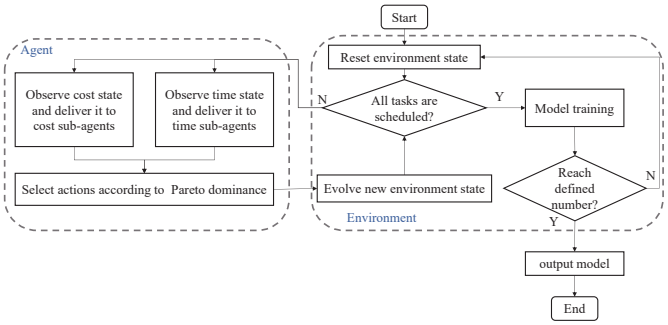


Fig. 5: Interaction processes between agent and environment.

Second, it adopts the idea of value function approximation to obtain a lower variance than normal Monte-Carlo policy gradient. Therefore, it consists of two kinds of models: the critic model evaluating an action-value function, and the actor model taking actions suggested by the critic. When it is combined with our RL structure, as a result, there are two types of sub-models contained in each sub-agent to achieve its decision making and model updating.

The actor and critic in A3C algorithm follow approximate policy and value gradients as follows:

$$\nabla_{\theta_a^q} J(\theta_a^q) = E(\nabla_{\theta_a^z} \log\ \pi(a_t|s_t;\theta_a^z)(R - V(s_i;\theta_s^z))) \quad (27)$$

$$\nabla_{\theta_s^q} J(\theta_s^q) = E(\nabla_{\theta_s^z} (R - V(s_t;\theta_s^z))^2) \quad (28)$$

where θ_a^q, θ_a^z, θ_s^q and θ_s^z are actor and critic model parameters in global and local agents respectively, $V(s_t)$ is the value of state s_t, R is the long-term reward which can be calculated as:

$$R = \sum_{m=1}^{t} \gamma^{m-1} r_t \quad (29)$$

where γ is a discount factor. It can be seen that A3C utilizes the sum of future discounted reward R rather than only using r when computing gradients to update model parameters, which can help to reduce variance and improve the stability of models.

To accelerate the training process, we collect samples in parallel since the learning procedures are based on varied samples. More specifically, 6 threads are activated and each thread contains one global agent and four local agents. When different environment parameters (corresponding to different types and sizes of workflow applications) are set for each thread, the sequences of interactions among threads are different. Details of our training approach for time sub-agent are listed in Algorithm 1 (it is the same as cost sub-agent).

In general, the processes of TFP-RL are shown in Algorithm 2 as follows: First, an agent is initially constructed using the TFP network mentioned in Section III-B. Then, it will interact with the scheduling environment continuously. Once a workflow application is finished, we will train our model based on the A3C method. Finally, if the training time reaches the maximum number of episodes, the resulting model will be output which can be used to produce scheduling plans.

Algorithm 1: A3C based training method

Reset gradients: $d\theta_a^q \leftarrow 0$ and $d\theta_s^q \leftarrow 0$. Synchronize model parameters: $\theta_a^z \leftarrow \theta_a^q$, $\theta_s^z \leftarrow \theta_s^q$. $n_{sa} \leftarrow$ the number of samples in sample pool **for** $i = 1$ *to* n_{sa} **do**
  take a sample according to chronological order.
  $n_w \leftarrow$ the number of the sample. **for** $t = n_w - 1$ *to 0* **do**
    $R \leftarrow r_t^m + \gamma R$. accumulate gradients: $d\theta_a^q \leftarrow d\theta_a^q + \nabla_{\theta_a^z} J$, $d\theta_s^q \leftarrow d\theta_s^q + \nabla_{\theta_s^z} J$.
  end
  perform asynchronous update: $\theta_a^q \leftarrow d\theta_a^q$, $\theta_s^q \leftarrow d\theta_s^q$.
end

Algorithm 2: Total procedures of TFP-RL

Construct agent (based on network in Section III-B) model. $e_{\max} \leftarrow$ defined maximum number of episodes.
for $i = 0$ *to* $e_{\max}$ **do**
  interact with the scheduling environment. train agent model.
end
Output trained model and apply to schedule tasks

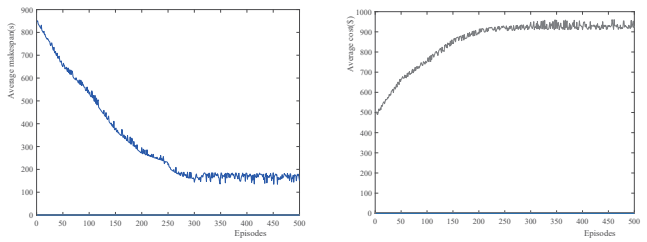
IV. Experiments and Analysis

In this section, we first train our agent network using randomly generated workflows (with 60-120 tasks) to establish its convergence. Then, we increase the number of training and enlarge workflow size to promote its scheduling performance and compare the trained model with other scheduling methods. More details are described as follows.

A. Training

To train our network model outlined in Section III-B and test its convergency, a simulated workflow scheduling environment is implemented. It contains all necessary functions for workflow executions, which can realize simulations with variable workflow sizes, the number of VM and computational power. For each episode, a workflow is dynamically generated whose size is 60-120 tasks and there are 500 episodes during each training. Additionally, the IPtr and OLstm in our network model consist of 260 and 300 hidden units respectively. To reduce and avoid random effects, we conduct model training for 6 times training and the results are depicted in FIG. 6 and Table. I.

From FIG. 6, the time-learning process of our agent starts at approximately 850, and makes a rapid decline before converging to an average makespan of around 150. For cost-learning, the agent starts at a relatively lower value and then makes a gradual increase until reaching an average cost of approximately 900. This results from initially the agent hasn't made enough explorations and prefers to use one or a few VMs. However, this leads to a large negative time reward and drives the agent to learn to accept some costs by using different kinds of VMs. Finally, it finds the balance between reducing makespan and cost. FIG. 6 just shows the third



(a) (b)

Fig. 6: The convergence of TFP-RL with respect to makespan and cost:(a)Average makspan over training process;(b)Average cost over training process.

learning process, while from Table. I, we can see that our agent can always converge to approximate 200 and 900 for makespan and cost though the initial condition is various in 6 training runs. It is apparent that TFP-RL is capable to handle workflow scheduling problems outlined in Section II.

B. Scheduling performance testing

In this subsection, the trained model is encapsulated and its scheduling performance is tested. Six types of VMs in the pricing model of Amazon EC2 [18] and three real-word workflows proposed in [19] are utilized. More details about VMs and workflows are depicted in Table. II and FIG. 7 respectively, where CPU capacity (cu) is represented by million floating-point operations per second (MFLOPS).

1) Algorithm parameters: For the comparison study, ECMSMOO [12] and NSGA-II [20] are selected as the benchmark algorithms, which are typical methods used to address multi-objective optimization problems. Additionally, an RL model with normal Ptr-net (RL-NP) is trained by the same approach as our model, which is used as the baseline method to verify our model effectiveness. For each workflow, 16 independent simulations are conducted for all algorithms. And for the sake of fairness, TFP-RL and RL-NP run 5000 episodes while ECMSMOO and NSGA-II run 100 generations with 50 individuals (100×50=5000) in each simulation. Other parameter settings for algorithms are given in Table III. To simplify the presentation in results, we use M0-M3 to denote TFP-RL, RL-NP, ECMSMOO and NSGA-II respectively.

2) Analysis: To compare the quality of those solutions resulting from algorithms, all non-dominated solutions obtained from each algorithm in 16 simulations are put into four sets respectively. Then, dominated solutions in each set are removed and the Pareto Fronts (depicted in Fig. 8) of each algorithm are constituted by the remainder. It can be intuitively seen that TFP-RL and RL-NP are much better than ECMSMOO and NSGA-II in all cases. This indicates that the searching space of multi-objective cloud workflow scheduling is a little huge for ECMSMOO and NSGA-II to explore. RL-based approaches grasp the ability to find good solutions through training, i.e., their exploring is much more efficient, which indicates a higher probability of obtaining better solutions. Besides, TFP-RL is better than RL-NP in most cases, which verifies the effectiveness of TFP.

Efficiency is another performance for TFP-RL to promote and can be represented as runtime. For each workflow, the average runtime of each algorithm in 16 simulations is reported by runtime (other methods)/ runtime (TFP-RL) (depicted in Fig. 9), which reflects the improvement of TFP-RL over other algorithms. Compared with ECMSMOO and NSGA-II, RL-based methods run much faster since the knowledge obtained through training can help them reduce the searching space. Although TFP-RL is a little slower than RL-NP due to its more complete model structure, the solution quality of TFP-RL is much better. In general, TFP-RL owns the best scheduling performance over those methods.

V. Related work

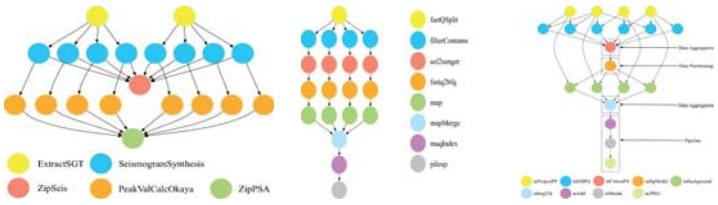
Multi-objective workflow scheduling is an NP-hard problem, which has been studied for many years. The most traditional way to handle this problem is to combine multiple objectives into a single one by assigning different weights [21], [22]. However, it depends on users' preferences to assign a proper weight, thus may not meet the requirements of different users simultaneously.

Therefore, it stimulates many single-objective optimization methods converting to a multi-objective field adopting the idea of Pareto dominance. Durillo and Prodan [23] used basic Heterogeneous Earliest Finish Time (HEFT) and proposed a Pareto-based list scheduling heuristic, i.e., multi-objective HEFT (MOHEFT) to optimize time and cost of the whole workflow execution. Kaur et al. [24] introduced a multi-objective bacteria foraging optimization algorithm (MOBFOA) by applying Pareto-optimal fronts to original BFOA. But heuristic methods are always restricted by problem statements and easily fall into local optima. What is more, there are many meta-heuristics adopted to generate solutions for multi-objective workflow scheduling, such as MOPSO [25] and BOGA [26]. But their computational costs are very high due to a large number of iterations during evolutionary processes.

Recently, RL based algorithms have been developed. Although not as many methods exist in workflow scheduling, the following ones can be found. Cui et al. [27] developed an RL based algorithm for multiple DAGs workflow scheduling. Orhean et al [28] introduced an approach using RL to reduce execution time when scheduling workflows across distributed resources. But those two approaches tackle scheduling problems from a much theoretical perspective which differs from practical scenarios. Wu et al. [29] proposed an improved Q-learning algorithm with a weighted tness value function to optimize completion time and load balancing in the cloud. However, it is hard for Q-learning to address complex scheduling problems, such as scheduling large-scale workflows, since the dimension of the Q matrix will become extremely large. In addition, Wang et al [30] developed a multi-objective workflow scheduling method with deep-q-network-based multi-agent RL (DQN-based MARL). But it is not versatile enough since the input and output size of its DQN network are fixed in length.

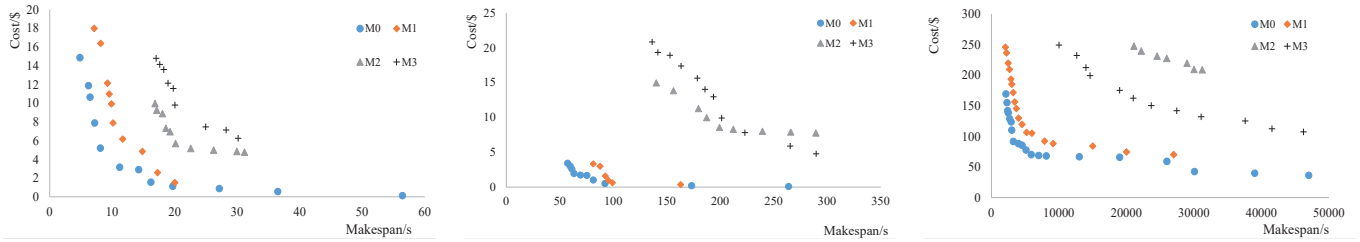
TABLE I: Results in 6 different trainings

		Training time					
		1	2	3	4	5	6
average makespan(s)	initial value	1523.63	966.29	852.13	1200.42	879.35	1422.3
	final value	148.23	151.92	156.2	166.38	141.22	158.96
average cost/($)	initial value	451.2	382.9	498.2	516.3	299.6	456.3
	final value	899.2	954.13	931.26	913.29	962.02	914.26



(a) CyberShake (b) Epigenomics (c) Montage

Fig. 7: Sample structure of five workflow applications.



(a) Montage25 (b) CyberShake100 (c) Epigenomics997

Fig. 8: Cost-makespan trade-off of algorithms for different workflows.

TABLE II: Detailed information of the six types of VM

Type	$cu(MFLOPS)$	$P(hour)$
m1.small	4400	0.06
m1.medium	8800	0.12
m1.large	17600	0.24
m1.xlarge	35200	0.48
m3.large	28300	0.19
m3.xlarge	57200	0.39

TABLE III: Parameter settings for algorithms

Algorithm	**parameters**
M2	$c_1 = c_2 = c_3 = c_4 = 2, w = 1.0 \rightarrow 0.35$.
M3	crossover rate = 1 and mutation rate = $1/n$, where n is the total number of tasks in a workflow.

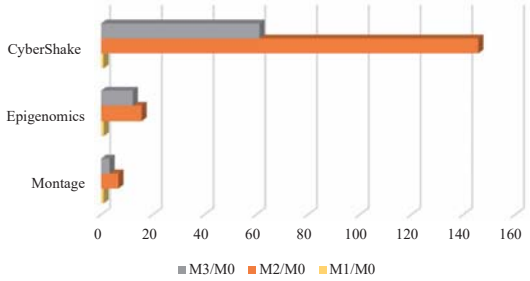


Fig. 9: Runtime comparison for different algorithms on different workflows.

On the contrary, our agent is able to schedule workflows with variable sizes.

VI. Conclusion and Future Work

In this paper, a temporal fusion pointer network-based RL algorithm for multi-objective workflow scheduling (TFP-RL) is proposed. It has the following advantages: (1) an RL based cloud workflow scheduling framework is explored to discover its own heuristics by continuous learning according to rewards from previous good scheduling decisions; (2) a novel temporal fusion pointer (TFP) network is designed for RL agent, which can help to generate more comprehensive and informative scheduling decisions by considering the influence from historical actions so as to improve solution quality; (3) to improve the convergence time, our TFP-RL model is trained independently by using A3C method and the resulting model is utilized to schedule workflow tasks; (4) to select actions more reasonably when addressing multi-objective problems, a Pareto dominance-oriented criterion is developed under multi-agent RL framework. Adequate experiments have been conducted to verify our algorithm and the experiment results illustrated that our algorithm outperforms the compared ones in most cases. In future work, we intend to consider more complex problems, like involving more objectives and constraints.

References

[1] L. M. Vaquero, L. Rodero-merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *ACM-*

SIGCOMM Computer Communication Review, vol. 39, no. 1, pp. 50–55, 2009.

[2] J. D. Ullman, "NP-Complete scheduling problems," *Computer and System Sciences*, vol. 10, no. 3, pp. 384–393, 1975.

[3] D. Nasonov, A. Visheratin, N. Butakov, N. Shindyapina, M. Melnik, and A. Boukhanovsky, "Hybrid evolutionary workflow scheduling algorithm for dynamic heterogeneous distributed computational environment," *Advances in Intelligent Systems and Computing*, vol. 299, pp. 83–92, 2014.

[4] J. M. Batalla, G. Mastorakis, C. X. Mavromoustakis, and J. Zurek, "On cohabitating networking technologies with common wireless access for home automation system purposes," *IEEE Wireless Communications*, vol. 23, no. 5, pp. 76–83, 2016.

[5] M. A. Rodriguez and R. Buyya, "A taxonomy and survey on scheduling algorithms for scientific workflows in IaaS cloud computing environments," *Computing Environments*, vol. 29, no. 8, 2017.

[6] Amazon, "Amazon ec2 pricing," in *Available:http://goo.gl/yKb41s*, 2014.

[7] Microsoft, "Virtual machines pricing details," in *Available:http://goo.gl/UrDkvF*, 2014.

[8] A. Barker and J. I. V. Hemert, "Scientific workflow: A survey and research directions," in *Parallel Processing and Applied Mathematics, 7th International Conference, PPAM 2007, Gdansk, Poland, September 9-12, 2007, Revised Selected Papers*, 2007.

[9] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Articial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[10] A. Greenwald and K. Hall, "Correlated Q-Learning," in *International Conference on Machine Learning, 20th International Conference*, 2003.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] G. Yao, Y. Ding, Y. Jin, and K. Hao, "Endocrine-based coevolutionary multi-swarm for multi-objective workflow scheduling in a cloud system," *Soft Computing*, vol. 21, no. 15, pp. 4309–4322, 2017.

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in Neural Information Processing Systems*, vol. 4, pp. 1–9, 2014.

[14] D. Bahdanau, C. Kyunghyun, and B. Yoshua, "Neural machine translation by jointly learning to align and translate," *ArXiv*, pp. 1–15, 2014.

[15] O. Vinyals, M. Fortunato, and N. Jaitly, "Pointer networks," *Advances in Neural Information Processing Systems*, vol. 28, pp. 1–9, 2015.

[16] A. M. Kintsakis, F. E. Psomopoulos, and P. A. Mitkas, "Reinforcement learning based scheduling in a workflow management system," *Engineering Applications of Artificial Intelligence*, vol. 81, pp. 94–106, 2019.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning, 33th International Conference*, 2016.

[18] Amazon, "Amazon ec2 pricing," in *Available:https://amazonaws-china.com/cn/ec2/pricing/on-demand/*, 2019.

[19] G. Juve, A. Chervenak, E. Deelman, S. Bharathi, G. Mehta, and K. Vahi, "Characterizing and profiling scientfic worflows," *Future Generation and Compute System*, vol. 29, no. 3, pp. 682–692, 2013.

[20] K. Deba, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multi-objective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[21] W. Chen and E. Deelman, "Workflowsim: A toolkit for simulating scientific workflows in distributed environments," in *E-Science, 2012 IEEE 8th International Conference*, 2012.

[22] D. Jeffrey and G. Sanjay, "Mapreduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[23] J. J. Durillo and R. Prodan, "Multi-objective workflow scheduling in amazon ec2," *Cluster Computinng*, vol. 17, no. 2, pp. 169–189, 2014.

[24] M. Kaur and S. Kadam, "A novel multi-objective bacteria foraging optimization algorithm (MOBFOA) for multi-objective scheduling," *Applied Soft Computing*, vol. 66, pp. 183–195, 2018.

[25] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 256–279, 2004.

[26] L. Zhang, K. Li, C. Li, and K. Li, "Bi-objective workflow scheduling of the energy consumption and reliability in heterogeneous computing systems," *Information Sciences*, vol. 379, pp. 241–256, 2016.

[27] D. Cui, W. Ke, Z. Peng, and J. Zuo, "Multiple dags workflow scheduling algorithm based on reinforcement learning in cloud computing," *International Symposium on Intelligence Computation and Applications*, pp. 305–311, 2015.

[28] A. I. Orhean, F. Pop, and I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *Parallel and Distributed Computing*, vol. 117, pp. 292–302, 2018.

[29] J. Wu, Z. Peng, D. Cui, Q. Li, and J. He, "A multi-object optimization cloud workow scheduling algorithm based on reinforcement learning," *Intelligent Computing Theories and Application*, pp. 550–559, 2018.

[30] Y. Wang, H. Liu, W. Zheng, Y. Xia, Y. Li, P. Chen, K. Guo, and H. Xie, "Multi-objective workflow scheduling with deep-q-network-based multi-agent reinforcement learning," *IEEE Access*, vol. 7, pp. 39975–39982, 2019.