Learning Workflow Scheduling on Multi-Resource Clusters

Yang Hu*, Cees de Laat*, Zhiming Zhao*
Informatics Institute, University of Amsterdam, The Netherlands
Email: {y.hu, delaat, z.zhao}@uva.nl

***Abstract*—Workflow scheduling is one of the key issues in the management of workflow execution. Typically, a workflow application can be modeled as a Directed-Acyclic Graph (DAG). In this paper, we present GoDAG, an approach that can learn to well schedule workflows on multi-resource clusters. GoDAG directly learns the scheduling policy from experience through deep reinforcement learning. In order to adapt deep reinforcement learning methods, we propose a novel state representation, a practical action space and a corresponding reward definition for workflow scheduling problem. We implement a GoDAG prototype and a simulator to simulate task running on multi-resource clusters. In the evaluation, we compare the GoDAG with three state-of-the-art heuristics. The results show that GoDAG outperforms the baseline heuristics, leading to less average makespan to different workflow structures.**

***keywords*—Workflow Scheduling, Multi-resource Clusters, Directed-Acyclic Graph (DAG), Deep Reinforcement Learning**

I. INTRODUCTION

Workflow scheduling problems are increasingly common in many scientific communities [1][2]. To workflow applications, a popular representation is the Directed Acyclic Graph (DAG), where each vertex represents a task and edges encode precedence constraints. A task in a DAG relies on the outputs of the precedent tasks and cannot be started until all its required inputs are in place. Workflow technologies [3] are responsible for scheduling computational tasks on distributed resources, and for managing dependencies among tasks.

Many workflow applications consist of a number of cooperative tasks which usually require more computing power beyond single machine capability. Thus, high performance computing clusters or cloud computing clusters [4] are generally leveraged to execute these workflows. For scheduling workflows in a common cluster with networked machines, it typically has following requirements.

- **Task multi-resource demands.** The resources demanded by a task of workflows are often a combination of CPU, memory, network, etc, which have to be satisfied by the underlying cluster.
- **Workflow time constraints.** For some quality critical workflows [5], the scheduling algorithm has to satisfy a user-defined deadline [6], or tries to minimize the makespan of the workflow [7].
- **Inter-task dependencies.** To a workflow scheduler, it has to carefully arrange the order of tasks with respect to inter-task dependencies of the workflow.

During the past years, cluster scheduling and workflow scheduling have attracted quite a lot research attention. Recent cluster scheduling methods mainly focus on the application performance or cluster efficiency for independent tasks. For instance, Tetris [8], a multi-resource cluster scheduler, packs tasks to machines based on their requirements of all resource types; Firmament [9], a centralized cluster scheduler, can make high-quality placement decisions over large-scale clusters. Unfortunately, these solutions would face difficulties for handling complex dependencies among the tasks. To the workflow scheduling, many scheduling methods have been proposed to satisfy a user-defined deadline with minimum costs [10], or try to minimize the makespan of the workflow [7]. However, the resource utilization of the underlying cluster is not considered in these scheduling algorithms. Therefore, workflow scheduling on multi-resource clusters is still a quite challenging. On one hand, the scheduler has to consider the cluster resource utilization, workflow time constraints and inter-task dependencies at the same to make a decision. On the other hand, most of the existing works are designed for some specific workloads or some specific metrics. As the workloads of request or the metrics of interest changes, researchers have to come up with new approaches to adapt the new situation.

With these challenges, we investigate how to apply machine learning techniques, specifically deep reinforcement learning [11], to handle the workflow scheduling problem. That is to say how to make the system learn to schedule workflows on their own. Reinforcement learning is to produce agents that interact with their environments to learn optimal behaviors. The agents will improve over time through trial and error. At beginning, the agent is not told which actions to take for a task. Then, the agent tries to interact with the environments to learn which actions yield the most reward that it receives based on how well it is doing on the task, which gradually helps the agent to make better decisions. Due to recent advances in deep learning, applying deep neural networks in reinforcement learning can make it possible to deal with more complex problems which have high-dimensional states or actions, such as playing Atari game [12], mastering the game of Go [13], [14], etc. Thus, the breakthrough of deep reinforcement learning provides a promising technique for enabling automatic workflow scheduling.

In this paper, we present GoDAG, an approach that can learn to well schedule workflows on multi-resource clusters. GoDAG directly learns the scheduling policy from experi-

ence through deep reinforcement learning, and the objective of GoDAG is to minimize the average makespan of workflows (the time difference between the start and finish of a workflow). In order to apply deep reinforcement learning to workflow scheduling problem, we present a novel state representation to feed the neural network, a practical action space to realize workflow scheduling, and a corresponding reward definition to distinguish action quality. Moreover, we adopt a critical path-based approach to encode the significant information of inter-task dependencies in state representation. We implement a GoDAG prototype and a simulator to simulate task running on multi-resource clusters. In the evaluation, we compare the GoDAG with three state-of-the-art heuristics. The results show that GoDAG outperforms the baseline heuristics in different scenarios.

II. PROBLEM FORMULATION

We first present the model of the workflow scheduling problem on multi-resource clusters, and the objective in this work.

A. Model Description

We model the cluster as federated resource pools with R types of resources, and let $\overrightarrow{C} = (c_1, c_2, ..., c_R)$ be the vector of resource capacities. We consider a slot-based resource model in this paper. Hence, the element c_i denotes the total slots of the resource i available in the cluster.

We model a workflow job as a set of tasks $\mathbb{J} = \{t_1, t_2, ..., t_N\}$ that are to be executed on the cluster, and $N = |\mathbb{J}|$ is the number of tasks. Similar to prior work [8], [10], the resource demands and the estimated execution time of each task is known upon arrival. For task t_i, let $\overrightarrow{D_i} = (d_i^1, d_i^2, ..., d_i^R)$ be the vector of its resource demands, where the element d_i^j denotes the slots of resource j that the task t_i demands. And let s_i be the estimated execution time of task t_i. We assume that the storage of the cluster is based on Network Attached Storage (NAS) or Storage Area Network (SAN), and the data transmission time between the tasks is roughly fixed. Hence, the estimated execution time includes the data transmission time in our model. To dependency specification, let 0-1 matrix $\mathbb{X} = [x_{ij}]_{N \times N}$ denote the inter-task dependencies. If $x_{ij} = 1$, it means that the task t_i is a precedent task of t_j. During execution, a task can only be started when all its precedent tasks are completed.

B. Objective

For simplicity, preemption is not allowed in the cluster, which means the resources must be allocated continuously from the time that the task starts until it is completed. When the scheduler schedules a task to the cluster, it must make sure that the cluster has sufficient resources to execute the task. The main objective in this paper is to minimize the average workflow makespan (the time difference between the start and finish of a workflow).

III. DEEP REINFORCEMENT LEARNING

In the section, we first briefly introduce deep reinforcement learning techniques [11], [15] which we used in this paper.

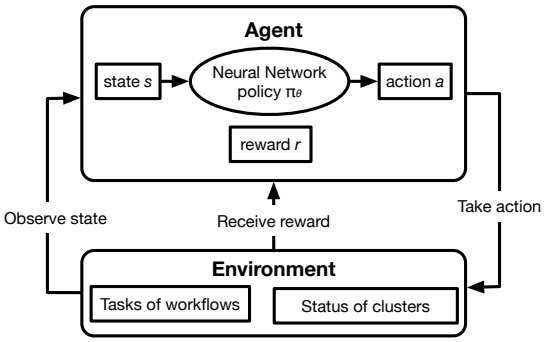


Fig. 1. An example of reinforcement learning

A. Reinforcement Learning

We consider the standard reinforcement learning setting shown in Fig. 1 where an agent interacts with an environment over a number of discrete time steps. At each time step t, the agent receives a state s_t through observation of the environment, and then selects an action a_t from a set of possible actions according to its policy π. The π is a mapping from states s to actions a; it denotes the probability of choosing different actions based on the states. Following the action, the environment transitions to the next state s_{t+1}, and the agent receives a scalar reward r_t. The state transitions and rewards are stochastic, which are assumed to have the Markov property. It means that the state transition probabilities and rewards depend only on the current state s_t of the environment and the action a_t taken by the agent. The process continues until the agent reaches a terminal state after which the process restarts. Every rollout of a policy accumulates rewards from the environment, resulting in the return $R = \sum_{t=0}^{\infty} \gamma^t r_t$ with discount factor $\gamma \in [0, 1]$. The goal of the agent is to find an optimal policy π^*, which achieves the maximum expected return from all states:

$$\pi^* = \underset{\pi}{\operatorname{argmax}}\, \mathbb{E}[R|\pi] \tag{1}$$

B. Value Functions

The value function $V^\pi(s)$ is the expected return when starting in state s and following policy π henceforth:

$$V^\pi(s) = \mathbb{E}[R|s, \pi] \tag{2}$$

The optimal policy π^* has a corresponding value function $V^*(s)$, and vice-versa. The optimal value function can be defined as:

$$V^*(s) = \max_{\pi} V^\pi(s) \tag{3}$$

If we had $V^*(s)$ available, the optimal policy π^* could be easily retrieved by choosing among all actions avail-

able at state s_t and picking the action a_t that maximizes $r_t + V^*(s_{t+1})$.

In small cases, tabular methods or non-parametric methods [11] can be used to compute the $V^*(s)$. However, there are too many possible states in most practical problems [12], [13], including the workflow scheduling problem. It is impossible to store the policy in a tabular form. Hence, the value function is commonly represented using a function approximator [16], such as neural networks. In neural networks, there are a certain number of adjustable parameters θ. Let $\pi_\theta(s, a)$ be the probability of taking action a in the state s given the parameters θ. Thus, different kinds of policies can be derived from adjusting the parameters θ of the neural network. Consequently, we obtain deep reinforcement learning methods when we use deep neural networks to approximate the value function and the policy with different parameters.

C. Actor-Critic Methods

In this paper, we focus on one of the policy gradient methods [11], *actor-critic method*, to train the neural network. The actor-critic method trains two neural networks at the same time: *Actor neural network* and *Critic neural network*. Actor neural network is trained to be an estimate of the optimal policy. Critic neural network is trained to be an estimate of the optimal value function. As actor-critic method is one of the policy gradient methods, it is to learn the policy by performing gradient descent on the parameters. The key idea in policy gradient methods is to estimate the gradient of the expected total rewards by observing the trajectories of executions obtained by following a policy. As mentioned earlier, the objective of reinforcement learning is to maximize the expected cumulative discounted rewards. The gradient of this objective can be computed as [17]:

$$\nabla_\theta \mathbb{E}_{\pi_\theta}[R] = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)] \tag{4}$$

$A^{\pi_\theta}(s, a)$ is the advantage function, which represents the difference in the expected total reward when we deterministically pick action a in state s, compared with the expected reward for actions drawn from policy π_θ. In our work, the agent samples a trajectory of scheduling decisions and uses the empirically computed advantage $A(s_t, a_t)$, as an unbiased estimate of $A^{\pi_\theta}(s_t, a_t)$. Each update of the policy neural network (actor neural network) parameters θ as follows, where α is the learning rate.

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) A(s_t, a_t) \tag{5}$$

The idea behind this equation can be intuitively explained as follows. The direction $\log \pi_\theta(s_t, a_t)$ specifies how to change the policy parameters in order to increase the probability $\pi_\theta(s_t, a_t)$. Equation 5 takes a step in this direction. The size of the step depends on the value of the advantage for action a_t in state s_t. Thus, the net effect is to reinforce actions that empirically lead to better returns.

To compute the advantage $A(s_t, a_t)$ for a given experience, we need an estimate of the value function $V^{\pi_\theta}(s)$ which is the expected total return when starting in state s and following policy π_θ. The role of the critic neural network is to learn an estimate of $V^{\pi_\theta}(s)$ from empirically observed rewards. Let $V^{\theta_v}(s)$, an estimate of $V^{\pi_\theta}(s)$, be the output of critic neural network. Hence, the advantage $A(s_t, a_t)$ can be estimated as $r_t + \gamma V^{\theta_v}(s_{t+1}) - V^{\theta_v}(s_t)$. It is important to note that the critic neural network merely helps to train actor neural network. Post-training, only the actor neural network is required to make scheduling decisions.

IV. GoDAG

In this section, we first present the basic design of our approach GoDAG. Second, we introduce how to represent the state and action of the workflow scheduling problem, and how to define the reward in deep reinforcement learning. Finally, we describe the training algorithm we used to train the neural network.

A. Design

The basic design of GoDAG is the reinforcement learning agent continuously observes the state of the system in discrete time steps, which includes the status of the cluster and the profiles of pending tasks whose precedent tasks have been completed. Based on the current state of the system, the agent makes a scheduling decisions through the policy neural network. A decision can be either an assignment of one specific task or a movement. If it is a valid assignment where the cluster has sufficient resources for the task, the agent schedules the task to the cluster. If the decision is an invalid assignment or a movement, the agent just lets the system run for one time step. This process continues until all tasks of the workflow are completed. Fig. 2 shows an example of GoDAG process. In the workflow, task1 is completed; task2 is running in the cluster; task3 and task4 are ready to execute. The agent feeds the policy neural network with the information of pending tasks and the cluster. According to the output of the policy neural network, the agent schedules the task3 to the cluster at this time step. After all tasks finish, the agent would train the neural network based on the rewards it received. Thus, the challenges of applying deep reinforcement learning are how to represent the state to feed the neural network, how to define the action to realize workflow scheduling, and how to define the reward to distinguish the quality of the actions taken by the agent.

B. State Space

To the feed the neural network, we represent the state of the system as different images. Fig. 3 shows an example of a state representation of GoDAG. In the image, all the information about the cluster and the pending tasks are represented as 0-1 matrices. The left of the representation shows the resource utilization of the cluster with 2 types of resources (CPU and memory). Each matrix represents a type of resource. In the matrix, each column denotes one resource slot in the cluster.

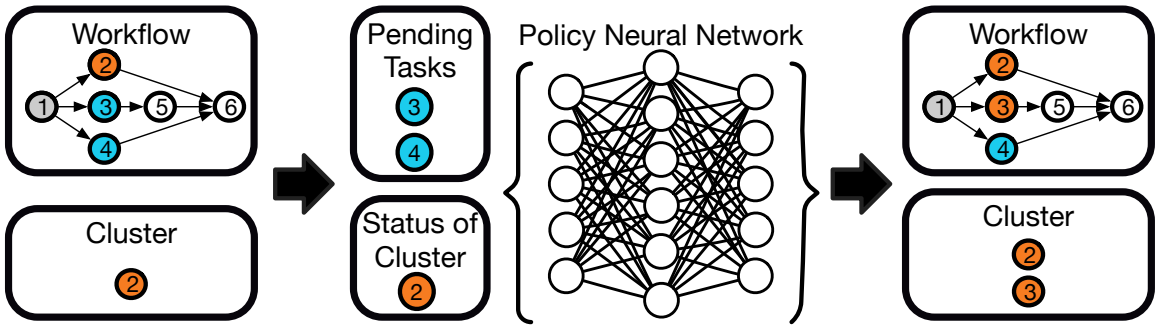


Fig. 2. An example of GoDAG process

If the column is not empty, it means the resource slot has been assigned to a task, and the number of occupations means how many time steps are left for the task to complete. For example, the first CPU slot of the cluster 1 has been assigned to a task, and the task will take 3 time steps to complete. If the column is empty, it means the resource slot is available to be allocated to pending tasks. The right of the representation shows the profiles of the pending tasks, whose precedent tasks are completed. For fixing the state representation, we only show up to N' pending tasks in the image. It also benefits the definition of the action space (discuss in next section). To each task, we use a single-column matrix to directly represent the resource demands and execution time. For example, the task3 demands 2 CPU slots and 3 Memory slots for 3 time steps.

In order to make better decisions for the scheduling, the inter-task dependencies of the workflow are very important to the agent. However, if we directly treat the dependency matrix $\mathbb{X} = [x_{ij}]_{N \times N}$ as a part of state representation and feed it to the neural network, the neural network is still not able to learn the knowledge of the inter-task dependencies due to lacking association information between the tasks and the matrix in the representation. In order to learn better, we adopt a critical path-based approach to encode the significant information of the inter-task dependencies. A critical path of a task is the path of the longest duration between the task and the finish of the workflow. Classic critical path-based schedulers [18], [19] schedule tasks in the order of their critical path length to minimize the makespan. In GoDAG, we present a more sophisticated way to depict the critical path of each task in the state representation. As shown in Fig. 3, each task has a stage number matrix and critical path matrix. Next, we describe how to construct the matrices.

As the task execution time and the inter-task dependencies are known upon arrival, we can preprocess the workflow in advance. Fig. 4 shows the preprocessing procedure. We first divide the tasks of a workflow into different stages. The division depends on the longest distance (number of tasks) from the start of the workflow. In our implementation, we perform depth first search to process the division, and the time complexity is $O(n^2)$. Meanwhile, the critical path of each task is computed during the dividing process, since each task only needs to record the next task in its critical path. Then, we can construct the stage number matrix and critical path matrix for each task. For instance, in Fig. 4, the critical path of task1 is: $task1 \rightarrow task2 \rightarrow task6$ with total duration 10 time steps, and the stages of these task are: $1 \rightarrow 2 \rightarrow 4$. In order to give the neural network an overview of the scheduling process, we also feed the number of scheduled tasks including the completed tasks and the running tasks to the neural network. In Fig. 3, two tasks of the workflow have been scheduled to the cluster.

C. Action Space

At each scheduling process of GoDAG, the agent may schedule any subset of the N' pending tasks (in state representation) to the cluster. It would make the size of action space as large as $2^{N'}$, where the deep reinforcement learning is almost impossible to learn a good scheduling policy [13]. To overcome this problem, we allow the agent take more the one schedule actions in one time step. Hence, we can define the action space as a set: $\{\emptyset, 1, 2, ..., N'\}$, and the size of action space is reduced to $(N' + 1)$. $action = \emptyset$ means a movement action; the agent would let the system run for one time step by taking this action. $action = i$ means an assignment action of the ith pending task in the state representation. If it is a valid assignment where the cluster has sufficient resources, the agent schedules the task to the cluster. Otherwise, let the system run for one time step. If a pending task is assigned to the cluster, the agent would observe the system immediately and perform the scheduling process again. That allows the agent to schedule many tasks in one time step until it chooses a movement action or an invalid assignment action.

D. Reward

In reinforcement learning, the reward is a signal that the environment tells the agent how well it is doing on the task. Typically, the reward is a scalar value. Since the objective of reinforcement learning is to maximize the expected cumulative rewards, the definition of the reward must reflect the goal of the problem to be solved. In GoDAG, we set the reward to 0 for all the actions that are taken during execution except the last action. The reward from the last action is defined as $\frac{N}{Makespan}$, as we can obtain the makespan of the workflow after the last action. Therefore, our goal is consistent with the

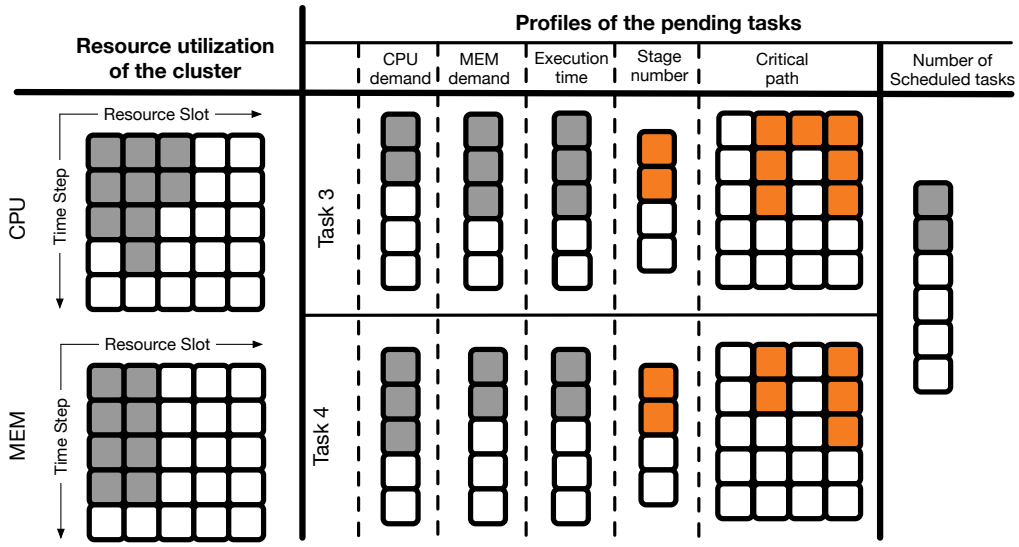


Fig. 3. An example of the state representation of GoDAG

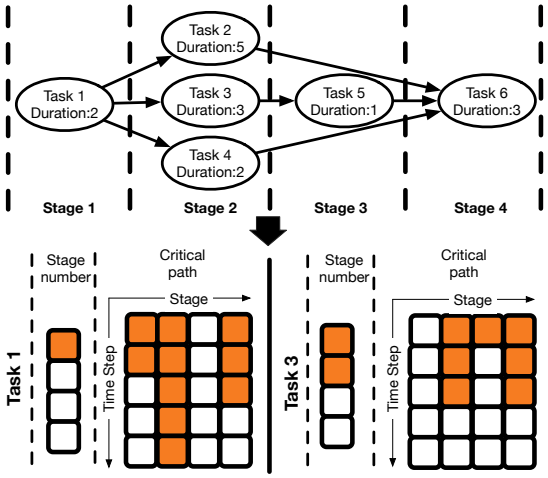


Fig. 4. A critical path-based approach to encode the information of inter-task dependencies

goal of the reinforcement learning. The smaller the makespan is, the greater the cumulative rewards are, and vice versa.

E. Training Algorithm

After defining the state space, the action space and the reward, we could build a deep reinforcement learning agent to learn the scheduling policy. According to the actor-critic method, we train two neural networks at the same time: actor neural network and critic neural network. As the critic neural network is an estimate of the optimal value function, the output of the network is an estimate of the maximum expected return from a state. While the actor neural network is an estimate of the optimal policy, the output of the network is a $(N'+1)$-dimensional vector whose element represents the probability of a corresponding action. Algorithm 1 is used to train the two neural networks in GoDAG. The basic process can be describes as follows. When a workflow job arrives, the agent schedules tasks of the workflow according to its current policy (output of the actor neural network). After the workflow completes, the agent first uses the entire trajectory includes all the states, the actions and the rewards to calculate the accumulated gradients. Then, the agent updates the parameters of the two neural networks with the accumulated gradients. This process would be iteratively repeated many times to empirically learn a better scheduling policy.

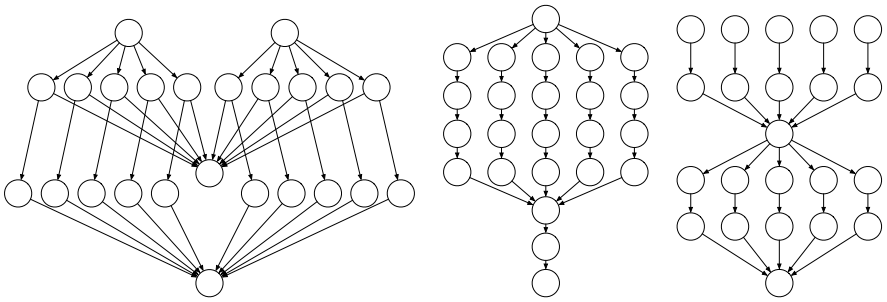
V. EVALUATION

We implement a GoDAG prototype and an simulator to simulate task running on multi-resource clusters. In the evaluation, we experimentally compare GoDAG with three state-of-the-art heuristics in different scenarios, and try to understand the convergence and improvement about GoDAG approach.

A. Setup

Cluster and workloads To the configuration of cluster, we consider 2 types of resources in the experimental cluster, which has 48 resource slots for each type. To the configuration of workloads, we use three classic workflows according to the work [20] as shown in Fig. 5. Each workflow we used in the evaluation has 100 tasks. Considering diversity of workflow jobs, we randomly generate the resource demands and execution time for each task in the workflow. We define the length of one time step as $1t$, and the capacity of one resource slot as $1r$. Then, the execution time of each task is uniformly picked at random from $[1t, 10t]$. Each task demands 2 types of resources, and each resource demand is also uniformly picked at random from $[1r, 12r]$.

GoDAG configuration The configuration of the actor neural network is: 1 input layer, 2 fully connected hidden layers with ReLU6 nonlinearity, and 1 softmax output layer. The configuration of the critical neural network is the same with the actor neural network except the output layer with only



(a) CyberShake (b) Epigenomics (c) LIGO

Fig. 5. Structure of the three different workflows used in the experiments

Algorithm 1: Actor-critical method in GoDAG

```
/* Assume the actor neural network with
   parameters θ, and the critical
   neural network with parameters θ_v  */
for each iteration do
    dθ ← 0;
    dθ_v ← 0;
    t ← 0;
    Get state s_t;
    repeat
        Perform action a_t according to policy π_θ;
        Receive reward r_t and new state s_{t+1};
        t ← t + 1;
    until workflow completes;
    R ← N/Makespan;
    /* N is the total number of tasks,
       Makespan = t                    */
    for i ← t − 1; i ≥ 0; i ← i − 1 do
        R ← r_i + γR;
        dθ ← dθ + ∇_θ log π_θ(s_i, a_i)(R − V^{θ_v}(s_i));
        dθ_v ← dθ_v + ∂(R − V^{θ_v}(s_i))^2/∂θ_v;
    end
    Perform update of θ using dθ;
    Perform update of θ_v using dθ_v;
end
```

1 neuron. We update the parameters of two neural networks using the $rmsprop$ [21] algorithm. The learning rate of the actor and critical neural network are configured to be 0.0001 and 0.001 respectively. We set the discount factor γ as 0.99 to calculate the cumulative discounted rewards. We adopt the asynchronous method, A3C [17], to speed up training. In the experiments, all the neural networks are trained for 10,000 iterations.

Baselines We compare GoDAG with three heuristics:

- Multi-resource Packer (PACK): The idea of this heuristic [8] is that it schedules pending tasks in increasing order of alignment between resource demands and resource availability.
- Shortest Task First (STF): The idea of this heuristic is that it schedules pending tasks in increasing order of the task execution time.
- Critical Path First (CPF): The idea of this heuristic [19] is that it schedules pending tasks in increasing order of the length of critical path.

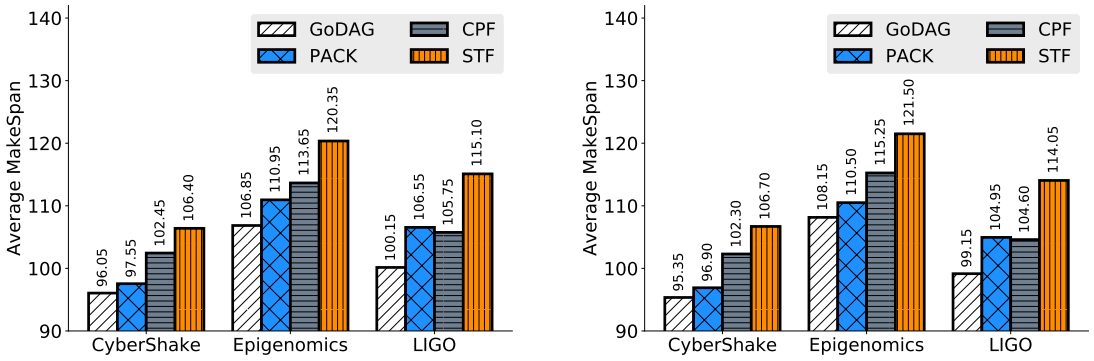
Metrics In the experiments, we mainly measure *average makespan* of the workflows in different scenarios.

B. Scenario with Same Structure

First, we evaluate the scheduling performance when running workflows with same structure in the system. For one structure, we generate 300 workflow jobs, 30,000 tasks in total, to train the neural networks. We generate other 300 workflow jobs of the same structure to test the trained neural network. At each training iteration, each workflow job is executed once. Fig. 6 shows the results after 10,000 iterations. We observe that GoDAG outperforms other baseline heuristics for all structures. Shortest task first heuristic does not perform well in this experiment. Multi-resource pack and critical path first performs comparably, which indicates that the resource efficiency of the cluster and the critical path of the workflow have important influence on the makespan of workflows. Comparing the testing workflows to the training workflows, we find that the improvement of the testing workflows slightly lower than the training workflows. It is may caused by the overfitting of learning some particular workflow patterns during the training process. The basic reason for the improvement is GoDAG can learn to schedule workflows by taking cluster resource utilization, task execution time, task resource demands and inter-task dependencies into consideration in the same time, where those information are well presented in the state representation.

C. Scenario with Different Structures

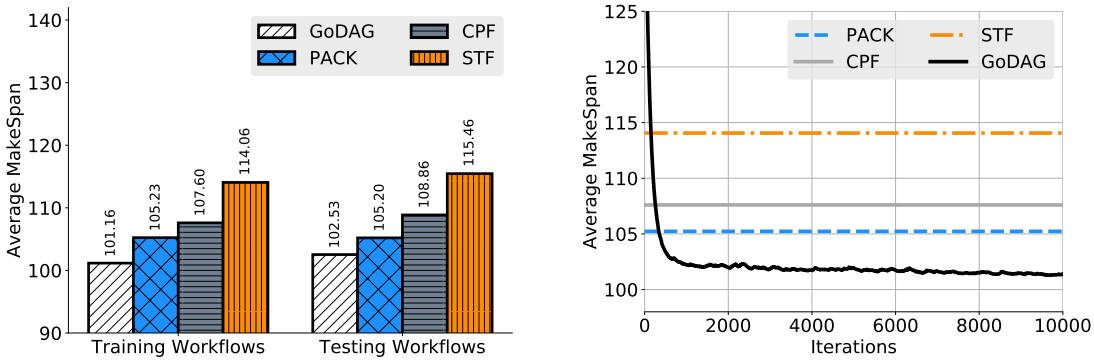
Next, we evaluate the scheduling performance when running workflows with different structures in the system. In this experiment, we generate 100 workflow jobs for each structure, 300 workflow jobs in total, to train the neural network. We also generate other 300 workflow jobs (100 for each structure) to test the trained neural network. Fig. 7(a) shows the results after 10,000 iterations. Similar to the scenario with same structure, GoDAG outperforms other baseline heuristics in the evaluation, and the improvement of testing workflows also slightly



(a) Training workflows

(b) Testing workflows

Fig. 6. The results of evaluating workflows with same structure



(a) Scheduling performance

(b) Makespan curve along the training

Fig. 7. The results of evaluating workflows with different structures

lower than the training workflows. Additionally, we show the average makespan curve along the training in Fig. 7(b). From the figure, we observe that GoDAG outperforms other heuristics after 300 iterations, and tends to be stable after 1,000 iterations. The agent slowly improve the performance after 1,000 iterations, where the agent may try to learn the specific scheduling policy only for the training workflows. Nevertheless, it demonstrates that GoDAG is able to learn one scheduling policy to handle workflow scheduling with different structures. It also implies GoDAG can capture common features among different structures to form a scheduling policy.

VI. RELATED WORK

The problem investigated in this paper - Learning Workflow Scheduling on Multi-Resource Clusters - is related to a variety of research topics as follows.

Cluster schedulers Many cluster schedulers have been proposed for different purposes [22], [9], [23]. Tetris [8] multi-resource cluster scheduler packs tasks to machines based on their requirements of all resource types. It adapts heuristics for the multi-dimensional bin packing problem to the context of cluster schedulers wherein task arrivals and machine availability change in an online manner. Graphene [24] DAG scheduler schedules jobs that have a complex dependency structure and heterogeneous resource demands. It first schedules troublesome tasks and then schedules the remaining tasks without violating dependencies to improves job completion time. Differently, we apply reinforcement learning to learn scheduling policy directly from the experience.

Deep Reinforcement Learning Recently, deep reinforcement learning has made great success in many areas [25], [26], [27], [28]. Volodymyr Mnih et al. [16] presented the first deep learning model to successfully learn control policies directly from high-dimensional sensory input using reinforcement learning. And they have successfully applied their approach to the computer video games. David Silver et al. [13] proposed AlphaGo to master the game of Go with deep neural networks and tree search. They introduced a new search algorithm that combines Monte Carlo simulation with value and policy networks, and successfully applied in the game of Go. In this paper, we try to apply the deep reinforcement learning to the workflow scheduling problem.

Scheduling with Reinforcement Learning DeepRM [29] is the first example solution that applies deep reinforcement learning to cluster scheduling problem. It translates the problem of packing tasks with multiple resource demands

into a learning problem. DeepRM is designed to handle job scheduling in an online setting, and represents the state of the system as distinct images which contain the current allocation of cluster resources and the resource profiles of jobs waiting to be scheduled. However, they did not model inter-task dependencies in their job models, and hence it can only schedule independent tasks.

VII. CONCLUSION

In this paper, we present GoDAG, an approach that can learn workflow scheduling on multi-resource clusters. GoDAG directly learns the scheduling policy from experience through deep reinforcement learning. In the evaluation, we compare the GoDAG with three state-of-the-art heuristics. The results show that GoDAG outperforms the baseline heuristics in different scenarios. In the future, we intend to investigate more compact state representation to enhance the scalability of GoDAG and apply convolution neural network to extract high-level features. Moreover, we plan to apply GoDAG to the production cluster and evaluate the performance in practical scenario.

ACKNOWLEDGMENTS

This research has received funding from the European Union's Horizon 2020 research and innovation program under grant agreements 643963 (SWITCH project), 654182 (ENVRIplus project), 676247 (VRE4EIC project), 824068 (ENVRI-FAIR project) and 825134 (ARTICONF project). The research is also supported by Chinese Scholarship Council.

REFERENCES

[1] Z. Zhao, A. Belloum, C. de Laat, P. Adriaans, and B. Hertzberger, "Distributed execution of aggregated multi domain workflows using an agent framework," in *2007 IEEE Congress on Services (Services 2007)*. Salt Lake City, UT, USA: IEEE, Jul. 2007, pp. 183–190. [Online]. Available: http://ieeexplore.ieee.org/document/4278795/

[2] G. Casale, C. Chesta, P. Deussen, E. Di Nitto, P. Gouvas, S. Koussouris, V. Stankovski, A. Symeonidis, V. Vlassiou, A. Zafeiropoulos, and Z. Zhao, "Current and Future Challenges of Software Engineering for Services and Applications," *Procedia Computer Science*, vol. 97, pp. 34–42, 2016. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1877050916320944

[3] Z. Zhao, A. Belloum, C. De Laat, P. Adriaans, and B. Hertzberger, "Using Jade agent framework to prototype an e-Science workflow bus," in *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*. Rio de Janeiro, Brazil: IEEE, May 2007, pp. 655–660. [Online]. Available: http://ieeexplore.ieee.org/document/4215434/

[4] Y. Hu, H. Zhou, C. de Laat, and Z. Zhao, "ECSched: Efficient Container Scheduling on Heterogeneous Clusters," in *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Cham: Springer International Publishing, 2018, vol. 11014, pp. 365–377. [Online]. Available: http://link.springer.com/10.1007/978-3-319-96983-1_26

[5] J. Wang, A. Taal, P. Martin, Y. Hu, H. Zhou, J. Pang, C. de Laat, and Z. Zhao, "Planning virtual infrastructures for time critical applications with multiple deadline constraints," *Future Generation Computer Systems*, vol. 75, pp. 365–375, 2017.

[6] Y. Hu, J. Wang, H. Zhou, P. Martin, A. Taal, C. de Laat, and Z. Zhao, "Deadline-Aware Deployment for Time Critical Applications in Clouds," in *Euro-Par 2017: Parallel Processing*, F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, Eds. Cham: Springer International Publishing, 2017, vol. 10417, pp. 345–357. [Online]. Available: http://link.springer.com/10.1007/978-3-319-64203-1_25

[7] R. Bajaj and D. P. Agrawal, "Improving scheduling of tasks in a heterogeneous environment," *IEEE Transactions on Parallel and Distributed Systems*, vol. 15, no. 2, pp. 107–118, 2004.

[8] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 455–466, 2015.

[9] I. Gog, M. Schwarzkopf, A. Gleave, R. N. Watson, and S. Hand, "Firmament: Fast, centralized cluster scheduling at scale." Usenix, 2016.

[10] S. Abrishami, M. Naghibzadeh, and D. H. Epema, "Deadline-constrained workflow scheduling algorithms for infrastructure as a service clouds," *Future Generation Computer Systems*, vol. 29, no. 1, pp. 158–169, 2013.

[11] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[14] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[15] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A brief survey of deep reinforcement learning," *arXiv preprint arXiv:1708.05866*, 2017.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[18] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM journal on Applied Mathematics*, vol. 17, no. 2, pp. 416–429, 1969.

[19] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE transactions on parallel and distributed systems*, vol. 7, no. 5, pp. 506–521, 1996.

[20] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good *et al.*, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, no. 3, pp. 219–237, 2005.

[21] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, p. 14, 2012.

[22] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica, "Sparrow: distributed, low latency scheduling," in *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*. ACM, 2013, pp. 69–84.

[23] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: fair scheduling for distributed computing clusters," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. ACM, 2009, pp. 261–276.

[24] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "G: Packing and dependency-aware scheduling for data-parallel clusters," in *Proceedings of OSDI'16: 12th USENIX Symposium on Operating Systems Design and Implementation*, 2016, p. 81.

[25] L. Deng, D. Yu *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[26] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun, "Dermatologist-level classification of skin cancer with deep neural networks," *Nature*, vol. 542, no. 7639, p. 115, 2017.

[27] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, 2018.

[28] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and Brain Sciences*, vol. 40, 2017.

[29] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. ACM, 2016, pp. 50–56.