A Dual-Agent Scheduler for Distributed Deep Learning Jobs on Public Cloud via Reinforcement Learning

Mingzhe Xing
mzxing@stu.pku.edu.cn
Peking University
Beijing, China

Hangyu Mao
maohangyu@sensetime.com
Sensetime Research
Beijing, China

Shenglin Yin
yinsl@stu.pku.edu.cn
Peking University
Beijing, China

Lichen Pan
plch368@pku.edu.cn
Peking University
Beijing, China

Zhengchao Zhang
iamzczhang@gmail.com
ByteDance
Shanghai, China

Zhen Xiao*
xiaozhen@pku.edu.cn
Peking University
Beijing, China

Jieyi Long
jieyi@thetalabs.org
Theta Labs, Inc.
San Jose, United States

ABSTRACT

Public cloud GPU clusters are becoming emerging platforms for training distributed deep learning jobs. Under this training paradigm, the *job scheduler* is a crucial component to improve user experiences, *i.e.,* reducing training fees and job completion time, which can also save power costs for service providers. However, the scheduling problem is known to be NP-hard. Most existing work divides it into two easier sub-tasks, *i.e., ordering task* and *placement task*, which are responsible for deciding the scheduling orders of jobs and placement orders of GPU machines, respectively. Due to the superior adaptation ability, learning-based policies can generally perform better than traditional heuristic-based methods. Nevertheless, there are still two main challenges that have not been well-solved. First, most learning-based methods only focus on ordering or placement policy independently, while ignoring their cooperation. Second, the unbalanced machine performances and resource contention impose huge overhead and uncertainty on job duration, but rarely be considered in existing work. To tackle these issues, this paper presents a dual-agent scheduler framework abstracted from the two sub-tasks to jointly learn the ordering and placement policies and make better-informed scheduling decisions. Specifically, we design an ordering agent with a scalable squeeze-and-communicate strategy for better cooperation; for the placement agent, we propose a novel Random Walk Gaussian Process to learn the performance similarities of GPU machines while being aware of the uncertain performance fluctuation. Finally, the dual-agent is jointly optimized with multi-agent reinforcement learning. Extensive experiments conducted on the real-world production cluster trace demonstrate the superiority of our model.

CCS CONCEPTS

• **Theory of computation** → **Scheduling algorithms**; **Multi-agent reinforcement learning**; **Gaussian processes**.

KEYWORDS

Job Scheduling; Reinforcement Learning; Gaussian Process

ACM Reference Format:
Mingzhe Xing, Hangyu Mao, Shenglin Yin, Lichen Pan, Zhengchao Zhang, Zhen Xiao, and Jieyi Long. 2023. A Dual-Agent Scheduler for Distributed Deep Learning Jobs on Public Cloud via Reinforcement Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3580305.3599241

1 INTRODUCTION

Driven by recent innovations of artificial intelligence, deep learning (DL) is powering various services such as computer vision [16, 25] and language processing [2, 9]. To deal with the ever-growing scale of training datasets, distributed DL (DDL) training is a common practice to shorten the training time. However, training DDL jobs usually requires powerful and expensive GPUs, and it gradually becomes infeasible to fit them into a private cluster. Therefore, many public cloud service providers, *e.g.,* AWS[1] and Alibaba Cloud[2], have built GPU clusters to gain monetary benefits by training DDL jobs for users. It is reported that the GPU as a service market size is valued at 2.8 billion dollars in 2022, and will keep a 40% growth rate in the next year[3]. Under this training paradigm, an effective DDL job scheduler is critical to improving user experiences, *i.e.,* saving budgets and reducing job completion time (JCT), and even marginal improvements can lead to large social benefits. From the

*Corresponding author.

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599241

[1] https://aws.amazon.com/
[2] https://www.alibabacloud.com/
[3] https://www.futuremarketinsights.com/reports/gpu-as-a-service-market

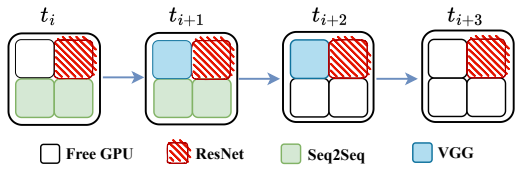


Figure 1: Illustrative example of uncertain performance fluctuation caused by future arrived and co-located jobs in a machine equipped with four GPUs during four timesteps.

perspective of service providers, optimizing JCT also allows them to reduce power costs and complete more jobs in the same time so as to raise revenue. In addition, cheap and fast computing services will attract more users and improve potential income.

As a typical online bin-packing problem, job scheduling is known to be NP-hard [17, 74], and the mainstream solutions divide this problem into two easier sub-tasks, *i.e., ordering task* and *placement task*. The ordering task is responsible for deciding the scheduling orders of jobs, and the placement task scores the affinity of a job with respect to each machine, where a higher affinity score implies that the job is more suitable to be placed on this machine. Most existing work adopts heuristic or meta-heuristic rules in the two sub-tasks, namely *heuristic-based* [12] and *meta-heuristic-based scheduler* [81]. They are typically based on rules meticulously tuned by experts or injecting randomness in solution space to search for the optimal solution, which is not only a time-consuming process but also subject to human cognitive bias. To remedy these drawbacks, another line of work [39, 73] uses reinforcement learning (RL) [56] to optimize the ordering or placement policy, namely *RL-based scheduler*.

Despite the superiority of RL-based schedulers, two challenging issues still have not been solved. First, to the best of our knowledge, all existing RL-based methods train the ordering or placement policies independently. They ignore the potential cooperation between the two policies, which can be exploited to further improve the performance. The second challenge comes from the *placement sensitivity*. The placement agent should be aware of the machine computation and topological communication performances, which is the key to placing distributed jobs [71]. However, the *system uncertainty* [57, 75] existing in the dynamic cluster environment is with little prior knowledge, especially for the *resource contention* [3] from future co-located jobs, making the machine performances fluctuate and hard to predict. For example, as shown in Figure 1, a ResNet [25] job is scheduled and placed with two Seq2Seq [55] jobs at t_i under slight resource contention. However, the future co-located VGG [52] job could lead to nearly 2-x slowdown [3] for the ResNet during t_{i+1} to t_{i+2}, as they all demand and contend for more network I/O to communicate gradients. This example shows the necessity and difficulty of perceiving uncertain future performance fluctuation multi-step-ahead when scheduling the ResNet job at t_i.

To alleviate the above issues, we propose a novel Dual-Agent Scheduler (named DAS) framework for scheduling DDL jobs. First, we analyze and abstract existing ordering and placement methods as a *dual-agent* structure, and formulate the DDL job scheduling process as a Decentralized Markov Decision Process [5]. Second, we design a scalable *squeeze-and-communicate* mechanism to convey cluster state to the ordering agent, which helps the two agents collaboratively make scheduling decisions, and leverage Transformer [59] architecture to compute the *position-aware* and *pair-wise* ranking for prioritizing jobs. Third, the placement agent should be able to model the computation and topological communication performances of machines, and be aware of the uncertain machine performance fluctuation caused by resource contention from future co-located jobs. To this end, we propose a *Random Walk Gaussian Process (RWGP)*, which consists of a stacked random walk kernel function to learn the topological performance similarities of machines. Furthermore, the RWGP is built on the Gaussian Process [65], where the estimated posterior covariance reflects the uncertain performance fluctuation and helps to guide the action decision. Finally, the dual-agent is optimized with the multi-agent proximal policy optimization [77] algorithm.

To the best of our knowledge, we are the first to abstract the DDL job scheduler as a dual-agent structure, and utilize the multi-agent reinforcement learning algorithm to optimize the joint policy. Experiments conducted on the real-world DDL job trace show that our method can reduce 34 minutes on JCT per job on average and save more than ten thousand dollars on training fees every day in a production cluster compared with the best learning-based scheduler. In addition, an ablation study, parameter tuning, and visualized case study are provided for a better understanding of our work.

2 BACKGROUND AND RELATED WORK

2.1 Scheduling Algorithms

Existing scheduling methods can be mainly divided into *heuristic-based*, *meta-heuristic-based* and *RL-based* schedulers. A representative **heuristic-based** ordering rule is First-in-first-out (FIFO) [12], which assigns higher priorities to jobs that come earlier. Dominant Resource Fairness (DRF) [23], a generalization of min-max fairness to multiple resource types, computes the shares of the dominant resource of jobs and first schedules the job with the smallest dominant share. As classical placement rules, First-fit [7] selects the first-retrieved machine, while Load-balance [58] prefers the machine with the least workload. Although heuristic algorithms have been widely deployed in industrial clusters due to their superior computation efficiency [26, 60], their performance is often sub-optimal. The **meta-heuristic-based** algorithms [81] try to find the global optimal solution with randomness. Multi-objective simulated annealing (MOSA) [51] searches for the job assignment and resource allocation to minimize makespan and resource cost. Multi-objective hybrid ant-lion optimizer (MALO) [1] solves the job scheduling problem by enhancing the ant optimization algorithm utilizing differential evolution as a local search technique. However, their low efficiency hinders them to be applied in the production cluster. More recently, another line of work proposes **RL-based** schedulers. DeepRM [39] uses a bitmap to model cluster state and employs REINFORCE algorithm [66] to decide the time slot when a job should be executed. DL^2 [43] comprises of offline supervised learning and online RL stages to optimize resource allocation policy. RIFLING [13] focuses on the placement strategy for DDL jobs with Q-learning [62], and adopts k-means [38] to group job representations to reduce state space. These RL-based techniques, although efficient, only consider ordering or placement policy independently.

In this paper, we jointly learn the ordering and placement policies and their cooperation with a dual-agent framework.

2.2 Multi-agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) [44] has received much attention in solving cooperative and competitive tasks, *e.g.,* traffic light control [64] and packet routing [8]. VDN [54] and QMIX [45] optimize agents by value decomposition. Built on the conventional PPO [49], multi-agent proximal policy optimization (MAPPO) [77] consists of multiple agents that could deal with both discrete and continuous action space, and a shared critic network to estimate the Q-value. Since partially observable agents cannot learn the global state and capture the dependencies with the other agents, existing work [40, 41] proposes to communicate between agents in order to stabilize the training process [18, 53] and increase exploration, reward, and diversity of solutions [4]. In order to have a better collaboration between ordering and placement agents, we design a squeeze-and-communicate strategy in our model, which allows scalable message communication.

2.3 Gaussian Process

Gaussian Process (GP) [30] is a Bayesian approach that has been widely used in analyzing time series with uncertainty [65]. Formally, it is defined as a collection of random variables, of which any finite subsets have joint Gaussian distributions. Consequently, a GP can be specified by its mean function $m(x)$ and covariance kernel function $k_\eta(x, x')$ as: $f(x) \sim \mathcal{GP}(m(x), k_\eta(x, x'))$, where $x, x' \subset \mathcal{X}$ denote the possible inputs and η is a set of hyper-parameters of the kernel function. In more realistic modeling situations, it is typical that we can only access noisy function values $y = f(x) + \varepsilon$. Denote f_* as the vector of (unknown) evaluations of f at a finite $X_* = \{x_{1*}, \cdots, x_{n*}\} \subset \mathcal{X}$, the joint prior distribution of the training outputs f and the test outputs f_* is:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} k(X,X) + \sigma^2 I & k(X, X_*) \\ k(X_*, X) & k(X_*, X_*) \end{bmatrix}\right),$$

where $k(X, X_*) = k(X_*, X)^T$ is the cross-covariance matrix between X_* and training input X, *i.e.,* the (i, j)-th element of $k(X_*, X)$ is $k(x_{i*}, x_j)$, and σ^2 is the variance of Gaussian noise ε. The predictive posterior distribution on f_* conditioned on X, X_* and y is therefore given by the followings:

$$m(f_*) = k(x_*, X)\left[k(X,X) + \sigma^2 I\right]^{-1} y \quad (1)$$

$$cov(f_*) = k(x_*, x_*) - k(x_* X)\left[K(X,X) + \sigma^2 I\right]^{-1} K(X, x_*). \quad (2)$$

Taking advantage of GP generalizing well with few labels and can naturally model uncertainty [35, 65], we design a novel GP with random walk kernel [32], which can well model the performance similarity and uncertain performance fluctuation.

3 PROBLEM DEFINITION

To begin with, we give a formal definition of the scheduling process for DDL jobs on a public cloud GPU cluster. Given a GPU cluster with M physical machines, each is equipped with the same amount of GPU cards, CPU, and memory. It consistently receives, schedules, and executes DDL jobs submitted by users, and charges a certain amount of training fees from users. As depicted in Figure 2, DDL job

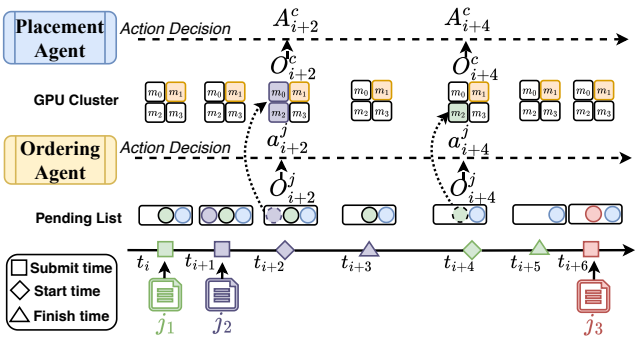


Figure 2: The Dec-MDP formulation of DDL job scheduling. Job j_2 (in purple) is scheduled at t_{i+2} by the ordering agent, and is placed on machines m_0 and m_2 by the placement agent.

j_2 with a *requirement list* $\langle C, n^{GPU}, n^{CPU}, n^{Memory}\rangle$ is submitted by a user at timestep t_{i+1}, namely *submission time*. It comprises of C parallelly running instances to be dispatched to one or more machines, where each instance works for a unique part of the complete training dataset or model, *i.e.,* in data- or model-parallelism manner [29], and requires the same volume of GPU, CPU, and memory, *i.e.,* n^{GPU}, n^{CPU} and n^{Memory}. After being submitted, j_2 will first be added into the *pending list*, then deployed on the cluster after the scheduler makes scheduling decisions for it at t_{i+2}, *i.e.,* the *start time*. When j_2 finishes at t_{i+3} (*i.e.,* the *finish time*), its job completion time (JCT) can be derived as $r^{jct} = t_{i+3} - t_{i+1}$, and the training fee is calculated as $r^{fee} = p \times C \times n^{GPU} \times d$, where p is the price of using a GPU per hour, and $C \times n^{GPU}$ is the total requested number of GPUs and $d = t_{i+3} - t_{i+2}$ is the job duration.

The scheduler module is deployed on the cluster to make *ordering* and *placement* decisions for jobs waiting in the pending list. The *ordering component* is responsible for prioritizing jobs and deciding which job should be scheduled first, while the *placement component* computes the *affinity score* for each job with respect to each machine, and a higher affinity score indicates that the job is more suitable to be placed on this machine. Please note that we only focus on the run-to-completion scheduling for DDL jobs without preemption or migration, since the resource availability of the cluster is constantly changing, and a dynamic scheduler may introduce extra scheduling overhead [13].

In order to train the scheduler with MARL, we abstract the ordering and placement components in the scheduler as *ordering agent* and *placement agent* respectively, and formulate the scheduling process as a Decentralized Markov Decision Process (Dec-MDP) based on the above notations, which can be specified as follows:

• **Observation:** The observation of ordering agent at timestep t is the pending job states that can be specified as $\mathbf{O}_t^j \in \mathbb{R}^{N \times u}$, where N is the number of pending jobs, and u is the number of job features consisting of the instance, CPU, GPU and memory requirements and the wait time in the pending list. Serving for the time-variant workload, the cluster shows a dynamic state as depicted in Figure 2, which is regarded as the observation of placement agent and can be represented as a collection of machine states $\mathbf{O}_t^c \in \mathbb{R}^{M \times v}$ at t, where the k-th row in $\mathbf{O}_t^c$ is a v-length feature vector of machine

m_k, containing the amount of free CPU, GPU, memory, bandwidth and the resource utilization ratio of m_k.

• **Action:** The action of the ordering agent is represented as $\boldsymbol{a}_t^j \in \mathbb{R}^N$, where the i-th element is the scheduling priority of job j_i. Denoted by $\mathbf{A}_t^c \in \mathbb{R}^{N\times M}$, the placement agent makes placement decision for each job, and the (i,k)-th element in $\mathbf{A}_t^c$ denotes the affinity score of placing job j_i on machine m_k.

• **Transition:** Given the ordering and placement action decisions, the scheduler sorts the pending jobs according to priorities $\boldsymbol{a}_t^j$, and allocates resources for them on machines sorted by affinity scores $\mathbf{A}_t^c$. The observations will be transited to the next timestep when the scheduler is triggered, *i.e.,* when there are enough free resources in the cluster to run the pending jobs.

• **Reward:** Basically, our objectives are to minimize JCT r^{jct} and training fee r^{fee}. We additionally design a penalty term for the jobs that are not successfully scheduled: $\alpha(C \times n^{GPU})$, where α is a hyper-parameter to tune the tolerability of failed scheduling. The total reward can be specified as follows:

$$r_t = \sum_{i=0}^{N'} \left(\frac{\zeta_i}{r_i^{fee} \times r_i^{jct}} - (1-\zeta_i)\alpha(C_i \times n_i^{GPU}) \right) / N', \tag{3}$$

where ζ_i is an indicator that equals 1 when job j_i is successfully scheduled and finished, and 0 when j_i fails to be scheduled, and N' is the number of jobs finished and failed during timestep $t-1$ to t.

Remark. In the reward function (*i.e.,* Eq. 3), the JCT and training fee are jointly optimized, but they are non-cooperative metrics and have been proven to expose a hard tradeoff [79] (can also be demonstrated in the experiment part). The proof and additional explanation about the Dec-MDP formulation can be found in Appendix A).

4 METHODOLOGY

In this section, we present a novel Dual-Agent Scheduler (named DAS) for DDL jobs. First, we introduce the ordering agent with a scalable *squeeze-and-communicate* mechanism that can integrate the cluster state, and a Transformer [59] encoder to learn the *position-aware* and *pair-wise* job priorities. Second, to be aware of the topological machine performances and uncertain performance fluctuation, we present the placement agent with a novel *Random Walk Gaussian Process*. After that, we detail the training process with the MAPPO algorithm.

4.1 Ordering Agent with Message Communication

The ordering agent is responsible for assigning *priority scores* for jobs. The job with a higher priority indicates that it will gain more benefits if be scheduled first. In this process, it is intuitive that the cluster observation $\mathbf{O}^c$ is an important side information for the ordering agent to infer the most proper scheduling order under the current cluster state. Hence, we assume that by properly fusing the cluster state, it will help the ordering agent make better decisions.

4.1.1 Squeeze-and-communicate mechanism. Considering that the machine numbers may vary in different clusters, and it will bring nonnegligible communication overhead to directly access the state of a large cluster, we design a *squeeze-and-communicate* module, making the communication process scalable. Specifically, as depicted in Figure 3, we first use two feed-forward networks (FFN) to extract the deep features of job and cluster states as follows:

$$\mathbf{S}^j = \text{relu}(\mathbf{O}^j\mathbf{W}_1 + \boldsymbol{b}_1), \quad \mathbf{S}^c = \text{relu}(\mathbf{O}^c\mathbf{W}_2 + \boldsymbol{b}_2), \tag{4}$$

where $\mathbf{S}^j$ and $\mathbf{S}^c$ are the extracted job and cluster features respectively, $\mathbf{W}_1 \in \mathbb{R}^{u\times e}$ and $\mathbf{W}_2 \in \mathbb{R}^{v\times e}$ are learnable weight matrices, e is the number of hidden units, and $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$ are the initial bias. After that, we generate the message to be delivered to the ordering agent by squeezing cluster representation with average pooling: $\widetilde{\boldsymbol{s}^c} = \text{AvgPool}(\mathbf{S}^c)$. Then we fuse the message $\widetilde{\boldsymbol{s}^c}$ into job representations with an adaptive gating mechanism as follows:

$$\begin{aligned} \widetilde{\mathbf{S}^j} &= \delta \cdot f(\widetilde{\boldsymbol{s}^c}) + (1-\delta)\cdot \mathbf{S}^j \\ \delta &= \sigma\big(\mathbf{W}_3[\mathbf{S}^j \oplus f(\widetilde{\boldsymbol{s}^c})] + \boldsymbol{b}_3\big), \end{aligned} \tag{5}$$

where $\mathbf{W}_3$ and $\boldsymbol{b}_3$ are learnable parameters, $\oplus$ denotes the concatenation operation, and f is the function to expand $\widetilde{\boldsymbol{s}^c}$ so that it can be concatenated with $\mathbf{S}^j$. In this way, the cluster state is conveyed to the ordering agent and integrated into job representations $\widetilde{\mathbf{S}^j}$.

4.1.2 Position-aware and self-attentive job ordering. After fusing the cluster state into job representations, the next step is to infer the job priorities. Intuitively, the ordering agent should be *position-aware*, as the jobs arrive at the pending list according to their submission order, which is a key factor for job scheduling [50]. Based on this intuition and inspired by the idea of pair-wise ranking [42], it is natural to leverage Transformer architecture to automatically quantify the pair-wise attentive correlations between jobs. First, we add the position embedding to $\widetilde{\mathbf{S}^j}$ as follows:

$$\begin{aligned} \mathbf{Z}^j &= \widetilde{\mathbf{S}^j} + PE(i) \\ PE(i)_{2k} = \sin\left(i/10000^{2k/e}\right), &\quad PE(i)_{2k+1} = \cos\left(i/10000^{2k/e}\right), \end{aligned}$$

where i is the position index (*i.e.,* the job submission order), and k is the dimension index. By applying the multi-head self-attention mechanism on $\mathbf{Z}^j$, the pair-wise attentive correlations are integrated into job representations $\widehat{\mathbf{S}^j}$ as follows:

$$\begin{aligned} \widehat{\mathbf{S}^j} &= [head_1\mathbf{W}_1^O, head_2\mathbf{W}_2^O, \cdots, head_h\mathbf{W}_h^O] \\ head_i &= \text{Attention}(\mathbf{Z}^j\mathbf{W}_i^Q, \mathbf{Z}^j\mathbf{W}_i^K, \mathbf{Z}^j\mathbf{W}_i^V), \end{aligned} \tag{6}$$

where $\mathbf{W}_i^Q$, $\mathbf{W}_i^K$, $\mathbf{W}_i^V$ and $\mathbf{W}_i^O$ are the corresponding learnable parameters for the i-th attention head. The attention function is implemented by scaled dot-product operation:

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{e/h}})\mathbf{V}.$$

Considering that the priority scores are continuous scalars, we assume the action $\boldsymbol{a}^j$ of the ordering agent follows a latent Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}^j, \Sigma^j)$, and can be explored by sampling from $\mathcal{N}(\boldsymbol{\mu}^j, \Sigma^j)$ in training phase as follows:

$$\boldsymbol{a}^j \sim \mathcal{N}(\boldsymbol{\mu}^j, \Sigma^j) \tag{7}$$

$$\boldsymbol{\mu}^j = \tanh(\mathbf{W}_4\widehat{\mathbf{S}^j} + \boldsymbol{b}_4), \quad \Sigma^j = \sigma(\mathbf{W}_5\widehat{\mathbf{S}^j} + \boldsymbol{b}_5), \tag{8}$$

where the i-th element in $\boldsymbol{a}^j \in \mathbb{R}^N$ is the priority score of job j_i, and $\mathbf{W}_4$, $\mathbf{W}_5$, $\boldsymbol{b}_4$ and $\boldsymbol{b}_5$ are learnable parameters.

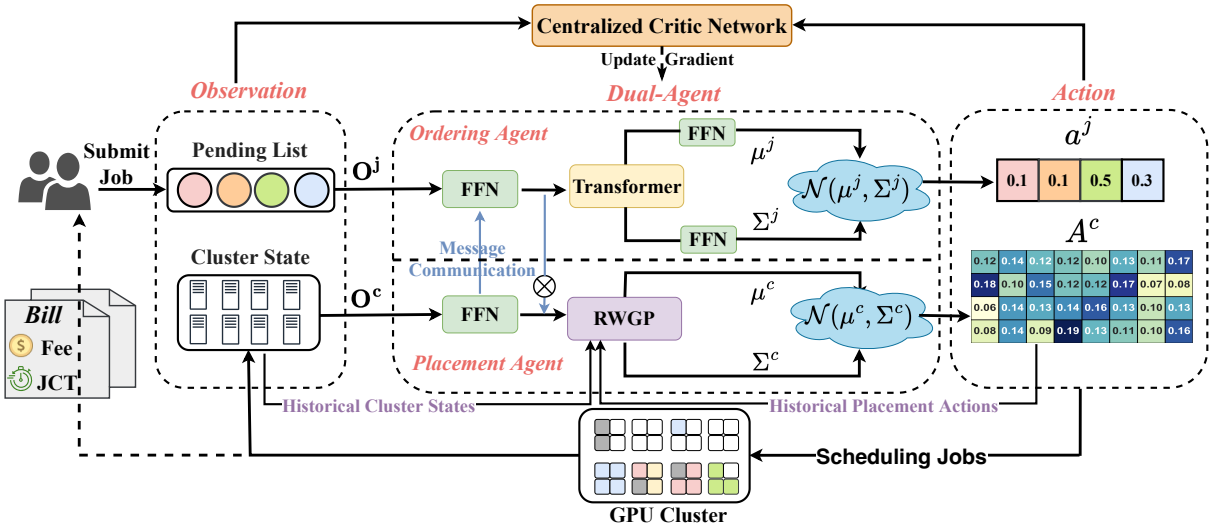


Figure 3: The overall architecture of our proposed DAS model. In this example, four jobs submitted by users are ready to be scheduled on the cluster with eight machines, where each machine is equipped with four GPUs.

4.2 Placement Agent with Random Walk Gaussian Process

The goal of the placement agent is to calculate the *affinity scores*, *i.e.,* action $\mathbf{A}^c \in \mathbb{R}^{N\times M}$, where a higher $\mathbf{A}^c_{ik}$ denotes that job j_i is more suitable to be placed on machine m_k. For the C_i running instances in job j_i, the scheduler places them on machines according to the descending order of $\mathbf{A}^c_i$. During the DDL job execution phase, the performance bottleneck lies in the machine with the worst computation and communication performance [11]. Moreover, as the resource contention introduced by co-located jobs is uncertain and with little prior knowledge, which is the main source of system uncertainty [57, 75] in the cluster environment, the machine performance is not deterministic and with uncertain fluctuation.

Behind this placement process, there are two intuitive principles: (1) the job instances should first be placed on the machines (*i.e.,* with higher affinity scores) with similar performances, *e.g.,* having similar free GPU, CPU, memory and bandwidth, to ensure that no machine could become the performance bottleneck. (2) The placement agent should be aware of the uncertain future performance fluctuation caused by resource contention. Based on the above principles, we propose a novel Random Walk Gaussian Process (RWGP) to predict the affinity scores. It models the topological performance similarities of machines with a stacked random walk kernel and perceives the uncertain performance fluctuation with Gaussian Process. The RWGP consists of the following four steps.

4.2.1 Job-aware cluster representations. First, considering that the cluster representations should be unique regarding different jobs as their resource requirements are distinct, we derive the *job-aware* cluster representations $\widehat{\mathbf{S}^c} = \widehat{\mathbf{S}^j} \otimes \mathbf{S}^c \in \mathbb{R}^{N\times M\times M}$, where $\widehat{\mathbf{S}^j}$ is the job representation (*i.e.,* Eq. 6), and $\otimes$ denotes the Kronecker product.

4.2.2 Kernel function in Euclidean space. In this step, we aim to tackle the performance uncertainty issue (*i.e.,* the second intuitive principle). Inspired by the Gaussian Process (GP) capturing uncertainty in the dynamic system by learning functions over mean and covariance [35, 65], we build a GP to infer the posterior probability distribution of placement action based on the input of historical cluster states and the output of historical placement actions:

$$\mathbf{X}_t = [\widehat{\mathbf{S}^c_{t-\tau}}, \widehat{\mathbf{S}^c_{t-\tau+1}}, \cdots, \widehat{\mathbf{S}^c_{t-1}}] \tag{9}$$

$$\mathbf{Y}_t = [\mathbf{A}^c_{t-\tau}, \mathbf{A}^c_{t-\tau+1}, \cdots, \mathbf{A}^c_{t-1}], \tag{10}$$

where τ is the length of historical data. The key to GP is the design of the kernel function, which should be able to capture the correlations between inputs effectively [72]. An easy way is to learn the correlations of historical cluster states in Euclidean space. Here, we adopt the Matern52 kernel [22] to measure their correlations as it is second-order derivable and is suitable to fit real functions [19, 70]. The Matern52 kernel can be achieved with the followings:

$$k_{\text{Matern}}(x_1, x_2) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\sqrt{2\nu}g\right)^{\nu} B_\nu\left(\sqrt{2\nu}g\right) \tag{11}$$

$$g(x_1, x_2) = (x_1 - x_2)^\top \rho^{-2}(x_1 - x_2),$$

where $\nu = \frac{5}{2}$ is a smoothness parameter, Γ and B_ν are the gamma function and the modified Bessel function, and g measures the distance between x_1 and x_2 scaled by the lengthscale parameter ρ. With the above GP formulation, we can estimate the posterior mean and covariance of $\mathbf{A}^c_t$ based on the input $\mathbf{S}^c_t$ with Eq. 1 and 2.

4.2.3 Random walk kernel in non-Euclidean space. In order to incorporate the topological structure of the cluster into GP, we abstract the cluster as a graph G_t at timestep t, where each node denotes a machine and the adjacency matrices are $\mathcal{P}_t = k_{\text{Matern}}(\widehat{\mathbf{S}^c_t}, \widehat{\mathbf{S}^c_t}) = [\mathbf{P}^{(1)}_t, \mathbf{P}^{(2)}_t, \cdots, \mathbf{P}^{(N)}_t]$, where $\mathbf{P}^{(i)}_t \in \mathbb{R}^{M\times M}$ is the adjacency matrix for job j_i. Suppose that there is a random walker, it selects the next visited machine node according to $\mathbf{P}^{(i)}$, where a larger $\mathbf{P}^{(i)}_{kl}$ denotes that machines m_k and m_l are more similar for placing job j_i and there is a higher probability to transit from m_k to m_l (following the

first intuitive principle). Motivated by the graph kernel theory [61] and deep kernel learning [67, 68], we propose a two-layer kernel that wraps kernel k_{Matern} (Eq. 11) with random walk kernel [32] to measure the transition correlations:

$$k_{\text{RW}}(G_1, G_2) = \sum_{i,k=1}^{|V_\times|} \left[\sum_{\ell=0}^{\infty} \lambda^\ell \mathbf{P}_\times^\ell \right]_{ik}, \tag{12}$$

where $V_\times$ and $\mathbf{P}_\times$ denote the Kronecker product of the nodes and adjacency matrices of G_1 and G_2, λ is a positive and real-valued weight, and ℓ denotes the length of random walking. In what followings, we show that the stacked kernel k_{RW} is symmetric and positive semi-definite, which satisfies the prerequisites of GP kernel [65]. The proof of Proposition 1 is in Appendix B.1.

Proposition 1. *Since Matern52 kernel k_{Matern} is symmetric and positive semi-definite [22], the new kernel k_{RW} that wraps it with a random walk kernel is also symmetric and positive semi-definite.*

To efficiently compute the stacked random walk kernel k_{RW}, we transform it to the exponential form with the following proposition (the proof can be found in Appendix B.2):

Proposition 2. *The stacked random walk kernel k_{RW} (*i.e.*, Eq. 12) is diagonalizable and can be transformed to the exponential series [21, 31] as $k_{RW}(G_1, G_2) = \exp(\beta \mathbf{P}_\times)$, where β is a positive parameter.*

4.2.4 Posterior distribution of RWGP. Based on the above kernel formulations and Eq. 1 and 2, we can derive the posterior mean $p(\boldsymbol{\mu}_t^c | \widehat{\mathbf{S}_t^c}, \mathbf{X}_t, \mathbf{Y}_t)$ and covariance $p(\Sigma_t^c | \widehat{\mathbf{S}_t^c}, \mathbf{X}_t, \mathbf{Y}_t)$ as follows:

$$\boldsymbol{\mu}_t^c = k_{\text{RW}}(\widehat{\mathbf{S}_t^c}, \mathbf{X}) \left[k_{\text{RW}}(\mathbf{X}_t, \mathbf{X}_t) + \sigma^2 \mathbf{I} \right]^{-1} \mathbf{Y}_t \tag{13}$$

$$\Sigma_t^c = k_{\text{RW}}(\widehat{\mathbf{S}_t^c}, \widehat{\mathbf{S}_t^c}) - k_{\text{RW}}(\widehat{\mathbf{S}_t^c}, \mathbf{X}_t) \left[k_{\text{RW}}(\mathbf{X}_t, \mathbf{X}_t) + \sigma^2 \mathbf{I} \right]^{-1} k_{\text{RW}}(\mathbf{X}_t, \widehat{\mathbf{S}_t^c}). \tag{14}$$

With this Random Walk Gaussian Process, the estimated posterior covariance (*i.e.,* Eq. 14) naturally reflects the system uncertainty [15, 34, 47] from a dynamic and topological view. The uncertainty awareness is crucial for the agent to explore the action space [36], and we sample the placement decision $\mathbf{A}_t^c$ from the posterior distribution:

$$\mathbf{A}_t^c \sim \mathcal{N}(\boldsymbol{\mu}_t^c, \Sigma_t^c).$$

In this way, the action with a higher uncertainty can be effectively explored with the RWGP, while the action with a higher predictive value is well exploited. Furthermore, the job-specific cluster topological structure can be integrated into the learning process, which encourages the random walker to select the machines with similar performances regarding each job. So far we have obtained the ordering and placement action decisions, and the scheduler can allocate according resources for each job.

4.3 Optimizing via MAPPO

In the previous sections, we have presented the dual-agent structure. In this section, we introduce the centralized critic network and the MAPPO [77] algorithm we use to optimize the networks.

The architecture of the centralized critic network is designed as a simple multi-layer perception (MLP), taking the observations and actions as inputs to estimate the Q-value as follows:

$$Q_\phi(\mathbf{O}_t^j, \mathbf{O}_t^c, \boldsymbol{a}_t^j, \mathbf{A}_t^c) = \text{MLP}\left(\left[\mathbf{O}_t^j \oplus \mathbf{O}_t^c \oplus \boldsymbol{a}_t^j \oplus \mathbf{A}_t^c \right] \right), \tag{15}$$

where ϕ is the model parameter of the critic network, and $\oplus$ denotes the concatenation operation. The centralized critic network can motivate the dual-agent to learn cooperation by perceiving a more comprehensive landscape. It can be optimized with Temporal-Difference Error (TD Error) [46] as follows (for brevity, we use π_θ to denote the joint ordering policy π_{θ^j} and placement policy π_{θ^c} in the followings, where θ^j and θ^c are the policies parameters):

$$\mathcal{L}_t^{TD} = \left(Q_\phi(\mathbf{O}_t^j, \mathbf{O}_t^c, \boldsymbol{a}_t^j, \mathbf{A}_t^c) - y_t \right)^2 \tag{16}$$

$$y_t = r_t + \gamma Q_{\phi_{old}} \left(\mathbf{O}_{t+1}^j, \mathbf{O}_{t+1}^c, \pi_{\theta_{old}}(\mathbf{O}_{t+1}^j, \mathbf{O}_{t+1}^c) \right),$$

where Q_ϕ and $Q_{\phi_{old}}$ denote the critic and target critic networks, $\pi_{\theta_{old}}$ is the target agent policy, θ is the model parameter of the dual-agent, and γ is the reward discount factor. The parameters θ_{old} and ϕ_{old} of target networks are copied from θ and ϕ every z timesteps, respectively.

The objective function of the dual-agent comprises policy gradient and entropy loss, which can be specified as follows:

$$\mathcal{L}^{Actor} = -\left(\mathcal{L}^{PG} + \lambda H(\pi_\theta) \right) \tag{17}$$

$$\mathcal{L}^{PG} = \mathbb{E}_t \left[\min \left(\text{ratio}, \text{clip}(\text{ratio}, 1-\epsilon, 1+\epsilon) \right) B_t \right]$$

$$\text{ratio} = \mathbb{E}_t \left[\exp \left(\log(\pi_\theta) - \log(\pi_{\theta_{old}}) \right) \right], \tag{18}$$

where H denotes the entropy of the given policy distribution, λ is the weight of entropy loss, ϵ controls the bound of the difference between policy π_θ and target policy $\pi_{\theta_{old}}$, and B_t is the Generalized Advantage Estimation [48] that measures the advantage of taking an action, which can be derived by the Q-value (in Eq. 15). For the term of log-probability of policy distributions in Eq. 18, since the prior distributions of ordering and placement actions are identical, *i.e.,* following Gaussian distribution, we only present the log-probability of π_{θ^j} as an explanation, which can be derived by substituting the mean $\boldsymbol{\mu}^j$, covariance Σ^j (in Eq. 8) and action $\boldsymbol{a}^j$ (in Eq. 7) into the following equation:

$$\log(\pi_{\theta^j}) = -\frac{1}{2} \left[(\boldsymbol{a}^j - \boldsymbol{\mu}^j)^2 + N \log(2\pi) \right] - \log(|\Sigma^j|).$$

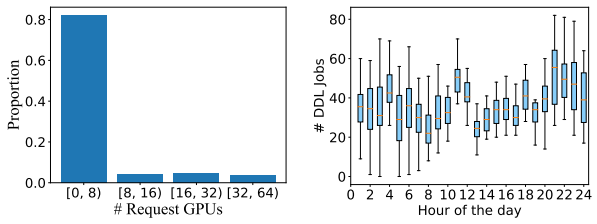
By minimizing the TD Error (Eq. 16) and actor loss (Eq. 17), the critic network is optimized to accurately estimate the Q-value, and the agents try to maximize the final return, *i.e.,* cumulated reward. The complete learning process of DAS is detailed in Appendix C.

5 EXPERIMENTS

5.1 Experimental Setup

5.1.1 Simulation environment. In order to efficiently evaluate the performances of baselines and our model, we adopt a modified GPU cluster scheduling simulator that has been used in many DDL scheduling research work [20, 27, 76]. It continuously receives and schedules DDL jobs submitted by users, and then reports the JCT and duration of each job. The training fee can be obtained from job duration and resource requirements as introduced in Section 3.

5.1.2 Dataset. The DDL jobs in our experiments are from Alibaba Cluster Trace [63], which is recorded in a real-world production GPU cluster over a two-month period. In this dataset, each entry contains the basic information of a DDL job, *i.e.,* the number of running instances, submission time, job duration, required GPU,



(a) Requested GPU number. (b) Number of submitted jobs.

Figure 4: Statistical information of our DDL job dataset.

CPU, and memory. In the pre-processing stage, we sort all the jobs according to the increasing order of their submission time, and then prune the jobs that are too large to fit into the clusters. We select the first 20,000 jobs, and split them into training and test sets with a ratio of 8 : 2. We present their statistical information in Figure 4. From Figure 4(a), we can see that the majority of jobs need a small number of GPUs, while a certain proportion of jobs are big-model and require more than 32 GPUs. Besides, we show the distribution of the number of submitted jobs by the day hours in Figure 4(b).

5.1.3 Experiment setting. To evaluate the performance of our model on clusters with different scales, we initialize three clusters with various numbers of machines, where each machine is equipped with the same amount of V100 GPU, CPU, and memory. Specifically, the *Small* cluster has 8 machines with 64 GPUs; the *Medium* cluster has 16 machines with 128 GPUs; the *Large* cluster has 32 machines with 256 GPUs. For the parameters in our model, the number of hidden units e is set as 64. According to the price provided by Alibaba Cloud [4] in August 2022, the monetary cost p of using a V100 GPU per hour is 2.84 dollars. The complete hyper-parameter settings of our model are summarized in Appendix D.

5.1.4 Evaluation metrics. We adopt the average JCT and training fee of jobs in the test dataset as metrics to evaluate the performances of different models, which can be specified as follows:

$$JCT = \sum_{i=0}^{|\mathcal{J}|} \frac{r_i^{jct}}{|\mathcal{J}|}, \qquad Fee = \sum_{i=0}^{|\mathcal{J}|} \frac{r_i^{fee}}{|\mathcal{J}|},$$

where $\mathcal{J}$ is the test job set, and the definitions of r_i^{jct} and r_i^{fee} can be found in Section 3. Please note that the lower these metrics are, the performance is better.

5.1.5 Baselines. We compare our model with baselines from three categories, namely (1) heuristic-based schedulers, (2) meta-heuristic-based schedulers, and (3) RL-based schedulers, including:

• **FIFO-FirstFit** first schedules the earliest come job [12] and places it on the available machine first retrieved [7].

• **FIFO-LoadBalance** first schedules the earliest come job and places it on the machine with the least workload [58].

• **DRF-FirstFit** first schedules the job with the minimum dominant resource share [23] and places it on the available machine first retrieved.

• **DRF-LoadBalance** first schedules the job with the minimum dominant resource share and places it on the machine with the least workload.

• **Tetris** [24] computes the dot product of normalized required resources of jobs and available resources of machines to make scheduling decisions.

• **MALO** [1] uses a hybrid ant-lion optimization algorithm based on an elite differential evolutionary algorithm to search for an optimal multi-objective job scheduling policy.

• **MOSA** [51] uses a two-step evolutionary approach to search for the optimal ordering and placement strategy.

• **DL**^{2} [43] combines offline supervised learning and online RL for faster convergence and better *ordering decisions*.

• **RIFLING** [13] groups job states with k-means to obtain robust job representation and reduce state space, and then uses RL to make *placement decisions* for jobs.

More details about baselines can be found in related work. For fair comparisons, we use the same features for all the RL-based baselines (heuristic- and meta-heuristic-based methods can only use partial features due to the limitations of their rules), and make slight modifications for them to fit our scenario. For example, RIFLING only adopts the job states to make decisions, while we complement it by concatenating with the cluster states. To reproduce the results of the baselines, we report their parameter settings in Appendix D.

5.2 Performance Comparison

We present the JCT and Fee of all baselines and our method on Small, Medium, and Large cluster settings in Table 1.

Among all these baselines, the heuristic-based schedulers perform the worst, as they use hand-crafted heuristic rules, which cannot adapt to dynamic workloads well. Meta-heuristic-based schedulers perform better, as they explore the solution space by injecting randomness. However, these exploration strategies make them much less efficient (about *50-x slower* than DAS in our experiments), and inapplicable in real-world production clusters. The RL-based models can be optimized to dynamically adapt to the cluster environments and time-variant job workloads during the learning process. And RIFLING achieves the lowest Fee as it focuses on finding the best *placement decision*, which can reduce the job duration. However, its JCT is slightly larger than DL^2 which focuses on deciding the best *ordering decision* and has a lower time cost of waiting in the pending list. Please note that the JCT and Fee have been proven to be non-cooperative and expose a hard-tradeoff [79] (see Appendix A), which means that they may not decrease at the same time as they are correlated but distinguishing objectives, *i.e.,* JCT optimizes all the jobs fairly, while Fee tends to optimize the high-value jobs with more GPUs and longer duration.

As a comparison, our DAS model outperforms all the competitors in all cases. We can observe that the improvement of JCT decreases as the cluster scale becomes larger. This is because for the same job workload, the Large cluster has relatively enough resources and hence the different scheduling decisions have less impact on the metric. It implies that our approach is especially contributable to the cluster with a heavy workload. We further conduct an experiment for the same cluster but with different workloads in Appendix E. Moreover, despite the non-cooperative and hard-tradeoff natures of JCT and Fee, our method outperforms the baselines on both metrics

[4] https://www.alibabacloud.com/en/product/gpu/pricing

Table 1: Performance comparisons with baselines and ablation study for four variants of our method. The JCT is in minutes and Fee is in dollars. The best and second best are in bold and underline, respectively. Please note that the metrics are presented at job level, and even marginal improvements can lead to significant benefits regarding all the clusters running on the cloud.

Algorithms		Small		Medium		Large		Average	
		JCT	Fee	JCT	Fee	JCT	Fee	JCT	Fee
Heuristic-based	FIFO-FirstFit	720.228	51.685	171.282	51.684	110.316	51.219	333.942	51.529
	FIFO-Load Balance	715.776	51.559	162.102	50.900	112.290	51.112	330.054	51.189
	DRF-FirstFit	862.242	52.055	164.778	51.354	109.098	52.325	378.708	51.911
	DRF-Load Balance	852.984	51.798	173.106	51.908	113.556	51.924	379.884	51.877
	Tetris	813.480	51.668	176.688	52.567	113.364	53.109	367.842	52.448
Meta-heuristic-based	MALO	708.858	51.675	182.862	51.129	111.660	51.233	334.458	51.346
	MOSA	759.222	51.783	173.400	51.998	109.422	51.933	347.346	51.905
RL-based	DL^2	721.083	51.166	160.100	50.970	110.215	51.080	330.468	51.072
	RIFLING	717.576	51.160	176.676	50.980	109.872	51.039	334.708	51.060
Ours	DA	697.732	51.106	173.354	51.097	108.821	51.025	326.636	51.076
	DA+MC	680.861	50.952	163.456	50.941	109.854	50.701	318.057	50.865
	DA+RWGP	686.894	51.029	157.071	50.856	**107.896**	50.439	317.287	50.684
	DA+MC+RWGP (DAS)	**638.046**	**50.484**	**155.003**	**50.019**	108.700	**50.354**	**300.584**	**50.286**

in all cases, which further shows the effectiveness of the designed reward function (*i.e.,* Eq. 3) and the overall learning framework.

Although it seems the improvement of Fee is not significant compared with the best baseline RIFLING, which achieves the lowest Fee and competitive JCT among all baselines, it still will save more than ten thousand dollars for users per day as over 17 thousand jobs are submitted to a production cluster in Alibaba Cloud every day [63]. Please note that this saving is only for a single cluster and the aggregate savings will be significant regarding all the clusters running in Alibaba Cloud. The optimized JCT also allows service providers to reduce power costs and complete more jobs in the same time. Cheap and fast computing services will also attract more users and improve potential income. We conduct the Wilcoxon signed-rank test [69] on the JCT and Fee metrics of our method and RIFLING on the three cluster environments, and the p-values of JCT and Fee metrics are less than 0.05 and 0.10 respectively, which also proves that the improvements are statistically significant (the detailed results and analysis can be referred in Appendix G).

5.3 Ablation Study

To examine the effects of different modules, four variants of our method are compared, including: (1) **DA** denotes only using the dual-agent formulation (introduced in Section 3) and MAPPO as the training framework; (2) **DA+MC** denotes additionally using the message communication mechanism (in Section 4.1) for the ordering agent based on DA; (3) **DA+RWGP** denotes additionally using the Random Walk Gaussian Process (in Section 4.2) for the placement agent based on DA; (4) **DA+MC+RWGP** denotes our complete model, *i.e.,* DAS.

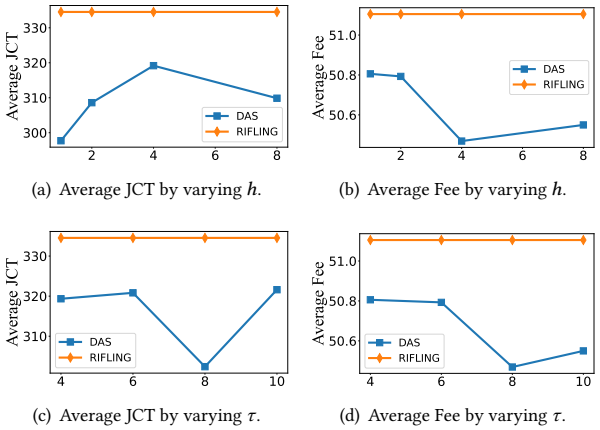
From the bottom four rows in Table 1, we can see that DA can achieve better JCT and competitive Fee compared with RL-based baselines, but the improvement is minor as it is just a simple combination of the two agents. With the squeeze-and-communicate mechanism, DA+MC performs better than DA, which proves that the communication can improve cooperation between agents and help them collaboratively make better decisions. By incorporating the RWGP, it can also lead to improvements on both metrics. It shows that the RWGP can well model the topological performance similarities and uncertain performance fluctuation, which are necessary for well scheduling DDL jobs. The DA+MC+RWGP, *i.e.,* our complete DAS model, combines the message communication and RWGP, and achieves the best performances among all variants on average, which further proves the effectiveness of each module.

5.4 Performance Tuning

In this part, we examine the robustness of our model. Specifically, we analyze the model performances influenced by two important parameters in the ordering and placement agents respectively, *i.e.,* the number of attention head h (in Eq. 6) and the length of historical data τ (in Eq. 9 and 10) in RWGP. We average the JCT and Fee metrics on the three cluster environments by varying the two parameters and present the results in Figure 5. For simplicity, we only incorporate RIFLING, which achieves the lowest Fee and competitive JCT among all baselines from Table 1, as a comparison.

In the *ordering agent*, the h attention heads learn the position-aware and pair-wise attention from different weight space views. We vary h in the set of $\{1, 2, 4, 8\}$. It can be observed from Figure 5(a) and 5(b) that our model is consistently better than RIFLING at the four choices, and achieves the best Fee when $h = 4$. However, the JCT and Fee show opposite trends. Recall that fairly scheduling all jobs can obtain a lower average JCT while prioritizing large and long-duration jobs (*i.e.,* high-value jobs) can result in an optimized Fee. It indicates that a proper number of attention heads (*i.e.,* $h = 4$) in the job ordering phase can help distinguish jobs with different characteristics and first schedule high-value jobs, but will introduce more noise when optimizing JCT as it treats jobs equally.

In the *placement agent*, the RWGP derives the posterior distribution of placement actions based on the τ-length of historical cluster states and actions. We vary τ in the set of $\{4, 6, 8, 10\}$. As we can see from Figure 5(c) and 5(d), the JCT and Fee of our model consistently



(a) Average JCT by varying h. (b) Average Fee by varying h.

(c) Average JCT by varying τ. (d) Average Fee by varying τ.

Figure 5: Performance tuning by varying h and τ. Our method consistently outperforms RIFLING on both metrics.

outperform RIFLING, and reach their lowest point when $\tau = 8$. It indicates that a small τ cannot provide adequate historical knowledge, while a larger τ may introduce stale and noisy information when making placement decisions.

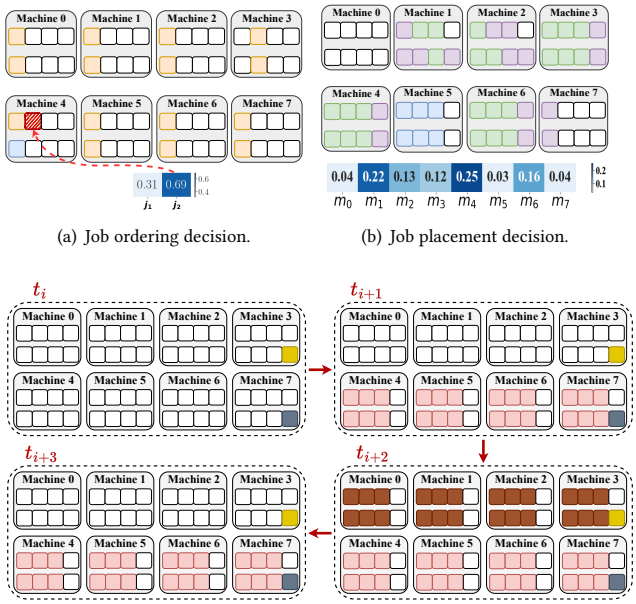
5.5 Case Study

When utilizing the strong decision-making ability of RL, the poor explainability of its black-box policy limits its credibility. In this section, we provide qualitative case studies based on experiments conducted on the *Small* cluster (8 machines, each equipped with 8 GPUs) to add the credibility and feasibility of our method in the real world and also provide enlightenment to the experts and benefit future decision-making.

First, we visualize the cluster state and heatmap of the *ordering action* in Figure 6(a). From the heatmap, we can see that job j_2 (in red) has a significantly higher probability (*i.e.,* 0.69) of being scheduled first. By inspecting into the profile of j_1, we find that it needs 8 instances and each requires 7 GPUs, which cannot be satisfied by the available resources of the current cluster. It implies that the communication mechanism is able to convey the cluster state to the ordering agent, so as to make proper ordering decisions.

Furthermore, Figure 6(b) demonstrates the cluster state and heatmap of *placement action* for job j_5 (in purple). It can be observed that j_5 prefers to be placed on machine m_1, m_2, m_3, m_4 and m_6, which have similar workload, *i.e.,* only running job j_3 (in green). m_5 has the smallest affinity score as there might be resource contention to place j_5 with j_4 (in blue) on m_5, and the workloads and performances of placed machines could be imbalanced a lot. This example shows that the RWGP is able to detect the performance similarities of machines and the potential resource contention.

In addition, we present a case of four-step scheduling process in Figure 6(c) to illustrate the superiority of our model over the baselines. In which, two jobs are running on machines 3 and 7 at t_i respectively, and the job j_6 (in grey) on machine 7 is more CPU-intensive than the job j_7 (in yellow) on machine 3. At t_{i+1}, our model schedules the new job j_8 (in pink) with smaller CPU requirements on machines 4-7, freeing up machines 0-3 for the upcoming



(a) Job ordering decision. (b) Job placement decision.

(c) Job scheduling decisions in four steps.

Figure 6: Case study on the Small cluster to show the effectiveness of the ordering and placement policies of our model.

CPU-intensive job j_9 (in brown) at t_{i+2}. However, the FirstFit- and LoadBalance-based methods will schedule job j_8 on machines 0-3 or 0-2 and 4 as they are first retrieved or with the least workload, which makes it infeasible to schedule and place job j_9. It is also hard for the other methods to make optimal scheduling decisions as they ignore to explicitly model the machine performance similarities and potential resource contention from future arrived and co-located jobs, which can be effectively tackled by our DAS model.

6 CONCLUSION

In this paper, we propose a dual-agent scheduler for DDL jobs to optimize the JCT and training fee. First, we abstract the ordering and placement sub-tasks as a dual-agent structure, and formulate the scheduling process as Dec-MDP. For the ordering agent, we enhance it with a scalable squeeze-and-communicate mechanism for better collaboration with the placement agent. A Transformer encoder is employed to model the arrival order and pair-wise correlations of pending jobs. For the placement agent, we propose a novel Random Walk Gaussian Process to learn the performance similarities and uncertain performance fluctuation of machines. Finally, the dual-agent and centralized critic network are jointly optimized with the MAPPO algorithm. Extensive experiments on the real-world production cluster trace demonstrate the effectiveness of our DAS model compared with nine baselines. In the future, we will explore to generalize our model to other scheduling and pricing scenarios.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their insightful comments on this paper. Zhen Xiao is the corresponding author.

REFERENCES

[1] Laith Abualigah and Ali Diabat. 2021. A novel hybrid antlion optimization algorithm for multi-objective task scheduling problems in cloud computing environments. *Cluster Computing* 24, 1 (2021), 205–223.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* (2014).

[3] Yixin Bao, Yanghua Peng, and Chuan Wu. 2022. Deep Learning-Based Job Placement in Distributed Machine Learning Clusters With Heterogeneous Workloads. *IEEE/ACM Transactions on Networking* (2022).

[4] Daniel Barkoczi and Mirta Galesic. 2016. Social learning strategies modify the effect of network structure on group performance. *Nature communications* 7, 1 (2016), 1–8.

[5] Raphen Becker, Shlomo Zilberstein, Victor Lesser, and Claudia V Goldman. 2004. Solving transition independent decentralized Markov decision processes. *Journal of Artificial Intelligence Research* 22 (2004), 423–455.

[6] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research* 27, 4 (2002), 819–840.

[7] Alan A Bertossi, Luigi V Mancini, and Federico Rossini. 1999. Fault-tolerant rate-monotonic first-fit scheduling in hard-real-time systems. *IEEE Transactions on Parallel and Distributed Systems* 10, 9 (1999), 934–945.

[8] Justin Boyan and Michael Littman. 1993. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems* 6 (1993).

[9] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[10] Yanshuai Cao, Marcus A Brubaker, David J Fleet, and Aaron Hertzmann. 2013. Efficient optimization for sparse Gaussian process regression. *Advances in Neural Information Processing Systems* 26 (2013).

[11] Tsung-Hui Chang, Wei-Cheng Liao, Mingyi Hong, and Xiangfeng Wang. 2016. Asynchronous distributed ADMM for large-scale optimization—Part II: Linear convergence analysis and numerical performance. *IEEE Transactions on Signal Processing* 64, 12 (2016), 3131–3144.

[12] Jilan Chen, Dan Wang, and Wenbing Zhao. 2013. A task scheduling algorithm for Hadoop platform. *Journal of Computers* 8, 4 (2013), 929–936.

[13] Zhaoyun Chen. 2022. RIFLING: A reinforcement learning-based GPU scheduler for deep learning research and development platforms. *Software: Practice and Experience* 52, 6 (2022), 1319–1336.

[14] Tyson Condie, Neil Conway, Peter Alvaro, Joseph M Hellerstein, Khaled Elmeleegy, and Russell Sears. 2010. MapReduce online.. In *Nsdi*, Vol. 10. 20.

[15] Giulio D'Agostini. 1994. On the use of the covariance matrix to fit correlated data. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 346, 1-2 (1994), 306–311.

[16] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[17] Jianzhong Du and Joseph Y-T Leung. 1989. Complexity of scheduling parallel task systems. *SIAM Journal on Discrete Mathematics* 2, 4 (1989), 473–487.

[18] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[19] Emily Fox and David Dunson. 2012. Multiresolution gaussian processes. *Advances in Neural Information Processing Systems* 25 (2012).

[20] Wei Gao, Zhisheng Ye, Peng Sun, Yonggang Wen, and Tianwei Zhang. 2021. Chronus: A novel deadline-aware scheduler for deep learning training jobs. In *Proceedings of the ACM Symposium on Cloud Computing*. 609–623.

[21] Thomas Gärtner, Peter Flach, and Stefan Wrobel. 2003. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*. Springer, 129–143.

[22] Marc G Genton. 2001. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research* 2, Dec (2001), 299–312.

[23] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica. 2011. Dominant resource fairness: Fair allocation of multiple resource types. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*.

[24] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. 2014. Multi-resource packing for cluster schedulers. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 455–466.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 770–778.

[26] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D Joseph, Randy Katz, Scott Shenker, and Ion Stoica. 2011. Mesos: A Platform for {Fine-Grained} Resource Sharing in the Data Center. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*.

[27] Qinghao Hu, Peng Sun, Shengen Yan, Yonggang Wen, and Tianwei Zhang. 2021. Characterization and prediction of deep learning workloads in large-scale gpu datacenters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–15.

[28] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. 2007. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*. 59–72.

[29] Zhihao Jia, Matei Zaharia, and Alex Aiken. 2019. Beyond Data and Model Parallelism for Deep Neural Networks. *Proceedings of Machine Learning and Systems* 1 (2019), 1–13.

[30] Marc Kac and AJF Siegert. 1947. An explicit representation of a stationary Gaussian process. *The Annals of Mathematical Statistics* 18, 3 (1947), 438–442.

[31] Janis Kalofolias, Pascal Welke, and Jilles Vreeken. 2021. SUSAN: The Structural Similarity Random Walk Kernel. In *Proceedings of the 2021 SIAM International Conference on Data Mining (SDM)*. SIAM, 298–306.

[32] U Kang, Hanghang Tong, and Jimeng Sun. 2012. Fast random walk graph kernel. In *Proceedings of the 2012 SIAM international conference on data mining*. SIAM, 828–838.

[33] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[34] Arkadiusz Lewandowski, Dylan F Williams, Paul D Hale, Jack CM Wang, and Andrew Dienstfrey. 2010. Covariance-based vector-network-analyzer uncertainty analysis for time-and frequency-domain measurements. *IEEE Transactions on Microwave Theory and Techniques* 58, 7 (2010), 1877–1886.

[35] Zhao-Yang Liu, Shao-Yuan Li, Songcan Chen, Yao Hu, and Sheng-Jun Huang. 2020. Uncertainty aware graph gaussian process for semi-supervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 4957–4964.

[36] Owen Lockwood and Mei Si. 2022. A Review of Uncertainty for Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 18. 155–162.

[37] Dinh The Luc. 2008. Pareto optimality. *Pareto optimality, game theory and equilibria* (2008), 481–515.

[38] J MacQueen. 1967. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*. 281–297.

[39] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. 2016. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*. 50–56.

[40] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, Zhibo Gong, and Yan Ni. 2020. Learning agent communication under limited bandwidth by message pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5142–5149.

[41] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, Zhibo Gong, and Yan Ni. 2020. Learning multi-agent communication with double attentional deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems* 34, 1 (2020), 1–34.

[42] Sahand Negahban, Sewoong Oh, and Devavrat Shah. 2012. Iterative ranking from pair-wise comparisons. *Advances in neural information processing systems* 25 (2012).

[43] Yanghua Peng, Yixin Bao, Yangrui Chen, Chuan Wu, Chen Meng, and Wei Lin. 2021. Dl2: A deep learning-driven scheduler for deep learning clusters. *IEEE Transactions on Parallel and Distributed Systems* 32, 8 (2021), 1947–1960.

[44] Roberta Raileanu, Emily Denton, Arthur Szlam, and Rob Fergus. 2018. Modeling others using oneself in multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 4257–4266.

[45] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 4295–4304.

[46] Edmund T Rolls, Ciara McCabe, and Jerome Redoute. 2008. Expected value, reward outcome, and temporal difference error representations in a probabilistic decision task. *Cerebral cortex* 18, 3 (2008), 652–663.

[47] Rebecca L Russell and Christopher Reale. 2021. Multivariate uncertainty in deep learning. *IEEE Transactions on Neural Networks and Learning Systems* (2021).

[48] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[49] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[50] Uwe Schwiegelshohn and Ramin Yahyapour. 1998. Analysis of first-come-first-serve parallel job scheduling. In *SODA*, Vol. 98. Citeseer, 629–638.

[51] Norelhouda Sekkal and Fayçal Belkaid. 2020. A multi-objective simulated annealing to solve an identical parallel machine scheduling problem with deterioration effect and resources consumption constraints. *Journal of Combinatorial Optimization* 40, 3 (2020), 660–696.

[52] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[53] Sainbayar Sukhbaatar, Rob Fergus, et al. 2016. Learning multiagent communication with backpropagation. *Advances in neural information processing systems* 29 (2016).

[54] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296* (2017).

[55] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[56] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[57] Andrei Tchernykh, Uwe Schwiegelsohn, Vassil Alexandrov, and El-ghazali Talbi. 2015. Towards understanding uncertainty in cloud computing resource provisioning. *Procedia Computer Science* 51 (2015), 1772–1781.

[58] Wenhong Tian, Yong Zhao, Yuanliang Zhong, Minxian Xu, and Chen Jing. 2011. A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters. In *2011 IEEE international conference on cloud computing and intelligence systems*. IEEE, 311–315.

[59] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[60] Vinod Kumar Vavilapalli, Arun C Murthy, Chris Douglas, Sharad Agarwal, Mahadev Konar, Robert Evans, Thomas Graves, Jason Lowe, Hitesh Shah, Siddharth Seth, et al. 2013. Apache hadoop yarn: Yet another resource negotiator. In *Proceedings of the 4th annual Symposium on Cloud Computing*. 1–16.

[61] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. 2010. Graph kernels. *Journal of Machine Learning Research* 11 (2010), 1201–1242.

[62] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[63] Qizhen Weng, Wencong Xiao, Yinghao Yu, Wei Wang, Cheng Wang, Jian He, Yong Li, Liping Zhang, Wei Lin, and Yu Ding. 2022. MLaaS in the Wild: Workload Analysis and Scheduling in Large-Scale Heterogeneous GPU Clusters. In *19th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 22)*.

[64] Marco A Wiering et al. 2000. Multi-agent reinforcement learning for traffic light control. In *Machine Learning: Proceedings of the Seventeenth International Conference (ICML'2000)*. 1151–1158.

[65] Christopher KI Williams and Carl Edward Rasmussen. 2006. *Gaussian processes for machine learning*. Vol. 2. MIT press Cambridge, MA.

[66] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[67] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. 2016. Deep kernel learning. In *Artificial intelligence and statistics*. PMLR, 370–378.

[68] Andrew G Wilson, Zhiting Hu, Russ R Salakhutdinov, and Eric P Xing. 2016. Stochastic variational deep kernel learning. *Advances in neural information processing systems* 29 (2016).

[69] Robert F Woolson. 2007. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials* (2007), 1–3.

[70] Zhiliang Wu, Yinchong Yang, Jindong Gu, and Volker Tresp. 2021. Quantifying predictive uncertainty in medical image analysis with deep kernel learning. In *2021 IEEE 9th International Conference on Healthcare Informatics (ICHI)*. IEEE, 63–72.

[71] Wencong Xiao, Shiru Ren, Yong Li, Yang Zhang, Pengyang Hou, Zhi Li, Yihui Feng, Wei Lin, and Yangqing Jia. 2020. {AntMan}: Dynamic Scaling on {GPU} Clusters for Deep Learning. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. 533–548.

[72] Mingzhe Xing, Shuqing Bian, Wayne Xin Zhao, Zhen Xiao, Xinji Luo, Cunxiang Yin, Jing Cai, and Yancheng He. 2021. Learning Reliable User Representations from Volatile and Sparse Data to Accurately Predict Customer Lifetime Value. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3806–3816.

[73] Mingzhe Xing, Hangyu Mao, and Zhen Xiao. 2022. Fast and Fine-grained Autoscaler for Streaming Jobs with Reinforcement Learning. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, Lud De Raedt (Ed.). International Joint Conferences on Artificial Intelligence Organization, 564–570. https://doi.org/10.24963/ijcai.2022/80 Main Track.

[74] Mingzhe Xing, Ziyun Wang, and Zhen Xiao. 2021. Analysis of Resource Management Methods Based on Reinforcement Learning. In *2021 International Conference on High Performance Big Data and Intelligent Systems (HPBD&IS)*. IEEE, 27–31.

[75] Jin Xu, Huiqun Yu, Guisheng Fan, and Jiayin Zhang. 2022. Uncertainty-aware scheduling of real-time workflows under deadline constraints on multi-cloud systems. *Concurrency and Computation: Practice and Experience* (2022), e7562.

[76] Zhisheng Ye, Peng Sun, Wei Gao, Tianwei Zhang, Xiaolin Wang, Shengen Yan, and Yingwei Luo. 2021. ASTRAEA: A Fair Deep Learning Scheduler for Multi-tenant GPU Clusters. *IEEE Transactions on Parallel and Distributed Systems* (2021).

[77] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. 2021. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955* (2021).

[78] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*.

[79] Hong Zhang, Yupeng Tang, Anurag Khandelwal, Jingrong Chen, and Ion Stoica. 2021. Caerus:{NIMBLE} Task Scheduling for Serverless Analytics. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. 653–669.

[80] Wenshuai Zhao, Jorge Peña Queralta, and Tomi Westerlund. 2020. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 737–744.

[81] Guangyao Zhou, Wenhong Tian, and Rajkumar Buyya. 2021. Deep reinforcement learning-based methods for resource scheduling in cloud computing: A review and future directions. *arXiv preprint arXiv:2105.04086* (2021).

APPENDIX

A ANALYSIS OF JCT AND TRAINING FEE

The JCT and training fee are non-cooperative metrics and expose a hard tradeoff [79]. In this part, we will give a non-formal proof. Let us consider two extreme and opposite scheduling methods, which are also used in production clusters as naive scheduling policies [14, 28, 78]. (1) **Lazy.** Each job in the pending list should wait to be scheduled until all the jobs running in the cluster finish. This scheduling rule will provide the utmost free resources and the slightest resource contention for each job, which can bring the lowest job duration and thus is *Fee-efficient*, but the JCT is almost the worst as the waiting time of each job is the summation of job duration of all previous running jobs. (2) **Eager.** The job will be scheduled whenever there are enough resources in the cluster. The eager approach, on the other hand, is *JCT-efficient* since it minimizes the waiting time in the pending list, but introduces a much higher Fee as the resource contention and communication cost are heavy. Aside from the above discussion, the non-cooperative and hard tradeoff natures can also be demonstrated in Table 1, where baselines can hardly achieve good performances on both metrics, while our model outperforms them on both JCT and Fee, which further shows the effectiveness of the designed reward function (*i.e.,* Eq. 3) and the overall learning framework. Please note that in this paper, we do not focus on the mutual relation and ideal balance of the two metrics, which can be explored with Pareto Optimality [37] and will be regarded as our future work. It is also worth noting that it is more proper to formulate the scheduling problem as Decentralized Partially Observable Markov Decision Process [6], but we use the Dec-MDP formulation for simplicity and clarity.

B PROOF OF PROPOSITIONS

B.1 Proof of Proposition 1

Proposition 1. *Since Matern52 kernel k_{Matern} is symmetric and positive semi-definite (PSD) [22], the new kernel k_{RW} that wraps it with random walk kernel is also symmetric and positive semi-definite.*

Proof. To prove that k_{RW} is PSD, it is enough to prove the eigenvalues ξ_i of $\mathbf{P}_\times^\ell$ are non-negative. Let v_i be the eigenvector of $\mathbf{P}_\times$ associated with eigenvalue δ_i, then

$$\begin{aligned}\mathbf{P}_\times^\ell v_i =& \mathbf{P}_\times^{\ell-1}(\mathbf{P}_\times v_i) = \mathbf{P}_\times^{\ell-1}(\delta_i v_i) = \delta_i \mathbf{P}_\times^{\ell-2}(\mathbf{P}_\times v_i)\\ &= \delta_i^2 \mathbf{P}_\times^{\ell-3} v_i = \cdots = \delta_i^{\ell-1}\mathbf{P}_\times v_i = \delta_i^\ell v_i.\end{aligned}$$

Since $\mathbf{P}_\times$ is PSD and its eigenvalues $\delta_i \geq 0$ for all i, the eigenvalues $\xi_i = \delta_i^\ell \geq 0$, and $\mathbf{P}_\times^\ell$ is proved to be PSD. Besides, as the power of a symmetric matrix is also symmetric, it is obvious that $\mathbf{P}_\times^\ell$ is symmetric. □

B.2 Proof of Proposition 2

Proposition 2. *The stacked random walk kernel k_{RW} (*i.e., *Eq. 12) is diagonalizable and can be transformed to the exponential series [21, 31] as $k_{RW}(G_1, G_2) = \exp(\beta \mathbf{P}_\times)$, where β is a positive parameter.*

Proof. From Proposition 1, $\mathbf{P}_\times$ is symmetric and PSD, thus can be diagonalized as $\mathbf{P}_\times = \mathbf{T}^{-1}\mathbf{D}\mathbf{T}$. The power of the matrix can be easily calculated as $\mathbf{P}_\times^\ell = (\mathbf{T}^{-1}\mathbf{D}\mathbf{T})^\ell = \mathbf{T}^{-1}\mathbf{D}^\ell\mathbf{T}$. Since the exponential of a square matrix $\mathbf{H}$ can be defined as $e^{\beta\mathbf{H}} = \lim_{n\to\infty}\sum_{i=0}^{n}\frac{(\beta\mathbf{H})^i}{i!}$, thus $e^{\beta\mathbf{P}_\times} = \mathbf{T}^{-1}e^{\beta\mathbf{D}}\mathbf{T}$, where $e^{\beta\mathbf{P}_\times}$ is calculated component-wise. □

C LEARNING ALGORITHM

We present the overall scheduling and learning process of our DAS model in Algorithm 1.

Algorithm 1 The learning process for the DAS model.

Initialize: Ordering and placement agents with model parameters $\theta = \{\theta^j, \theta^c\}$, the target agent networks $\theta^j_{old} = \theta^j$ and $\theta^c_{old} = \theta^c$, critic network ϕ, target critic network $\phi_{old} = \phi$; empty pending list and GPU cluster, timestep $t = 0$;

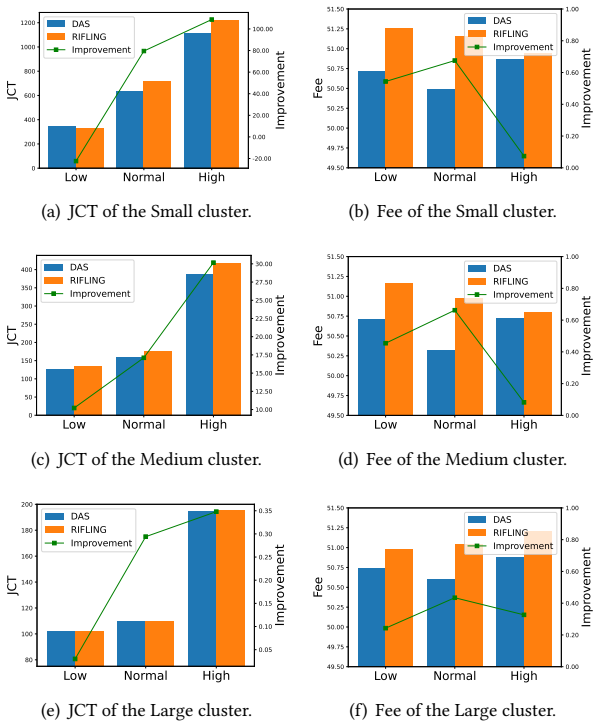
while Submitted jobs not finished **do**
  Obtain job and cluster observations $\mathbf{O}_t^j$ and $\mathbf{O}_t^c$;
  Obtain the posterior distributions of ordering and placement actions as Eq 8, 13 and 14:
    $\boldsymbol{\mu}^j, \Sigma^j, \boldsymbol{\mu}^c, \Sigma^c = \pi_\theta(\mathbf{O}^j, \mathbf{O}^c)$
  Sample ordering and placement actions:
    $\boldsymbol{a}^j \sim \mathcal{N}(\boldsymbol{\mu}^j, \Sigma^j), \quad \mathbf{A}^c \sim \mathcal{N}(\boldsymbol{\mu}^c, \Sigma^c)$
  Schedule jobs according to the action decisions. Record r_t^{jct} and r_t^{fee} of finished jobs, and compute r_t by Eq. 3;
  Update the centralized critic network:
    Minimize the loss $\mathcal{L}_t^{TD}$ computed by Eq. 16
  Update the ordering and placement agents:
    Minimize the loss $\mathcal{L}^{Actor}$ computed by Eq. 17
  if $t \,\%\, z == 0$ **then**
    Update target critic and agent networks:
      $\theta^j_{old} \leftarrow \theta_j, \quad \theta^c_{old} \leftarrow \theta_c, \quad \phi_{old} \leftarrow \phi$
  end if
  $t \leftarrow t + 1$
end while

D ADDITIONAL PARAMETER SETTINGS

To reproduce the results of all the comparison methods, we report their parameter settings in this part.

For the baselines, all the models have some parameters to be tuned except for the heuristic-based methods. Note that the RL-based methods employ PPO [49], the single-agent version of MAPPO, as the RL framework. We report their parameter settings used throughout the experiments in Table 2.

For our DAS model, apart from the parameters introduced in Section 5.1.3, we report the other hyper-parameters in the followings. The reward discount factor γ and weight of entropy loss λ are 0.98 and 0.03, respectively. In each episode, we randomly select a segment containing 1000 jobs, and the episode ends when all these jobs finish. The network is trained using Adam optimizer [33] with learning rate tuned in $\{0.0001, 0.00005, 0.00001, 0.000005, 0.000001\}$. For the number of pending jobs N, if more than N jobs exist in the pending list, we would sample N jobs, and pad the pending jobs with dummy jobs when there are less than N jobs in the pending list. The α in Eq. 3 is 0.1, the copy interval z of target networks is 2, and the bound ϵ between the policy and the target policy is 0.2.



(a) JCT of the Small cluster. (b) Fee of the Small cluster.

(c) JCT of the Medium cluster. (d) Fee of the Medium cluster.

(e) JCT of the Large cluster. (f) Fee of the Large cluster.

Figure 7: The performance comparisons by varying the workload, *i.e.,* the job arrival rate.

Table 2: Parameter settings of baselines.

Models	Settings
MALO	initial temperature T=400, cooling ratio P=0.99, number of iterations I=50
MOSA	number of iterations I=100, number of solutions S=50
DL^2	hidden_size=64, entropy_loss_weight=0.03, clip_ratio=0.2, γ=0.98, learning_rate=0.000001, Adam optimizer
RIFLING	num_clusters=$\sqrt{\frac{N}{2}}$, hidden_size=64, entropy_loss_weight=0.03, clip_ratio=0.2, γ=0.98, learning_rate=0.000001, Adam optimizer

E PERFORMANCE COMPARISON FOR DIFFERENT WORKLOADS

From Table 1, we can see that for the same workload, the improvement on JCT decreases as the cluster size becomes larger. It is mainly because regarding the same workload, the Large cluster has relatively enough resources, and hence different actions have less impact on the metrics. In this part, we aim to explore the performances of our DAS model and RIFLING regarding the same cluster but with different levels of workloads. Specifically, we set three levels of workloads, *i.e.,* Low, Normal and High, which stand for the 2-x slower, the original speed, and 2-x faster of the job arrival rate, and present the JCT, Fee, and relative improvements in Figure 7. As depicted, the JCT and Fee of DAS outperform RIFLING in most cases. The improvement of JCT increases as the workload becomes heavy, while Fee is improved the most for the Normal setting and is still better than RIFLING on the other settings. This experiment shows that our model can improve JCT a lot especially for the heavy workload. For the Fee, its improvement declines when the workload is heavy, which is most likely due to the high resource contention under heavy workload and the hard tradeoff of the metrics, but it still outperforms RIFLING on all workload settings.

F DISCUSSION OF SCHEDULER OVERHEAD

We investigate the inference speed of job ordering and placement decision-making by our model on an Ubuntu 16.04 server with 4 cores CPU, 32G memory, and an NVIDIA GTX 1080Ti GPU. The average inference time per job on the three cluster settings, *i.e.,* Small, Medium and Large, are 0.0132, 0.0138 and 0.0156 seconds, which is much less than the average JCT reported in Table 1 (more than 300 minutes) and can be ignored in the scheduling process. We believe that this finding supports the feasibility of using our proposed scheduling algorithm in practical scenarios. We would also like to highlight that additional speedup techniques can be applied to further reduce the scheduler overhead, *e.g.,* , data parallelism [29], or replacing the Gaussian Process in the RWGP with the sparse Gaussian Process [10]. Regarding the training overhead, since we trained our model on the simulator as other work, the time cost is also acceptable. With the sim-to-real algorithms [80], the trained policy can be adapted to the real scenario with a few fine-tuning overheads.

G EXTENDED PERFORMANCE COMPARISON

Table 3: The p-values of Wilcoxon signed-rank test conducted on the JCT and Fee metrics for our DAS model and RIFLING.

Small		Medium		Large	
JCT	Fee	JCT	Fee	JCT	Fee
0.0002	0.0588	0.0006	0.0570	0.04911	0.0792

Considering that the JCT and Fee metrics do not follow Normal distributions as most jobs are short-duration jobs and need no more than 4 GPUs (see Figure 4(a)), and a small proportion of jobs are large and long-running jobs, we adopt the Wilcoxon signed-rank test [69], a non-parametric significant test algorithm that has no assumption on the data distribution, to conduct the significant test. The results are shown in Table 3. We can see that the p-values of JCT and Fee metrics are less than 0.05 and 0.10 respectively, which further demonstrates that the improvements of our model are statistically significant.