Deep Reinforcement Learning-aided Multi-step Job Scheduling in Optical Data Center Networks

CHE-YU LIU[1], XIAOLIANG CHEN[2,*], ROBERTO PROIETTI[3], ZUQING ZHU[2], AND S. J. BEN YOO[4]

[1] *Department of Computer Science, University of California, Davis, CA 95616 USA*
[2,*] *School of Information Science and Technology, University of Science and Technology of China, Hefei, P. R. China*
[3] *Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Torino, Italy*
[4] *Department of Electrical and Computer Engineering, University of California, Davis, CA 95616 USA*
[*] *xlichen@ieee.org*

Compiled June 2, 2025

Orchestrating job scheduling and topology reconfiguration in optical data center networks (ODCNs) is essential for meeting the intensive communication demand by novel applications, such as distributed machine learning (ML) workloads. However, this task involves joint optimization of multi-dimensional resources that can barely be effectively addressed by simple rule-based policies. In this paper, we leverage the powerful state representation and self-learning capabilities from deep reinforcement learning (DRL) and propose a multi-step job schedule algorithm for ODCNs. Our design decomposes a job request into an ordered sequence of virtual machines (VMs) and the related bandwidth demand in between, and then make a DRL agent learn how to place the VMs sequentially. To do so, we feed the agent with global bandwidth and IT resource utilization state embedded with the previous VM allocation decisions in each step, and reward the agent with both team and individual incentives. The team reward encourages the agent to jointly optimize the VM placement in multiple steps to pursue successful provisioning of the job request, while the individual reward favors advantageous local placement decisions, *i.e.*, to prevent effective policies being overwhelmed by few subpar decisions. We also introduce a penalty on reconfiguration to balance between performance gains and reconfiguration overheads. Simulation results under various ODCN configurations and job loads show our proposal outperforms the existing heuristic solutions and reduces the job blocking probability and reconfiguration frequency by at least $7.35\times$ and $4.59\times$, respectively.

http://dx.doi.org/10.1364/ao.XX.XXXXXX

1. INTRODUCTION

Distributed machine learning (ML) workloads, particularly, large language models that comprise billions to trillions of parameters, involve distributed computing clusters executing parallel computations and frequent parameter synchronization, with individual connections generating data transfers at tens of Gbps. As a result, bandwidth bottlenecking has emerged as a critical factor limiting the performance of these communication-intensive applications, highlighting the need for advanced datacom architectures and flexible job scheduling strategies to orchestrate multi-dimensional resources (*e.g.*, CPU, memory, storage and bandwidth) [1].

Earlier works on job scheduling have reported significant advances in algorithm designs tailored to ML workloads [2–11]. However, they primarily focused on optimizing the allocation of computing resources while overlooking bandwidth limitation, which is key to accelerating the gradient synchronization process. Recently, researchers have begun to dig into the synergistic benefit of bandwidth-aware job scheduling [12–15], but predominantly considered traditional electronic packet-switched (EPS) data center networks (DCNs). These EPS-based DCNs face increasing challenges in terms of high latency and energy consumption, poor scalability and adaptability when handling today's ML workloads that double every few months.

In this context, optical data center networks (ODCNs) have emerged as promising solutions that naturally meet the above challenges [16, 17]. Built on commercially available optical circuit switching technologies, ODCNs can provide large-capacity and direct-connect communications between computing units while sustaining high energy efficiency (by largely eliminating power-hungry electrical-layer processing). Moreover, ODCNs allow for dynamic topology reconfiguration to adapt to the time-vary and skewed traffic patterns driven by the evolution of jobs [18–23]. Indeed, the benefits of ODCNs have been validated in production environment by leading high-tech enterprises like

Google and Meta. Nevertheless, job scheduling in reconfigurable ODCNs, *i.e.*, how to leverage the reconfigurability of ODCNs to assist in bandwidth-aware job placement, so that network-wide performance related to job throughput, latency and system overheads is maximized, remains underexplored.

Building on the above insights, this work aims at developing a comprehensive framework for the joint optimization of job scheduling and bandwidth allocation in reconfigurable ODCNs. Specifically, we extend our prior work in [24] and address the limitations of the simple heuristic policy therein by proposing a deep reinforcement learning (DRL)-based multi-step job scheduling design in ODCNs. Our contributions include:

- we propose a multi-step job scheduling design that tackles the scalability limitation of standard DRL approaches by decomposing the provisioning of a job into sequential virtual machine (VM) placement tasks;
- we present a comprehensive DRL agent design including the state representation, action space, reward function and training mechanism, and particularly, we devise a novel team and individual rewarding mechanism that drives the agent to optimize the job-level performance while avoiding discouraging performant local decision making;
- simulation results under various ODCN configurations (job loads and network scales) show superior performance of our proposal over the existing approaches.

The rest of the paper is organized as follows. Section 2 provides a brief review of related work on job scheduling and optical interconnects. Section 3 formulates the job scheduling problem in reconfigurable ODCNs. Section 4 presents the DRL-aided scheduling framework, including the agent design and reward structure. Section 5 provides evaluations of the proposed approach against the heuristic baselines under various job loads and network scales. Finally, Section 6 concludes the paper and outlines future research directions.

2. RELATED WORK

The optimization of job scheduling has been widely studied in literature, many of them targeting distributed ML workloads. Early research primarily focused on computational job scheduling to balance workload distribution and minimize training time. Zhang *et al.* introduced SLAQ, a cluster scheduler optimizing resource allocation for ML training by leveraging quality-runtime trade-offs, achieving up to 73% quality improvement and 44% training time reduction [2]. Bao *et al.* proposed online scheduling frameworks that minimize training time and improve fairness using workload predictions, enhancing job completion time and resource efficiency [3, 4]. Yu *et al.* developed an online scheduling algorithm optimizing locality and resource allocation in distributed ML systems [5], while Zhou *et al.* designed a dynamic scheduling algorithm to manage heterogeneous DML jobs across diverse resources, ensuring efficient job allocation [7]. Lu *et al.* leveraged reinforcement learning (RL) to dynamically adjust job assignments, significantly improving training efficiency and system performance over heuristic scheduling techniques [8]. Wang *et al.* integrates heuristic scheduling with DRL to optimize job completion time and model accuracy by prioritizing tasks based on spatial and temporal features [9]. It also implements load control and an optimal DML iteration stopping method, improving the model accuracy by up to 64% while reducing job completion time by 52%. TapFinger [10], a distributed scheduler for edge clusters, co-optimizes task placement and multi-resource allocation using multi-agent RL. By employing a heterogeneous graph attention network and Bayesian optimization, TapFinger achieves a 54.9% reduction in task completion time and improves resource efficiency. Luo *et al.* proposed A-SRPT, a prediction-assisted online scheduling algorithm for DML training in GPU clusters [11]. By modeling jobs as graphs representing heterogeneous DNNs and training configurations, A-SRPT minimizes inter-server communication overheads and predicts training iterations using a random forest model. It applies the shortest-remaining-processing-time-first strategy, leading to provable scheduling efficiency.

Beyond the management of computing resources, optimizing network bandwidth for DML job scheduling has gained attention due to the increasing data exchange required for distributed gradient synchronization. Zhang *et al.* introduced Prophet, a system that optimizes communication scheduling by leveraging predictable communication patterns, reducing network overhead and accelerating DML training [12]. Gu *et al.* proposed a network-aware scheduling framework for DML workloads in GPU clusters, enhancing job throughput and reducing network congestion through intelligent resource estimation and scheduling [13]. Guo *et al.* tackled joint job scheduling and bandwidth augmentation in hybrid data centers, transforming a complex mixed-integer nonlinear problem into an efficient Branch and Bound solution [14]. This method reduces job completion time by up to 10%, with performance gains influenced by data size. Similarly, Bi *et al.* presented AISAW, an adaptive scheduling algorithm for optimizing DL workload training in heterogeneous distributed systems. AISAW reduces job co-location interference, optimizes job-node matching, and minimizes migration overhead under bandwidth constraints. Experiments show AISAW cuts job completion time by 13.02% compared to the state-of-the-art algorithms [15]. However, these works depend on traditional electronic switching, which suffers from scalability, latency, and energy limitations, restricting their adaptability to growing ML workloads.

Traditional electrical packet switching architectures, constrained by their high latency, limited scalability, and energy inefficiencies, have spurred research into reconfigurable optical interconnects [16, 17]. Khani *et al.* introduced silicon photonics-based optical interconnects to enhance DML training, resulting in a 9.1× speedup [18]. Our prior work, Sl-hyper-flex, introduced a cognitive flexible-bandwidth optical interconnect that utilizes silicon photonic switch fabrics and self-supervised learning to optimize network performance [19]. It improves throughput by 1.62× and reduces end-to-end latency by up to 3.84×. Zheng *et al.* developed a fast, nondisruptive reconfiguration algorithm for topology migration in ODCNs, reducing traffic disruption by 74.5× [20]. Ottino *et al.* introduced a high-performance all-optical network for DML workloads, offering full-bisection bandwidth with nanosecond reconfiguration [21]. Xie *et al.* proposed P4INC-AOI, an in-network computing approach integrated in all-optical interconnects for DML training [22]. By optimizing AOI reconfiguration and resource allocation using both mixed-integer linear programming and heuristic designs, P4INC-AOI reduces job completion time by up to 56.34%. Guo *et al.* proposed an all-optical switching architecture to overcome communication bottlenecks in large-scale DML training, enabling nanosecond-level reconfiguration and seamless support for different parallelization strategies [23].

Aside from the aforementioned works that perform computing and bandwidth resource optimization, optimization of

job scheduling under various system constraints has gained increasing interest. Mahmoud *et al.* introduced a dynamic load-balancing technique for heterogeneous cloud environments, enhancing resource utilization and reducing execution time [25]. Yao *et al.* proposed a job duplication strategy optimizing workflow execution under budget constraints, balancing cost efficiency and performance [26]. Bal *et al.* presented a hybrid ML-based approach addressing resource allocation, security, and scheduling by dynamically adapting resource distribution based on workload demand and security constraints [27]. Mangalampalli *et al.* developed a DRL-based job scheduling framework for multi-cloud environments, efficiently allocating jobs based on resource availability, job demands, and system performance metrics [28]. Their results indicate that DRL-based scheduling enhances execution efficiency and scalability in resource-constrained environments. Despite these advancements, existing research has mainly focused on optimizing computation, network bandwidth, or system constraints individually. The interplay between job scheduling, adaptive bandwidth allocation, and topology reconfiguration in reconfigurable network architectures, particularly for ODCNs, remains an underexplored challenge.

3. NETWORK AND SERVICE MODELS

In this section, we describe the ODCN architecture and formally present the job scheduling model therein.

A. ODCN Architecture

We consider an optical circuit switching (OCS)-based reconfigurable ODCN employing the spine and leaf architecture. As shown in Fig. 1, the ODCN is composed of a set of racks, each housing a certain number of servers equipped with IT resources. The top-of-rack (ToR) switches are wired to a group of optical circuit switches (OCS's). By reconfigurations of the OCS switching matrices, the ODCN provides flexible-bandwidth optical interconnects to handle the time-varying and skewed traffic among the ToRs.

The ODCN exploits a software-defined networking (SDN)-based centralized control and management paradigm. In the control plane, the cloud service manager (CSMgr) communicates with the network manager (NMgr, which can be built on an SDN controller) to orchestrate the IT and bandwidth resources for optimized job scheduling. Specifically, the CSMgr receives job requests, each specifying a certain number of VMs with IT capacity requirements and a bandwidth demand matrix between VM pairs. Then, it inquires network information (*e.g.*, connectivity graph, bandwidth utilization) from the NMgr and combines them with the IT resource distribution in its sovereignty to form a global representation of the ODCN state. The ODCN state, together with the job requests, are forwarded to the DRL-aided job scheduling module for service provisioning calculation. The DRL module suggests the placement of the VMs, which in turn decides the additions of bandwidth demand among the ToRs. The CSMgr validates the feasibility of the suggested service schemes by verifying the IT resource viability and also requesting for bandwidth allocation by the NMgr. If either of the demanded IT and bandwidth resources is unavailable, the job is blocked, otherwise, the CSMgr works with the NMgr to commit the corresponding resource provisioning. Note that, the NMgr may perform event-driven (*e.g.*, upon failure of bandwidth allocation) reconfigurations of the optical interconnects to fit the evolving traffic demand in dynamic job scheduling. Such reconfigurations facilitates successful bandwidth allocation but introduces nonnegligible control plane overheads [17], and therefore, should be minimized. The CSMgr feeds back key performance metrics related to the service schemes, such as whether the jobs are successfully serviced, the reconfiguration costs and the perturbations to end-to-end performance (*e.g.*, latency), to the DRL module. Consequently, the DRL module oversees long trajectories of operation samples and learns successful state-aware provisioning policies from repeated trial and error.

B. Job Scheduling Problem

We model an ODCN as $G(V, W, E)$, where V and W denote the sets of racks and OCS's, respectively, and E describes the connectivity among the racks which is determined by the configuration of W. Specifically, each $e_{u,v} \in E$ gives the number of OCS connections configured for the communication from rack u to v and is bounded by $N \cdot |W|$, with N being the number of ports each rack uses to connect to an OCS. Let B be the capacity of an optical port/connection, then, the total bandwidth from rack u to v is $e_{u,v} \cdot B$. Each rack comprises M servers that provide IT resources in terms of computing units (*e.g.*, CPUs, GPUs), memory and disk. The IT resource of each server ϕ_i can be signified by a triple $C_i = [C_i^{\text{cu}}, C_i^{\text{mem}}, C_i^{\text{disk}}]$. Consequently, given a set of dynamically arriving distributed computing jobs R during a provisioning period, where each job $r_j(t_j, \{[Q_{j,k}^{\text{cu}}, Q_{j,k}^{\text{mem}}, Q_{j,k}^{\text{disk}}]_{k=1}^{K_j}\}, D_j, T_j) \in R$ is characterized by its arrival time t_j, a sequence of K_j VM requests with individual resource demands, a bandwidth demand matrix D_j describing inter-VM communication requirements, and a job duration T_j, our problem is formulated as: **optimizing the scheduling of R and the dynamic reconfiguration of W for maximizing the job acceptance ratio, while securing desirable quality-of-service (*e.g.*, end-to-end latency and packet loss rate) and system overheads, and complying with the resource constraints**. Table 1 summarizes the major symbols and notations used throughout the paper.

4. ALGORITHM DESIGN

A. Principle of DRL-aided Scheduling

Fig. 2 sketches the principle of DRL-aided multi-step job scheduling. Leveraging DRL's capability of optimizing long-term system performance, we first decompose a job r_j into a sequence of VM requests (①) and make the DRL agent decide the allocation of the VMs sequentially through multi-step decision making (②). Compared with one-shot job placement where the agent needs to handle a large and variable action space (which makes the successful training of the agent extremely challenging), this multi-step formulation significantly improves the model scalability and hereby ease policy learning. The VM requests can be ordered according to their IT resource requirements to facilitate servicing higher demanding VMs. Note that, the communication topology of the VMs in Fig. 2 is for illustrative purposes, and the actual topologies are task-specific, for instance, being a ring for a distributed ML task employing ring-based data parallelism. In each step k, the DRL agent reads state data $s_{j,k}$ encompassing the corresponding VM request and ODCN resource utilization like the residual IT resource of each rack and the available bandwidth among the racks, and generates with its policy neural network a provisioning policy $\pi_{j,k}(s_{j,k}, a)$ that guides the choice of a rack (*i.e.*, an action a) to which the VM should be deployed

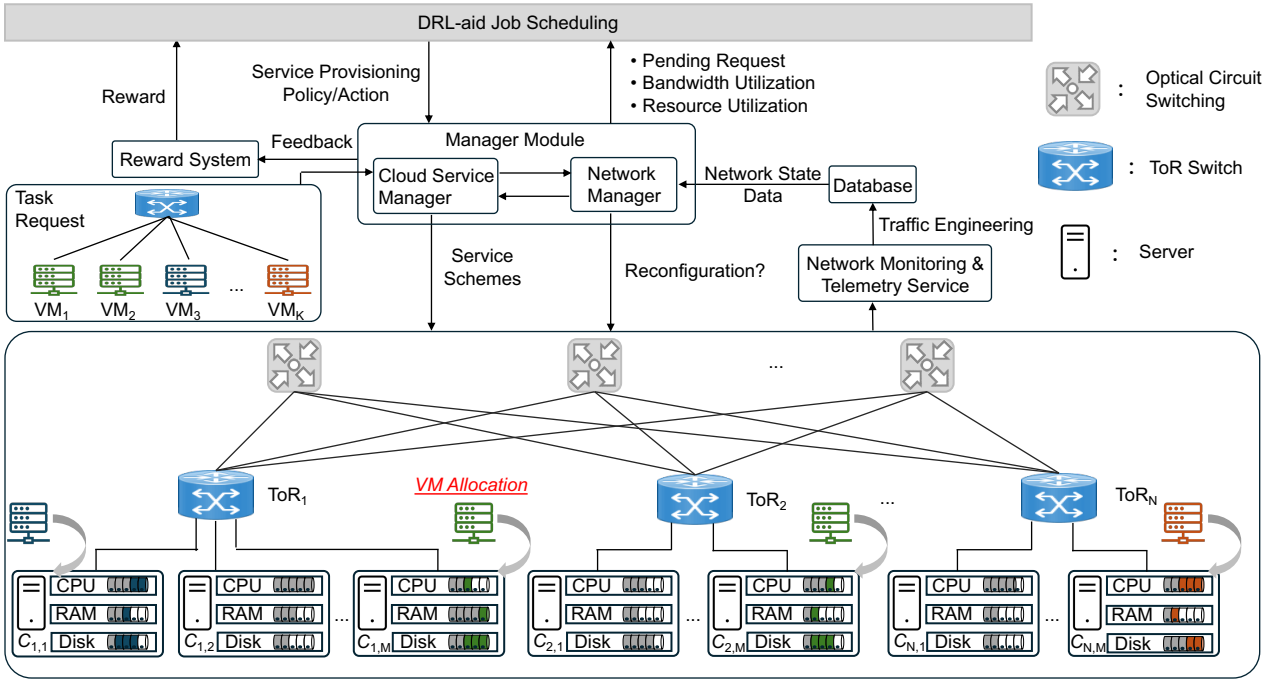

Fig. 1. ODCN architecture with DRL-aided job scheduling.

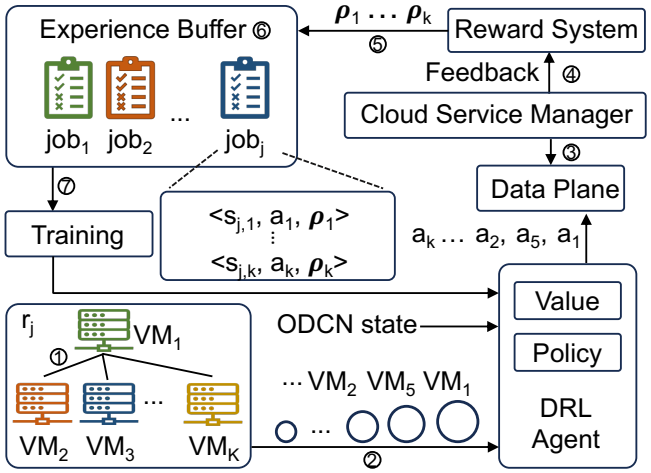

Fig. 2. Schematic of DRL-aided multi-step job scheduling.

in. Here, since each decision step $k \in [1, K_j]$ is tasked the placement of the k-th VM within job r_j, we use k indiscriminately to denote the indices of VM and scheduling step. The agent also employs a value neural network to later help judge the goodness of the agent's decisions by anticipating the future rewards. The CSMgr decides the VM allocation $a_{j,k} \sim \pi_{j,k}(s_{j,k}, a)$ (③) and provides feedback indicating whether sufficient resources have been withheld for the VM (④). If there is not sufficient IT resource for the VM, the provisioning phase is terminated and the job is marked as blocked. Otherwise, the agent proceeds to the next VM request. After all the VMs have been successfully processed, the cloud service manager interacts with the network manager to perform routing and bandwidth allocation for the job. If this process fails, the network manager retrieves the demand matrix and attempts to reconfigure the optical interconnect by augmenting the hot-spot links. The topology reconfiguration and bandwidth allocation results, together with the monitoring of end-to-end performance, such as latency and packet loss rates, are also fed back to the DRL agent (④). Afterward, the agent calls the reward system to translate these feedback into numerical rewards (⑤). Note that, every action taken in the multi-step decision making process is assigned a unique reward $\rho_{j,k}$ based on its individual performance (individual reward) as well as the global provisioning outcome (team reward) to encourage the agent pursuing job-level optimization. Next, the reward function will be detailed in the next section. The state, action and reward tuples $\{[s_{j,k}, a_{j,k}, \rho_{j,k}]_{k\in[1,K_j]}\}$ are stored in the experience buffer (⑥). Finally, every time a certain number of jobs have been serviced, we train the policy and value neural networks with the samples in the experience buffer by reinforcing advantageous actions (⑦). This way, the DRL agent is able to learn and adapt its policy from dynamic job scheduling operations.

B. DRL Agent Design

We design the DRL agent based on the advantage actor-critic (A2C) framework [29], which decouples policy learning and value estimation into two separate neural networks: a policy network (actor) and a value network (critic). The actor selects actions based on the current state, while the critic estimates the expected cumulative reward from that state. To improve sample efficiency and training stability, A2C uses the advantage function to evaluate how much better an action turns out to be than originally expected (by the critic, see Eq. 3). During training, actions with higher advantage values are reinforced. Next, we detail the key components of the agent.

1) *State*: Effective modeling of the ODCN states enables the agent perceiving global resource utilization and hereby learning effective scheduling policies. We construct $s_{j,k}$ for each r_j as a $1 \times 2|V|$ array, where for each rack u, we calculate the minimum residual IT and bandwidth capacities in percentage if the VM in step k were deployed (*i.e.*, hypothetical placement) in the

Table 1. Summary of notations.

Symbol	Description	Symbol	Description
V	Set of racks (ToRs) in an ODCN	W	Set of optical circuit switches (OCSs)
E	Set of optical connections among racks	$e_{u,v}$	Number of connections between racks u and v
B	Bandwidth capacity of a connection	M	Number of servers per rack
C_i	IT resource of server i: $[C_i^{\text{cu}}, C_i^{\text{mem}}, C_i^{\text{disk}}]$	R	Set of incoming job requests
r_j	The j-th job request	A_j	Set of VM allocation schemes for r_j
t_j	Arrival time of job r_j	T_j	Duration of job r_j
K_j	Number of VMs (steps) in job r_j	D_j	VM bandwidth demand matrix of job r_j
$[Q_{j,k}^{\text{cu}}, Q_{j,k}^{\text{mem}}, Q_{j,k}^{\text{disk}}]$	Resource requirement of VM k in job r_j	$s_{j,k}$	State representation for VM k in job r_j
$a_{j,k}$	Action (rack selection) for VM k in job r_j	$\rho_{j,k}$	Reward for the k-th decision in job r_j
Φ^{reconf}	Cost of topology reconfiguration	l	Average end-to-end latency
α	Scaling factor for latency term	γ	Discount factor for reward accumulation
β	Entropy regularization weight in policy loss	σ_n	Discounted cumulative reward for sample n
$\aleph_n$	Advantage estimate for sample n	a_n	Action taken in training sample n
$\mathcal{L}_{\theta_p}$	Policy network loss	$\mathcal{L}_{\theta_v}$	Value network loss
Π	Experience buffer storing training samples	L	Number of samples used in each training
ε, ε_0, $\varepsilon_{\min}$	Exploration rate, its variation step size and minimum value	$\pi_{\theta_p}(s_{j,k})$	VM allocation policy: probability distribution over actions given state $s_{j,k}$
$f_{\theta_p}(\cdot)$	Policy network	$f_{\theta_v}(\cdot)$	Value network

rack. Specifically, we adopt the balanced load strategy for VM allocation within a rack, and subsequently, the minimum IT resource is taken from minima of available computing units, memory and disk across all the servers in the rack to characterize the bottlenecking effect. The minimum bandwidth capacity is measured as the bottleneck of all the incoming and outgoing links of u. If any of these resource requirements is unmet by a rack, the corresponding feature value in $s_{j,k}$ is set as -1. Such encoding provides distinct signals for the agent to distinguish between infeasible solutions and situations where a rack can provide just enough resources (with feature values close or equal to 0). Then, $s_{j,k}$ will show an evident blocking case when all the racks have at least one feature value of -1.

2) *Action space*: The agent selects one from the $|V|$ racks as the host for each VM request. Therefore, policy $\pi_{j,k}(s_{j,k}, a)$ can be a probability of choosing action $a_{j,k} = a, a \in V$.

3) *Neural network structures*: The policy and value neural networks employ the same structure (*e.g.*, fully-connected or convolutional neural networks) for feature extraction, and a separate policy and value heads for outputting the provisioning policy and value estimation, respectively.

4) *Reward function*: We define the reward function as,

$$\rho_{j,k} = \begin{cases} -K_j, & \text{if } r_j \text{ is blocked in step } k, \\ 0.5, & \text{if } r_j \text{ is blocked in step } k' > k, \\ 1 + \frac{\alpha}{l}, & \text{if } r_j \text{ is serviced without reconfiguration,} \\ 1 + \frac{\alpha}{l} - \Phi^{\text{reconf}}, & \text{otherwise.} \end{cases} \tag{1}$$

where l denotes the average network-wide end-to-end latency in nanoseconds, Φ^{reconf} represents the additional cost associated with topology reconfiguration, and α is a scaling factor. Since multiple actions collectively determine the provisioning of a job, we differentiate the credit (*i.e.*, immediate reward) assigned to each action by rewarding the agent both individual and team incentives. In particular, we penalize an immediate blocking of a job by a reward of $-K_j$ (*Line* 1), while successful placement of VMs prior to job blocking still receives a positive but discounted reward of 0.5 (*Line* 2). Note that, if a job is blocked at step k, all subsequent samples are discarded. These two terms correspond to individual rewards which penalize invalid solutions while avoiding discouraging performant local decision making due to failure of a global task. *Lines* 3 and 4 of Eq. 1 define the cases of successful job placement, where the agent receives not only an individual reward of 0.5 but also a team reward of $0.5 + \alpha/l$ in each step. This team reward incentivizes successful servicing of the job and also favors more advantageous solutions by incorporating an adaptive term based on the influence of the job on the network-wide performance, *i.e.*, end-to-end latency. Additionally, we introduce a penalty of Φ^{reconf} if accommodating the job entails topology reconfiguration of the ODCN. Consequently, as we will detail next, by optimizing the cumulative rewards, *i.e.*, sum of $\rho_{j,k}$ over a long term of service provisioning, we drive the agent to learn to allocate the resources more reasonably for maximizing the job acceptance ratio while minimizing latency and reconfiguration operations.

5) *Training mechanism*: Algorithm 1 summarizes the overall workflow of the DRL agent. In *Lines* 2, we initiate an empty experience buffer Π and set ε as 1 to encourage exploration. Then, *Lines* 3 initiate the policy network $f_{\theta_p}(\cdot)$ and the value network $f_{\theta_v}(\cdot)$. For each request $r_j \in R$, the DRL agent first updates the resource utilization state of the ODCN by releasing expired requests (*Line* 5). *Lines* 6-7 set a flag g_j to indicate whether the job is blocked and initialize an empty set A_j to carry the VM allocation action in each step. The for-loop spanning *Lines* 8-16 perform multi-step job scheduling with the assist of the DRL agent. Specifically, for each VM k, the agent obtains $s_{j,k}$ (*Line* 9), and calls its policy and value networks to calculate an allocation policy $f_{\theta_p}(s_{j,k})$ and a value estimation $f_{\theta_v}(s_{j,k})$ (*Line* 10). In *Lines* 11-12, with a probability of ε, the agent samples an action using the roulette method following the distribution of $f_{\theta_p}(s_{j,k})$. Otherwise, it selects the action associated with the highest probability which represents the optimal action based on the agent's current knowledge. Hence, different values of ε represent different ex-

Algorithm 1. Workflow of the DRL agent.

1: **Input**: $G(V, W, E)$, B, M, C_i, R, ε_0
2: Initialize $\Pi = \emptyset$, $\varepsilon = 1$;
3: Initialize policy and value networks $f_{\theta_p}(\cdot)$, $f_{\theta_v}(\cdot)$;
4: **for** each $r_j \in R$ **do**
5: Release resources occupied by expired jobs;
6: Set flag $g_j \leftarrow$ **false**;
7: Initialize the job scheduling solution $A_j \leftarrow \emptyset$;
8: **for** each step $k \in K_j$ **do**
9: Obtain $s_{j,k}$ based on r_j and current resource utilization;
10: Compute $f_{\theta_p}(s_{j,k})$ and $f_{\theta_v}(s_{j,k})$;
11: Compute cumulative sum of $f_{\theta_p}(s_{j,k})$ as Σ;
12: Select action $a_{j,k}$:

$$a_{j,k} = \begin{cases} \arg\min\{\Sigma(a) \geq \text{rand}()\}, & \text{with probability } \varepsilon, \\ \arg\max\{\pi_{\theta_p}(s_{j,k})\}, & \text{otherwise;} \end{cases}$$

13: **if** $a_{j,k}$ is infeasible **then**
14: Set $g_j \leftarrow$ **true** and **break**;
15: **else**
16: Append $a_{j,k}$ to A_j;
17: **if** g_j is **false then**
18: Check bandwidth availability according to A_j;
19: **if** bandwidth demand is unmet **then**
20: Re-check bandwidth availability with hypothetical ODCN reconfiguration;
21: Set $g_j \leftarrow$ **true** and **continue** if bandwidth demand is still unmet;
22: Commit bandwidth/VM allocation according to A_j and reconfigure the ODCN if necessary;
23: Compute $\rho_{j,k}$ using Eq. 1 and store $< s_{j,k}, a_{j,k}, \rho_{j,k}, f_{\theta_v}(s_{j,k}) >$ in Π for all $k \in K_j$;
24: **if** $|\Pi| \geq 2L - 1$ **then**
25: **for** each $< s_n, a_n, \rho_n, f_{\theta_v}(s_n) >$ in $\Pi[:L]$ **do**
26: Compute σ_n and $\aleph_n$ using Eqs. 2 and 3;
27: Compute $\mathcal{L}_{\theta_p}$ and $\mathcal{L}_{\theta_v}$ using Eqs. 4 and 5;
28: Apply gradients of $\mathcal{L}_{\theta_p}$ and $\mathcal{L}_{\theta_v}$ to update networks;
29: Remove the first L samples from Π;
30: Set $\varepsilon \leftarrow \max(\varepsilon - \varepsilon_0, \varepsilon_{\min})$;

ploration/exploitation preferences of the agent. If the action is infeasible, the job is marked as blocked in *Line* 14. Otherwise, the action is recorded in A_j (*Line* 16). If feasible VM allocation schemes have been found for all the VMs, the agent checks the bandwidth availability with ODCN reconfiguration possibilities in *Lines* 17-20. Only when sufficient bandwidth is also viable, we commit bandwidth and VM allocation according to the actions in A_j and reconfigure the ODCN if necessary (*Line* 22), otherwise, the job is finally blocked in *Line* 21. After all the VMs have been serviced or r_j is blocked, we calculate the reward for each action with Eq. 1 and store the corresponding sample in the experience buffer (*Line* 23). *Lines* 24-30 perform training when at least $2L - 1$ samples have been collected. Wherein, the for-loop from *Lines* 25-26 calculates the discounted cumulative reward and advantage for each of the first L samples in the buffer by,

$$\sigma_n = \sum_{t \in [0, L-1]} \gamma^t \rho_{n+t}, \tag{2}$$

$$\aleph_n = \sigma_n - f_{\theta_v}(s_n), \tag{3}$$

where $\gamma \in (0, 1]$ is the discount factor. In other words, the goal of the agent is maximizing its total reward within a provisioning window L, while the advantage represents the gain of taking this specific action a_n over the original estimation. Subsequently, we can training the policy and value networks by computing the policy loss $\mathcal{L}_{\theta_p}$ and the value loss $\mathcal{L}_{\theta_v}$ with Eqs. 4 and 5 and by applying their gradients with respect to the neural network weights (*Lines* 27-28).

$$\begin{aligned} \mathcal{L}_{\theta_p} = &- \mathbb{E}\left[\log f_{\theta_p}(s_n, a_n)\aleph_n\right] \\ &+ \beta \mathbb{E}\left[\sum_a f_{\theta_p}(s_n, a) \log f_{\theta_p}(s_n, a)\right], \end{aligned} \tag{4}$$

where $\beta \in [0, 1]$ is a hyperparameter that controls the trade-off between exploration and exploitation, *i.e.*, the first term in Eq. 4 favors actions with high advantage to maximize rewards, while the second term serves as an entropy regularizer to encourage exploration. In particular, the entropy regularization term promotes exploration by encouraging stochastic policies and preventing early convergence to suboptimal deterministic behaviors. This regularization helps the DRL agent maintain action selection diversity and improves policy robustness during training. The value loss function is defined as the mean squared error (MSE) between the estimated and observed cumulative rewards.

$$\mathcal{L}_{\theta_v} = \mathbb{E}\left[\left(\sigma_n - f_{\theta_v}(s_n)\right)^2\right] \tag{5}$$

Finally, *Lines* 29-30 remove obsolete samples and update ε before proceeding to the next job.

5. PERFORMANCE EVALUATION

We evaluated the proposed DRL-aided multi-stage job scheduling design through numerical simulations under a 16-ToR setup. Each ToR connects to the OCS plane with 30 ports (15 for input and another 15 for output) and hosts 32 servers. Server configurations were based on [30], with each server featuring 32 CPU cores, 256 GB memory, and 3,584 GB storage. We assumed a port capacity of 40 Gbps (i.e., B = 40). Each job requests for $[10, 21]$ VMs, randomly chosen from four specifications: [4 cores, 15 GB memory, 80 GB storage], [16 cores, 32 GB memory, 320 GB storage], [16 cores, 122 GB memory, 320 GB storage], and [16 cores, 122 GB memory, 3,200 GB storage]. These VMs remained unchanged during jobs' life cycles. The bandwidth demand per VM pair follows a uniform distribution within $[6, 8]$ Gbps. We tested different job loads and empirically selected task arrival rates of $[54, 60, 66, 72, 78]$ to represent operational regimes with moderate resource contention, avoiding both underutilized and heavily saturated conditions where scheduling decisions do not play a significant role in determining the network performance. For each training epoch, we generated 200,000 job requests. The job requests arrived according to a Poisson process, with the inter-arrival time following an exponential distribution centered at 1 time unit. We varied the mean value of job duration T_j to produce different traffic load scenarios. We conducted a wide range of ablation studies and selected the following hyperparameter configurations for the DRL agent. The policy and value networks of the DRL agent consist of five fully-connected hidden layers, each with 128 neurons. The hidden layers use *ELU* as the activation functions. α, β, γ, ε_0, ε_{min}, L and the learning rate were set as 100, 10^{-2}, 0.9, 10^{-5}, 0.5, 50 and 10^{-3}, respectively. We

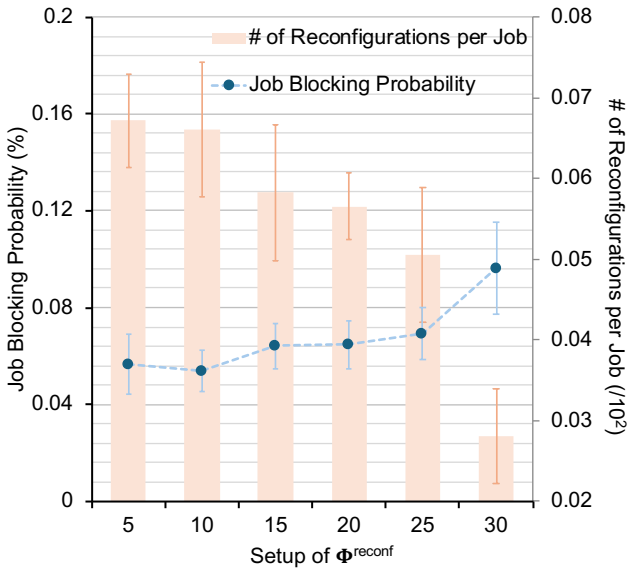

Fig. 3. Results of job blocking probability (left) and number of reconfigurations per job (right) as functions of Φ^{reconf}.

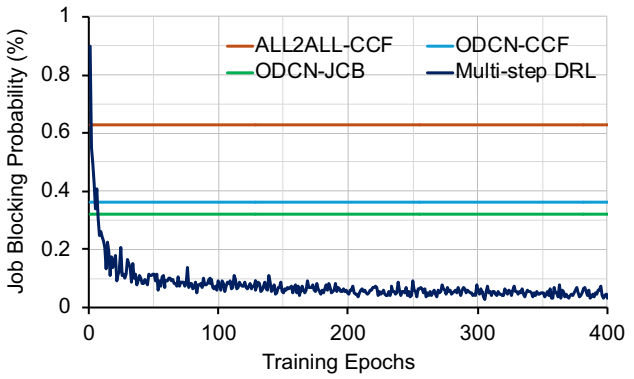

Fig. 4. Evolution of blocking probability from the DRL agent during training.

performed training with the Adam optimizer [31]. End-to-end latency and packet loss rate were evaluated with the queuing model presented in [32], considering the extreme cases where VMs transmit data at full rates. We assumed the packet size to be 296 bytes and buffer size to be 15 packets.

A. Evaluation on the Impact of Reconfiguration Cost

Despite higher frequencies of OCS reconfigurations may enhance the job acceptance ratio and end-to-end network performance, they add notable control and data plane overheads. Therefore, we first evaluated the impact of the setup of reconfiguration cost in Eq. 1 (*i.e.*, Φ^{reconf}) on the performance of the DRL agent and sought to figure out the best configuration of Φ^{reconf}. Fig. 3 presents the results of job blocking probability and number of reconfigurations per job as functions of Φ^{reconf}, when the average job duration is 66. We can see that as we increase the penalty on reconfiguration, the reconfiguration frequency drops consistently, indicating that the DRL agent learns scheduling policies that avoid excessive reconfigurations. The job blocking probability increases as fewer reconfigurations are invoked and its rising rate turns high when Φ^{reconf} goes beyond 25. This means that when $\Phi^{reconf} > 25$ (*e.g.*, 30), the reconfiguration cost gradually dominates the reward calculation so the agent is more inclined to block certain jobs than reconfiguring the optical interconnects. This phenomenon is confirmed by the sharp decrease in reconfiguration frequency at the same position. Following this observation, we set Φ^{reconf} to be 25 in the rest of the simulations to allow the DRL agent to better balance between the blocking probability and reconfiguration cost.

Table 2. Resource utilization ratio (%) from a 16-ToR ODCN configuration under different job loads.

Load	Policy	CPU	RAM	Disk
54	ALL2ALL-CCF	64.4	45.0	43.7
	ODCN-CCF	65.3	45.8	44.2
	ODCN-JCB	65.0	45.3	43.5
	Multi-step DRL	65.9	46.1	44.8
60	ALL2ALL-CCF	73.6	51.4	49.5
	ODCN-CCF	72.7	50.8	49.1
	ODCN-JCB	70.6	49.3	47.5
	Multi-step DRL	74.3	51.9	50.1
66	ALL2ALL-CCF	79.1	55.4	53.3
	ODCN-CCF	78.5	54.8	52.7
	ODCN-JCB	79.3	55.6	53.3
	Multi-step DRL	81.6	57.1	55.0
72	ALL2ALL-CCF	85.4	59.8	57.6
	ODCN-CCF	85.1	59.5	57.9
	ODCN-JCB	85.6	59.9	58.3
	Multi-step DRL	88.9	62.2	59.9
78	ALL2ALL-CCF	87.9	61.2	59.0
	ODCN-CCF	89.6	62.5	60.6
	ODCN-JCB	88.9	62.3	59.5
	Multi-step DRL	94.2	66.2	63.4

B. Comparison with Heuristic Baselines

We compared our proposal with three heuristic approaches from prior works [24]: 1) ALL2ALL-CCF, which applies the computing capacity first (CCF) VM allocation policy in a fixed all-to-all interconnect architecture (same port number and capacity configuration as in the flexible-bandwidth ODCN); 2) ODCN-CCF, which applies the CCF policy in the ODCN; and 3) ODCN-JCB, which applies a resource and bandwidth-aware joint optimization policy (JCB) in the ODCN. Hence, the baselines in this paper are ALL2ALL-CCF, ODCN-CCF and ODCN-JCB. Fig. 4 shows the evolution of blocking probability from the DRL agent during training when the average service duration is 66. The performance from the baselines remains flat as they employ fixed policies. It can be observed that our DRL design quickly converges and beats the baselines after training of 10 epochs. The DRL agent asymptotically reduces the blocking probability by 10.33×, 5.9× and 5.25× over ALL2ALL-CCF, ODCN-CCF, and ODCN-JCB, respectively. Further, we show the comparisons between our proposal and the baselines under different job loads (by varying the job duration) in Fig. 5. From Fig. 5(a), we

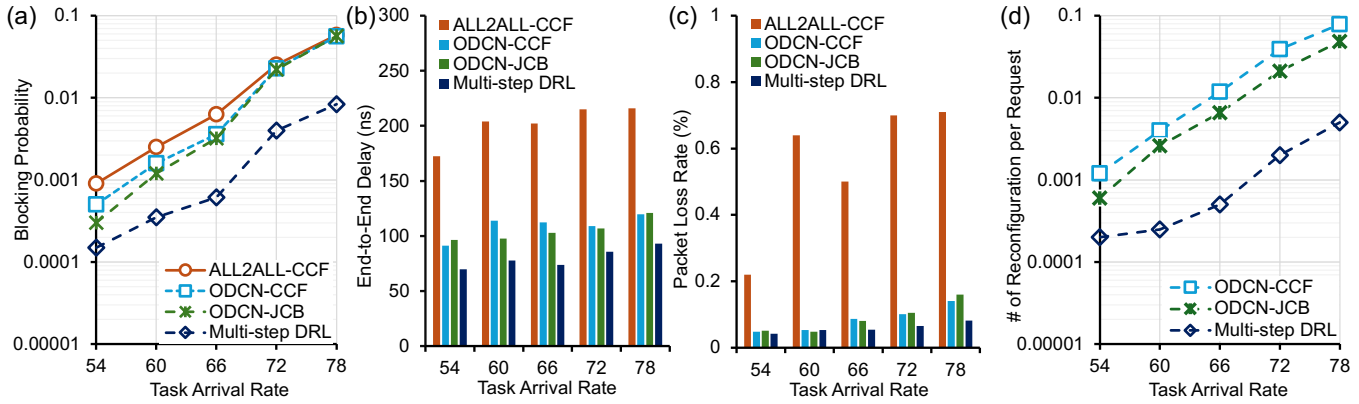

Fig. 5. Results from a 16-ToR ODCN configuration: (a) job blocking probability, (b) end-to-end latency, (c) packet loss rate, and (d) number of reconfigurations per job under different job loads.

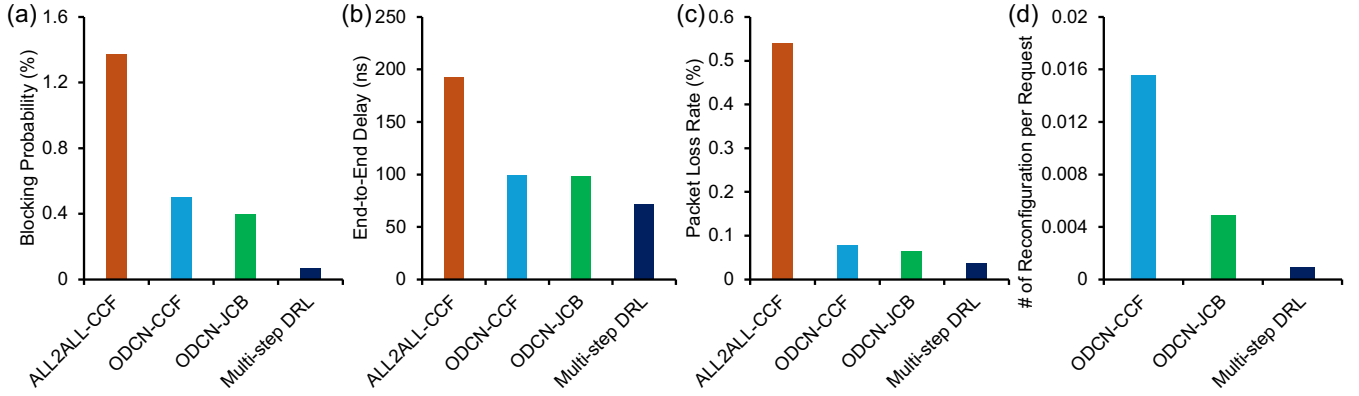

Fig. 6. Results from a 32-ToR ODCN configuration: (a) job blocking probability, (b) end-to-end latency, (c) packet loss rate, and (d) number of reconfigurations per job.

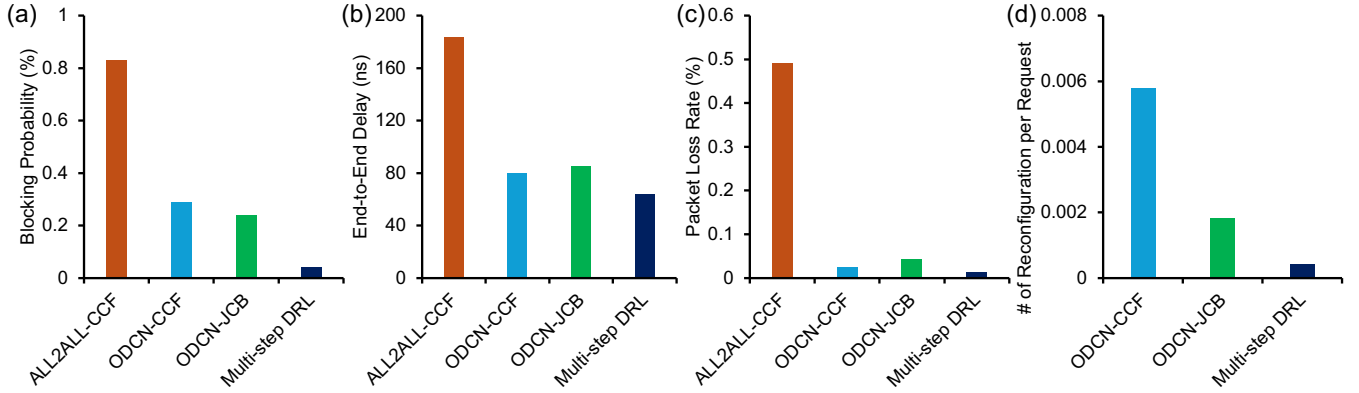

Fig. 7. Results from a 64-ToR ODCN configuration: (a) job blocking probability, (b) end-to-end latency, (c) packet loss rate, and (d) number of reconfigurations per job.

can see that our proposal consistently outperforms the baselines due to its ability to sense the complex multi-dimensional resource state and learn adaptive scheduling policies. On average, our proposal reduces the blocking probability by 7.35×, 5.24×, and 4.59× compared with the three baselines, respectively. As expected, ALL2ALL-CCF performs the worst among the four algorithms due to lacking the ability to adapt the interconnect topology. Because our reward definition also stimulates better end-to-end performance, Figs. 5(b)-(c) show that the multi-step DRL design also achieves the lowest end-to-end latency and packet loss rate, *i.e.*, achieving a latency reduction of 1.32× and a packet loss rate reduction of 1.44× over the best baseline. Again, ALL2ALL-CCF underperforms the rest by a large margin because it cannot steer bandwidth to obviate hot-spot links. The superiority of our proposal is further verified by its advantage in the number of reconfigurations as shown in Fig. 5(d). By

Table 3. Resource utilization ratio (%) under a 32-ToR ODCN configuration.

Policy	CPU	RAM	Disk
ALL2ALL-CCF	87.1	60.8	58.1
ODCN-CCF	87.0	60.7	58.3
ODCN-JCB	87.8	61.4	59.1
Multi-step DRL	90.7	63.5	61.0

Table 4. Resource utilization ratio (%) under a 64-ToR ODCN configuration.

Policy	CPU	RAM	Disk
ALL2ALL-CCF	88.1	61.7	59.6
ODCN-CCF	89.2	61.9	59.3
ODCN-JCB	88.3	61.8	58.9
Multi-step DRL	96.3	67.2	64.6

regularizing the reward function with a reconfiguration cost, our proposal requires 16.22× and 9.38× fewer reconfiguration operations than ODCN-CCF and ODCN-JCB, respectively. Table 2 presents the utilization of individual IT resources (*i.e.*, CPU, memory, and disk) across all scheduling policies under varying job load levels. As the job load increases, all policies exhibit a consistent upward trend in resource utilization, reflecting intensified system activity. The proposed multi-step DRL approach consistently achieves the highest utilization of all the three resource types. Compared with the worst-performing baseline in each category (ALL2ALL-CCF), our approach approximately improves CPU, memory and disk utilization by 7%, 8% and 7%, respectively. These results highlight the DRL agent's superior capability in orchestrating multi-resource allocations under high-load conditions.

C. Scalability Studies

Last, we evaluated the scalability of the proposed approach through simulations in larger ODCN configurations, *i.e.*, a 32-ToR and a 64-ToR ODCN, with each ToR hosting 64 and 128 servers, respectively. To maintain a reasonable level of resource utilization, we set the job duration as 300 and 1,320 respectively in the two configurations. The setups for the rest of the parameters remained the same as those used in the previous simulations. Fig. 6 shows the comparison between our proposal and the baselines in the 32-ToR ODCN. The results are in line with those obtained from the 16-ToR ODCN. In particular, our algorithm reduces job blocking probability, end-to-end latency and packet loss rate by at least 7.46×, 1.39× and 2.08× over the baselines, respectively. When compared with ODCN-CCF and ODCN-JCB, our design requires 17.33× and 5.44× fewer reconfiguration operations as depicted in Fig. 6(d). Similar observations can be drawn from the results obtained from the 64-ToR ODCN (see Fig. 7). Indeed, the results show our multi-step DRL approach maintains its advantages in larger-scale ODCNs, by reducing the blocking probability and reconfiguration frequency by up to 20.75× (over ALL2ALL-CCF) and 14.5× (over ODCN-CCF), respectively. Tables 3 and 4 present the CPU, memory and disk utilization under 32-ToR and 64-ToR ODCN configurations, respectively. In both cases, the multi-step DRL design achieves the highest utilization across all resource types. Under the 32-ToR configuration, it improves the CPU, memory and disk utilization by 3% over the best baseline. Under the 64-ToR configuration, it further attains 96.3% CPU, 67.2% memory and 64.6% disk utilization. These results demonstrate the scalability of the DRL approach in effectively managing more complex network configurations.

6. CONCLUSION

In conclusion, this paper presents a novel multi-step job scheduling algorithm for ODCNs leveraging DRL to address the complexities of job scheduling and topology reconfiguration. By decomposing job requests into VM placements and bandwidth demands, our approach optimizes both VM placement and resource allocation sequentially. The DRL agent learns to make informed decisions, balancing global resource utilization, local placement, and reconfiguration overheads. The proposed algorithm outperforms existing heuristic methods, achieving significant reductions in job blocking probability and reconfiguration frequency. These results highlight the effectiveness of our approach in enhancing the performance of ODCNs for demanding distributed ML workloads, paving the way for more efficient, scalable, and adaptive data center architectures. Future research directions include: (1) exploring joint optimization of job scheduling for intra- and inter-POD systems in reconfigurable ODCNs to further enhance resource utilization and performance; (2) enhancing DRL by integrating ensemble learning or graph neural networks to improve decision-making efficiency and generalization; (3) extending the framework to consider energy efficiency and sustainability by optimizing task placement and network reconfiguration to reduce power consumption.

REFERENCES

1. F. Giroire, N. Huin, A. Tomassilli, and S. Pérennes, "When network matters: Data center scheduling with network tasks," in *Proc. IEEE Conf. Comput. Commun.*, (2019), pp. 2278–2286.
2. H. Zhang, L. Stafman, A. Or, and M. J. Freedman, "Slaq: quality-driven scheduling for distributed machine learning," in *SoCC*, (2017), p. 390–404.
3. Y. Bao, Y. Peng, C. Wu, and Z. Li, "Online job scheduling in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, (2018), pp. 495–503.
4. Y. Bao, Y. Peng, and C. Wu, "Deep learning-based job placement in distributed machine learning clusters," in *Proc. IEEE Conf. Comput. Commun.*, (2019), pp. 505–513.
5. M. Yu, J. Liu, C. Wu, B. Ji, and E. S. Bentley, "Toward efficient online scheduling for distributed machine learning systems," IEEE Trans. Netw. Sci. Eng. **9**, 1951–1969 (2022).
6. M. Yu, Y. Tian, B. Ji, C. Wu, H. Rajan, and J. Liu, "Gadget: Online resource optimization for scheduling ring-all-reduce learning jobs," in *Proc. IEEE Conf. Comput. Commun.*, (2022), p. 1569–1578.
7. R. Zhou, J. Pang, Q. Zhang, C. Wu, L. Jiao, Y. Zhong, and Z. Li, "Online scheduling algorithm for heterogeneous distributed machine learning jobs," IEEE Trans. Cloud Comput. **11**, 1514–1529 (2023).
8. X. Lu, C. Liu, S. Zhu, Y. Mao, P. Lio, and P. Hui, "Rlpto: A reinforcement learning-based performance-time optimized task and resource scheduling mechanism for distributed machine learning," IEEE Trans. Parallel Distrib. Syst. **34**, 3266–3279 (2023).
9. H. Wang, Z. Liu, and H. Shen, "Machine learning feature based job

scheduling for distributed machine learning clusters," IEEE Trans. Netw. **31**, 58–73 (2023).
10. Y. Li, X. Zhang, T. Zeng, J. Duan, C. Wu, D. Wu, and X. Chen, "Task placement and resource allocation for edge machine learning: A gnn-based multi-agent reinforcement learning paradigm," IEEE Trans. Parallel Distrib. Syst. **34**, 3073–3089 (2023).
11. Z. Luo, J. Liu, M. Lee, and N. B. Shroff, "Prediction-assisted online distributed deep learning workload scheduling in gpu clusters," (2025).
12. Z. Zhang, Q. Qi, R. Shang, L. Chen, and F. Xu, "Prophet: Speeding up distributed dnn training with predictable communication scheduling," in *Int. Conf. Parallel Process.*, (2021), pp. 1–11.
13. R. Gu, Y. Chen, S. Liu, H. Dai, G. Chen, K. Zhang, Y. Che, and Y. Huang, "Liquid: Intelligent resource estimation and network-efficient scheduling for deep learning jobs on distributed gpu clusters," IEEE Trans. Parallel Distrib. Syst. **33**, 2808–2820 (2022).
14. B. Guo, Z. Zhang, Y. Yan, and H. Li, "Optimal job scheduling and bandwidth augmentation in hybrid data center networks," in *Proc. IEEE Global Commun. Conf.*, (2022), pp. 5686–5691.
15. Y. Bi, Y. Xi, and C. Jing, "Aisaw: An adaptive interference-aware scheduling algorithm for acceleration of deep learning workloads training on distributed heterogeneous systems," Futur. Gener. Comput. Syst. **166**, 107642 (2025).
16. G. Liu, R. Proietti, M. Fariborz, P. Fotouhi, X. Xiao, and S. Ben Yoo, "Architecture and performance studies of 3d-hyper-flex-lion for reconfigurable all-to-all hpc networks," in *Proc. IEEE Conf. Supercomput.*, (2020), pp. 1–16.
17. Q. Lv, Y. Zhang, S. Zhang, R. Li, K. Meng, B. Zhang, F. Huang, X. Chen, and Z. Zhu, "On the TPE design to efficiently accelerate hitless reconfiguration of OCS-based DCNs," IEEE J. Sel. Areas Commun. (2025).
18. M. Khani, M. Ghobadi, M. Alizadeh, Z. Zhu, M. Glick, K. Bergman, A. Vahdat, B. Klenk, and E. Ebrahimi, "Sip-ml: high-bandwidth optical network interconnects for machine learning training," in *ACM SIGCOMM Comput. Commun.*, (2021), p. 657–675.
19. C.-Y. Liu, X. Chen, Z. Li, R. Proietti, and S. J. B. Yoo, "Sl-hyper-flex: a cognitive and flexible-bandwidth optical datacom network by self-supervised learning [invited]," J. Opt. Commun. Netw. **14**, A113–A121 (2022).
20. W. Zheng, X. Chen, and Z. Li, "Fast and nondisruptive reconfiguration design for optical datacom networks," in *PSC,* (2023), pp. 1–3.
21. A. Ottino, J. Benjamin, and G. Zervas, "Ramp: A flat nanosecond optical network and mpi operations for distributed deep learning systems," Opt. Switch. Netw. **51** (2024).
22. X. Xie, B. Tang, X. Chen, and Z. Zhu, "P4INC-AOI: All-optical interconnect empowered by in-network computing for DML workloads," IEEE Trans. Netw. pp. 1–16 (2025).
23. Y. Guo, X. Xue, B. Guo, D. Dang, S. Shen, Y. Zhao, B. Wei, C. Yang, G. Wang, and S. Huang, "Awgr-based all-optical switching network for distributed machine learning," Opt. Express **33**, 829–841 (2025).
24. X. Chen, C.-Y. Liu, R. Proietti, S. Chen, Z. Li, and S. B. Yoo, "When task scheduling meets flexible-bandwidth optical interconnects: A cross-layer resource orchestration design," in *Proc. Conf. Opt. Fiber Commun.*, (2022), pp. 1–3.
25. H. Mahmoud, M. Thabet, M. H. Khafagy, and F. A. Omara, "An efficient load balancing technique for task scheduling in heterogeneous cloud environment," J. Clust. Comput. **24**, 3405–3419 (2021).
26. F. Yao, C. Pu, and Z. Zhang, "Task duplication-based scheduling algorithm for budget-constrained workflows in cloud computing," IEEE Access **9**, 37262 – 37272 (2021).
27. P. K. Bal, S. K. Mohapatra, T. K. Das, K. Srinivasan, and Y.-C. Hu, "A joint resource allocation, security with efficient task scheduling in cloud computing using hybrid machine learning techniques," IEEE Sens. J. **22**, 1242 (2022).
28. M. Sudheer, K. Reddy, M. V. Ratnamani, S. Mohanty, B. Jabr, Y. Ali, S. Ali, and B. Abdullaeva, "Efficient deep reinforcement learning based task scheduler in multi cloud environment," Sci. Rep. **14**, 21850 (2024).
29. 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 *Proc. 33rd Int. Conf. on Machine Learning,* vol. 48 (2016), pp. 1928–1937.
30. X. Dai, J. M. Wang, and B. Bensaou, "Energy-efficient virtual machines scheduling in multi-tenant data centers," IEEE Trans. Cloud Comput. **4**, 210–221 (2016).
31. D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," (2017).
32. L. Mason, T. Drwiega, and J. Yan, "Managing traffic performance in converged networks," in *ITC,* (2007).