Future Generation Computer Systems 167 (2025) 107760

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

GAS-MARL: Green-Aware job Scheduling algorithm for HPC clusters based on Multi-Action Deep Reinforcement Learning

Rui Chen [a], Weiwei Lin [a,b,*], Huikang Huang [a], Xiaoying Ye [c], Zhiping Peng [d,e]

[a] Department of Computer Science and Engineering, South China University of Technology, GuangZhou, 510000, China
[b] Pengcheng Laboratory, Shenzhen, 518066, China
[c] Neusoft Institute of Guangdong, Foshan, 528225, China
[d] Guangdong University of Petrochemical Technology, Maoming, 525000, China
[e] Jiangmen Polytechnic, Jiangmen, 529000, China

ARTICLE INFO

Dataset link: https://github.com/sioncr/green-rl-sched

Keywords:
Job scheduling
High-performance computing
Deep Reinforcement Learning
Renewable energy
Green computing

ABSTRACT

In recent years, the computational power of High-Performance Computing (HPC) clusters has surged. However, amidst global calls for energy conservation and emission reduction, their rapid power consumption poses a developmental bottleneck. Adopting renewable energy sources for power supply is a crucial measure to reduce carbon emissions from HPC clusters. However, due to the variability and intermittency of renewable energy, formulating effective job scheduling plans to fully utilize these sources has become urgent. To tackle this, we propose a Green-Aware job Scheduling algorithm for HPC clusters based on Multi-Action Deep Reinforcement Learning (GAS-MARL), which optimizes both renewable energy utilization and average bounded slowdown. In this algorithm, the agent outputs two actions during one decision-making period: job selection action and delay decision action. The introduction of delay decision actions enhances the flexibility of the scheduling algorithm, enabling each job to be executed during appropriate time slots. Furthermore, we have designed a new backfilling policy called Green-Backfilling to better cooperate with GAS-MARL for job scheduling. Experimental evaluations demonstrate that, compared to other algorithms, the combination of GAS-MARL and Green-Backfilling exhibits significant advantages in enhancing renewable energy utilization and decreasing average bounded slowdown.

1. Introduction

The rapid development of computer hardware over the past two decades has significantly increased the computational power of HPC systems, with the peak performance of advanced supercomputers reaching 1206 PFlop/s [1]. However, the swift enhancement in performance has led to a corresponding rise in power consumption. From 2012 to 2024, the world's top-ranked supercomputer system's power consumption escalated from 8209 kW to 22786 kW, according to the TOP500 list. This massive power demand implies substantial carbon emissions, facing enormous social pressure in today's global push for carbon neutrality.

To address challenges, introducing renewable energy into HPC cluster power systems is an important measure. Many large IT companies have established renewable energy systems for their data centers to pursue carbon neutrality [2]. However, the variability and intermittency of renewable energy pose significant challenges for its use in HPC clusters. Fortunately, jobs submitted by users in HPC clusters are generally batch jobs, which can tolerate some delay [3]. Thus, the job scheduling system can schedule jobs to execute during periods with relatively sufficient renewable energy generation to reduce emissions. On the other hand, users typically expect prompt job completion. Pursuing maximum renewable energy utilization alone may cause unacceptable delays for certain jobs. This necessitates a balance between renewable energy utilization and job delay in the scheduling system. Thus, arranging reasonable job scheduling to enhance renewable energy utilization while minimizing average bounded slowdown is a practically significant challenge.

The job scheduling system is a key component in HPC clusters, monitoring resource status and determining job execution order based on algorithms to optimize desired metrics. Researchers have proposed various scheduling algorithms, including heuristic [4–7], constraint programming [8,9], metaheuristic [3,10,11], and deep reinforcement learning (DRL) [12–17]. Heuristic algorithms are fast but have poor

* Corresponding author at: Department of Computer Science and Engineering, South China University of Technology, GuangZhou, 510000, China.
E-mail addresses: cssion@mail.scut.edu.cn (R. Chen), linww@scut.edu.cn (W. Lin), huikanghuang0321@gmail.com (H. Huang), yexiaoying@nuit.edu.cn (X. Ye), pengzp@gdupt.edu.cn (Z. Peng).

https://doi.org/10.1016/j.future.2025.107760
Received 20 August 2024; Received in revised form 4 December 2024; Accepted 7 February 2025
Available online 17 February 2025

performance and limited adaptability. Constraint programming algorithms have high complexity and scalability issues. Metaheuristic algorithms have relatively high complexity and require extensive solution times, making them less practical for real-world production environments.

Deep Reinforcement Learning, a decision optimization algorithm, has emerged recently, offering strong performance, rapid decision-making, and adaptability. While several DRL-based job scheduling algorithms for HPC clusters have been proposed, to our knowledge, none of them have considered optimizing renewable energy utilization or employing proactive delay decisions to enable jobs to execute during periods of sufficient renewable energy. To address this issue, we propose a Green-Aware job Scheduling algorithm for HPC clusters based on Multi-Action Deep Reinforcement Learning (GAS-MARL). This algorithm optimizes both renewable energy utilization and average bounded slowdown. During each decision period, it outputs two actions: one determines the job execution order, and the other involves proactive delay decisions, including whether and how long to delay jobs. Additionally, we have designed a new backfilling policy, Green-Backfilling, which can better collaborate with GAS-MARL and effectively reduce average bounded slowdown without sacrificing renewable energy utilization.

Specifically, the main contributions of this paper are summarized as follows:

(1) We propose GAS-MARL, a job scheduling algorithm based on Proximal Policy Optimization (PPO). It features a multi-action space and outputs two actions per decision period: job selection action and delay decision action. The introduction of delay decision action offers notable potential for optimizing renewable energy utilization.
(2) To address the lack of consideration for renewable energy usage in conventional backfilling policies, we have designed a new policy for GAS-MARL called Green-Backfilling. This policy integrates well with GAS-MARL, effectively reducing average bounded slowdown without lowering renewable energy utilization.
(3) We tested the proposed algorithm on various job traces and compared it with existing methods. The results show that the combination of GAS-MARL and Green-Backfilling offers significant advantages in both average bounded slowdown and renewable energy utilization.

The rest of the paper is organized as follows. Section 2 reviews related work, Section 3 defines the green-aware job scheduling problem for HPC clusters with hybrid power supply, Section 4 details the design of GAS-MARL and Green-Backfilling, Section 5 presents the algorithm's performance in experiments, and Section 6 concludes with a summary and future directions.

2. Related work

The research progress on applying deep reinforcement learning to HPC cluster job scheduling is as follows. Zhang et al. [12] proposed an HPC job scheduling algorithm, RLScheduler, based on PPO, which follows an actor–critic model structure. Its key feature is that the policy network uses a kernel-based neural network, designed as a three-layer fully connected network, applied individually to each waiting job to score them based on features, making the model insensitive to job order. To ensure a fixed input length, RLScheduler limits the observable number of jobs to a fixed window size. Additionally, it uses a trajectory filtering technique to improve training stability and efficiency. Fomperosa et al. [13] improved RLScheduler to enable its use in heterogeneous cluster environments. Li et al. [14] proposed an energy-aware job scheduling algorithm using DRL, which effectively reduces system energy consumption while maintaining service quality, employing a partition-based approach to reduce the action space. GARLSched [15] is a job scheduling algorithm based on generative adversarial reinforcement learning. Its training process is guided by optimal strategies from an expert library. Fan et al. [16] presented DRAS, which uses DRL for job selection during backfilling. de Freitas Cunha et al. [17] mapped the HPC batch job scheduling problem to a Semi-Markov Decision Process (SMDP), as SMDPs allow for temporal abstraction of actions. Based on this, they proposed a scheduling algorithm using PPO, which supports a large action space to minimize the average bounded slowdown. The algorithm uses a maskable PPO implementation, which applies a state-dependent differentiable function during the calculation of action probabilities to prevent the agent from sampling invalid actions, thereby improving training efficiency.

Some researchers have proposed HPC job scheduling algorithms that consider the use of renewable energy. Aikema et al. [18] developed an adaptive scheduling policy that halts job execution during periods of low renewable energy availability to reduce the carbon footprint. Kassab et al. [3] proposed a job scheduling algorithm for HPC platforms using only renewable energy, employing a Genetic Algorithm (GA) to optimize job scheduling and minimize completion time. Kessaci et al. [10] proposed an HPC job scheduling algorithm using a multi-objective GA to minimize energy consumption, CO2 emissions, and maximize profit, while respecting deadlines. The work [19] proposed and tested several heuristic algorithms for task scheduling optimization on renewable-powered HPC platforms. The works [3,19] are limited to HPC clusters powered only by renewable energy. In practice, when renewable energy is insufficient, brown energy is often used to supplement power to avoid excessive job waiting time. This requires the scheduler to decide whether to execute jobs when only brown energy is available, increasing the complexity of the scheduling problem. Moreover, existing job scheduling algorithms for HPC clusters that consider renewable energy are limited to heuristic and metaheuristic approaches, which respectively suffer from poor performance and long solution times.

Backfilling is a common method to reduce HPC resource fragmentation, and related research on backfilling policies is as follows. The work [20] uses machine learning to predict job running time and improve backfilling performance. Lelong et al. [21] studied the effects of different sorting strategies on reordering the main and backfill queues in EASY-Backfilling. The work [22] introduced the Fattened Backfilling algorithm, which allows short jobs to advance without delaying the first job beyond the average wait time, thereby increasing backfilling opportunities. Kolker-Hicks et al. [23] proposed RLBackfilling, a reinforcement learning-based algorithm that learns efficient backfilling policy through training. The shortcoming of these studies is that they do not consider job power consumption and renewable energy generation, which may impact the optimization of renewable energy utilization.

Compared to existing HPC job scheduling algorithms that consider renewable energy, GAS-MARL achieves higher performance through training and offers a significant advantage in decision-making speed. Compared to existing DRL-based HPC job scheduling algorithms, the key feature of GAS-MARL is the introduction of delay decision action. This allows the algorithm to actively delay job execution, enabling jobs to run during periods of abundant renewable energy, significantly enhancing its potential for optimizing renewable energy utilization. Additionally, compared to existing backfilling policies, Green-Backfilling takes renewable energy into account and improves upon GAS-MARL's features, effectively reducing average bounded slowdown without compromising renewable energy utilization.

3. Problem definition

In HPC clusters, users submit jobs intermittently, specifying required resources. After submission, jobs enter a queue and await resource allocation by the scheduling system. As HPC applications are usually batch jobs with some tolerance for delay, there is room to optimize scheduling for both system administrators and users. This section will model the green-aware job scheduling problem for HPC clusters with hybrid power supply and outline the optimization objectives.

3.1. Modeling of HPC clusters with mixed power supply

Homogeneous Cluster Modeling: For simplicity in our research, we set up the HPC cluster in the experiment as a homogeneous structure, meaning all computing resources in the cluster are treated as a single resource type. This system model has been adopted in many studies on HPC cluster job scheduling [12,15,16]. In this work, the smallest scheduling unit in the HPC cluster is set as the processor, which can be either idle or occupied at any given time. All machines in the cluster are assumed to have identical specifications, each containing a certain number of processors. To model machine power consumption, we follow methods from previous works [3,19,24], representing it as the sum of static and dynamic power consumption. Static power consumption refers to the power consumption when the machine is idle, while dynamic power consumption refers to the additional power consumption caused by running jobs. Thus, the total power consumption of the HPC cluster at timestamp s can be expressed as follows:

$$P_C(s) = P_{idle}(s) + \sum_{j \in Rj_s} pow_j(s) \tag{1}$$

where $P_C(s)$ is the total power consumption of the cluster at timestamp s, $P_{idle}(s)$ is the aggregate idle power of all machines, considered constant as we ignore shutdown and startup. $pow_j(s)$ is the power consumption of job j at timestamp s, and Rj_s is the set of running jobs at timestamp s.

Job Attributes: In HPC clusters, jobs are typically batch jobs, characterized by attributes such as submit time, number of requested processors, requested runtime, execution time, and power consumption. Power consumption is crucial for green-aware scheduling and varies by job type [19]. While job power may actually vary during execution, modeling it as dynamically changing would make the job scheduling problem overly complex. The work [25] shows that most HPC jobs exhibit relatively stable power consumption during runtime, with average power effectively representing job characteristics. Thus, a feasible approach is to predict the average power consumption of jobs and use it as one of the job attributes for scheduling decisions [9]. Many studies have proposed algorithms for predicting the power consumption of HPC jobs [25,26], but this is not the focus of our work. Hence, like some previous works [3,19], we directly use randomly generated job power consumption data for scheduling.

Modeling Hybrid Power Supply System: We configure the HPC cluster's power supply as a hybrid system using both brown and renewable energy sources. We consider solar and wind energy as the renewable sources, which are commonly used. Their generation varies with weather conditions. For modeling, time is discretized into equal slots, assuming solar power remains constant within each slot and only changes between slots. In this work, we set the time slot size to one hour and, as in some existing studies [3,19], assume that the electricity supplier can provide predicted values for renewable energy generation for the next day (i.e., 24 slots). Brown energy availability is constant but emits CO2, so our system prioritizes renewable energy and uses brown energy only when renewable energy generation cannot meet demand.

3.2. Optimization problem definition

The HPC cluster job scheduling problem studied in this work is a bi-objective optimization problem, aiming to optimize both renewable energy utilization and average bounded slowdown. The renewable energy utilization ($ReUtil$) refers to the ratio of renewable energy consumed by the HPC cluster to its total energy consumption across all timestamps, calculated as follows:

$$P_{GT}(s) = P_{solar}(s) + P_{wind}(s) \tag{2}$$

$$P_G(s) = \begin{cases} P_{GT}(s), & if\ P_{GT}(s) \leqslant P_C(s); \\ P_C(s), & otherwise \end{cases} \tag{3}$$

$$ReUtil = \frac{\int_{ST}^{ET} P_G(s)\mathrm{d}s}{\int_{ST}^{ET} P_C(s)\mathrm{d}s} \tag{4}$$

where $P_{GT}(s)$ is the total renewable energy generation power at timestamp s, $P_{solar}(s)$ and $P_{wind}(s)$ are the solar and wind power generation power at timestamp s, $P_G(s)$ is the renewable energy generation power consumed by the cluster at timestamp s, $P_C(s)$ is the cluster's total power consumption calculated as in Eq. (1), ST is the timestamp when the HPC cluster starts executing jobs, and ET is the completion timestamp of the last job.

Average bounded slowdown ($AvgBSLD$) is a widely used metric for HPC job scheduling [23], which measures the ratio of job turnaround time to execution time. The specific calculation formula is as follows:

$$AvgBSLD = \frac{1}{N} \times \sum_{i=1}^{N} max(\frac{w_i + e_i}{max(\tau, e_i)}, 1) \tag{5}$$

where N is the number of completed jobs, w_i is the waiting time of job i, e_i is the execution time of job i, and τ is a threshold used to prevent jobs with very short execution times from having an excessive impact on the average value. In this work, τ is set to 10 s.

When performing job scheduling, the following constraints need to be met: (1) A job can only start when the HPC cluster has the requested number of processors available, and it must occupy the requested number of processors throughout its execution; (2) The total number of processors used by all running jobs cannot exceed the cluster's total processors. Thus, the job scheduling problem in this work is defined as follows:

$$\begin{cases} maxmize f_1 = ReUtil \\ minimize f_2 = AvgBSLD \end{cases} \tag{6}$$

$$\text{s.t.},\ h_j^s = q_j, \forall j \in J_{all} \text{ and } \forall s \in T_j \tag{7}$$

$$\sum_{j \in Rj_s} h_j^s \leqslant procNum, \forall s \in T_{all} \tag{8}$$

where h_j^s is the number of processors job j occupies at timestamp s, q_j is the requested processors, T_{all} is the set of all executed jobs, T_s is the set of timestamps job j executes through, $procNum$ is the total processors in the HPC cluster, and T_{all} is the set of all timestamps that the HPC cluster has gone through.

4. Proposed algorithm

4.1. Algorithm overview

As shown in Fig. 1, our proposed algorithm consists of two parts: GAS-MARL and Green-Backfilling. GAS-MARL, based on the PPO, not only controls the execution order of jobs but also determining whether a job should be delayed and the duration of the delay. This multi-action decision-making enhances the scheduling flexibility of the algorithm, enabling jobs to be executed during periods of relatively abundant renewable energy generation. The workflow of the proposed algorithm is as follows: First, the HPC cluster management system gathers cluster information and inputs it into the GAS-MARL agent. The agent does not decide on the two actions in parallel. Instead, it first outputs the job selection action from the job selection network. Then, it adds the information of the selected job to the input vector of the delay decision network and finally outputs the delay decision action. Once both decisions are made, the cluster executes the scheduling accordingly. The system first considers the delay decision. If a delay is needed, it will not start the job until the delay ends. After the delay, it will check if resources are available for the job. If they are, the job starts immediately; otherwise, resources will be reserved to ensure prompt execution. During this process, Green-Backfilling will select jobs from the waiting queue for execution if delays or resource reservation occur.

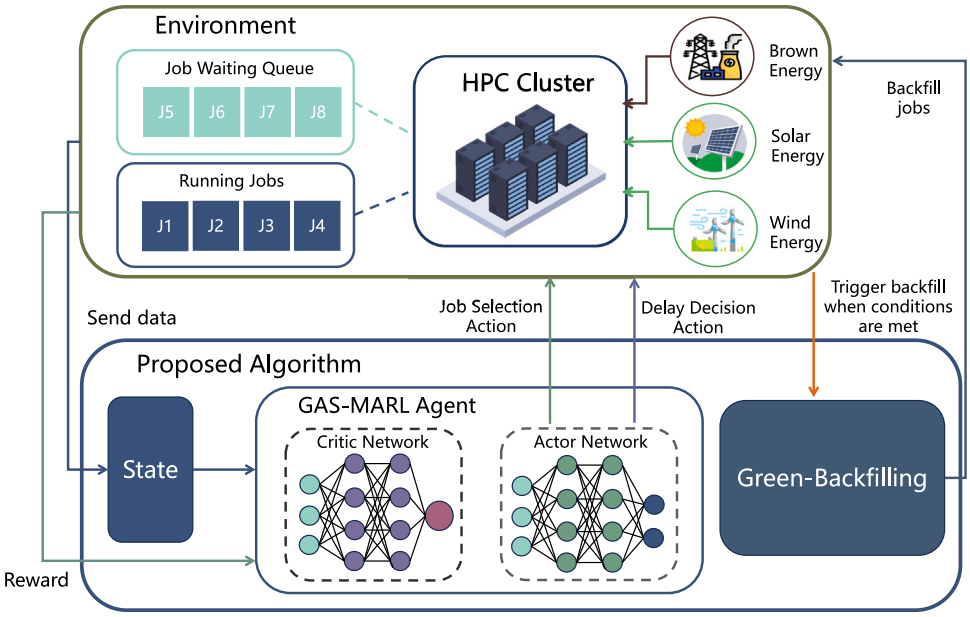

Fig. 1. Architecture of the proposed algorithm.

4.2. GAS-MARL

This section will first model the green-aware HPC cluster job scheduling problem as a Markov Decision Process (MDP) and provide details on the GAS-MARL network structure and the policy optimization algorithm based on PPO that we adopt.

4.2.1. MDP modeling

To address the green-aware HPC cluster job scheduling problem using reinforcement learning, we need to model it as an MDP, which is a framework for sequential decision-making. It consists of $\langle S, A, P, r, \gamma \rangle$: state space, action space, state transition matrix, reward, and discount factor. Here, the discount factor $\gamma \in [0, 1]$ is a key parameter that balances the importance of immediate and future rewards, A higher γ value encouraging the model to consider long-term rewards in its scheduling decisions. For the MDP model designed in this work, we found that setting γ to 1 enhances the training performance of the scheduling model.

Below, we will provide specific definitions for the state space, action space, and reward function.

(1) State space

The state space designed in this work consists of three types of information: waiting queue information, running jobs information, and renewable energy information. These will be described in detail next.

Waiting Queue Information: To fix the neural network's input length, we set the number of jobs in the waiting queue information to a fixed window size Q (set at 256). If there are more than Q jobs, select the earliest Q jobs based on submit time as the jobs within the window. A job can be represented as a vector of $[1, 8]$, so the waiting queue information is a $[Q, 8]$ matrix, where each row represents a job and contains the following attributes (taking job j as an example):

(a) $wait_j$: Waiting time, which is the current timestamp minus the job submit time.

(b) re_j: Requested runtime.

(c) q_j: Number of requested processors.

(d) pow_j: Job power consumption, i.e., the power consumption caused by running the job.

(e) pow_j^{per}: Power consumption per processor, which is the ratio of job power consumption to the number of processors.

(f) ibc_j: Indicates if executing the job immediately uses brown energy; 1 for yes, 0 for no.

(g) $brat_j$: Estimated ratio of brown energy consumption to total energy consumption for immediate job execution, with a value ranging from 0 to 1.

(h) csn_j: Indicates whether remaining resources can support immediate job scheduling (1 if yes, 0 if no).

It is noteworthy that when calculating (f) and (g), predicted values of future renewable energy generation are needed, but power supplier can only provide forecasts for a limited range (set to one day in this work). To address this, we set predicted values for hours exceeding this range to those of corresponding hours within the range, e.g., using the 2nd hour's prediction for the 26th hour.

Running Jobs Information: Similar to waiting queue information, we fix the number of running jobs in the running jobs information to a window size M (set at 64). If the number of running jobs exceeds M, select the first M jobs with the earliest completion time as the jobs within the window. The running jobs information is a $[M, 4]$ matrix, with each row containing attributes: number of requested processors q_j, job power consumption pow_j, power consumption per processor pow_j^{per}, and estimated remaining execution time le_j.

Renewable Energy Information: In this work, the HPC cluster receives hourly renewable energy predictions for the next 24 h, totaling 24 time slots. We represent the information for each time slot as a $[1, 2]$ vector, so the renewable energy information is a $[24, 2]$ matrix. Each time slot contains the following elements (taking time slot s as an example): remaining duration $rlst_s$ and renewable energy generation $rpow_s$.

(2) Action space

Our proposed model has a multi-action space, where each action at a time step consists of two sub-actions, denoted as $a_t = \{a_t^{job}, a_t^{delay}\}$. Among them, a_t^{job} is the job selection action at time step t, a_t^{delay} is the delay decision action at time step t. Next, we will introduce these two sub-actions in detail.

Job Selection Action a_t^{job}: This action is a discrete action responsible for selecting a job from the waiting queue. At time step t, if the model outputs action a_t^{job}, it means the job at index a_t^{job} in the queue is chosen.

Delay Decision Action a_t^{delay}: This action occurs after job selection and determines whether to delay the execution of the selected job and for how long. To enhance training efficiency, we have designed a discrete action space with few possible actions, including the following three types:

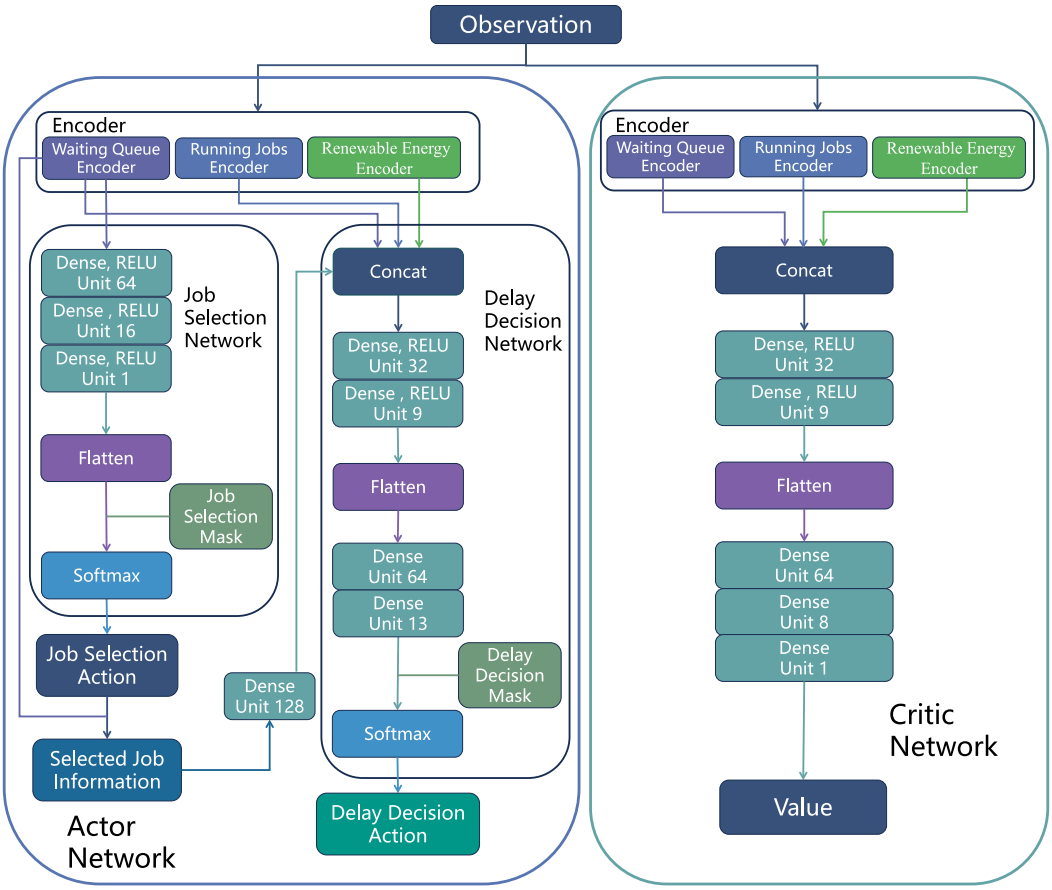

Fig. 2. The network structure of GAS-MARL.

(a) Type 1: No delay; attempt to execute the selected job immediately.

(b) Type 2: Delay until RN currently running jobs have completed (to avoid excessive waiting time, the maximum delay is set to 3600 s), then start attempting to execute the selected job.

(c) Type 3: Select a delay time D from the delay time candidate list DS to delay, and start trying to execute the selected job after the delay is over. where RN is limited to a maximum of 5, and DS is set to [300, 600, 1200, 1800, 2400, 3000, 3600], where the unit is second. Thus, there are 13 possible choices for the delay decision action.

(3) Reward function

This work addresses a bi-objective optimization problem, optimizing both renewable energy utilization and average bounded slowdown. Since these metrics are only calculable after the full job sequence is scheduled, we designed a sparse reward function that provides meaningful rewards only when the entire sequence is completed, with rewards set to 0 at all other decision steps, as follows:

$$r_t = \begin{cases} ReUtil - \eta \times AvgBSLD, & \text{if done;} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

where r_t is the reward at time step t, $ReUtil$ is the renewable energy utilization for the trajectory (as in Eq. (4)), $AvgBSLD$ is the average bounded slowdown for all jobs in the trajectory (as in Eq. (5)), and η is a penalty factor that adjusts the importance of average bounded slowdown.

4.2.2. Network architecture

The network architecture of GAS-MARL is shown in Fig. 2. It is based on PPO and includes two separate networks: the actor and the critic. The actor network generates job selection action and delay decision action, while the critic network provides an estimated state value function to support the calculation of the advantage function. Below, we will detail the design of the actor and critic networks. The GAS-MARL actor network comprises three state encoders: waiting queue encoder, running jobs encoder, and renewable energy encoder, each handling respective information. Each encoder consists of two fully connected layers (64 and 128 neurons, both using RELU activation function). The rest of the actor network is divided into two sections: job selection network and delay decision network.

The job selection network uses the waiting queue encoder's output. It processes this through fully connected layers with RELU activation and then applies a Softmax function to produce an action probability distribution of size $[bs, Q]$, where Q is the window size of the waiting queue and bs is the batch size of the input. The delay decision network utilizes the outputs from all the state encoders, combines them with the selected job information (after dimension transformation via a fully connected layer), then processes them through fully connected layers with RELU activation, finally generating an action probability distribution with dimensions $[bs, 13]$ via the Softmax function. Since invalid options may exist for both action types during a decision period, we employ the action mask mechanism to conceal them, thereby enhancing training efficiency.

The critic network uses the same state encoders as the actor network, and the remaining network structure is similar to the delay decision network of the actor network. Its output is an estimate of the value of the current state.

4.2.3. PPO-based policy optimization algorithm

We use PPO-Clip [27] for model training. It employs a simple clipping function to limit the difference between old and new policies in the actor network, preventing drastic changes and enhancing training stability. The objective functions for optimizing the actor and critic

networks are shown in Eqs. (10) and (11), respectively.

$$L^{clip}(\theta) = \hat{E}_t[min(rat_t(\theta)\hat{A}_t, clip(rat_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{10}$$

$$L^{V}(\emptyset) = \hat{E}_t[MSE(R_t, V_\emptyset(s_t))] \tag{11}$$

where s_t is the state at time step t, $V_\emptyset(s_t)$ is the critic network's output, R_t is the rewards-to-go, which represents the cumulative reward from the current step to the end of the trajectory. MSE is the mean squared error function, $rat_t(\theta)$ is the probability ratio between the new and old policies, ϵ is the clipping ratio (set at 0.2), and clip is a clipping function that restricts $rat_t(\theta)$ to lie within the interval $[1-\epsilon, 1+\epsilon]$, and $\hat{A}_t$ is the advantage estimation value. Our model outputs two actions per time step. To handle this multi-action space, we use a composite probability ratio [28,29], which multiplies the probabilities of the two sub-actions to get the composite probability ratio. Thus, $rat_t(\theta)$ can be expressed as follows:

$$rat_t(\theta) = \frac{\pi_\theta(a_t^{job}|s_t) \times \pi_\theta(a_t^{delay}|s_t)}{\pi_{old}(a_t^{job}|s_t) \times \pi_{old}(a_t^{delay}|s_t)} \tag{12}$$

where $\pi_\theta(a_t^{job}|s_t)$ and $\pi_\theta(a_t^{delay}|s_t)$ are the probabilities of the two actions under the new policy, while $\pi_{old}(a_t^{job}|s_t)$ and $\pi_{old}(a_t^{delay}|s_t)$ are the probabilities of the two actions under the old policy.

For calculating the advantage estimate $\hat{A}_t$, this work uses the Generalized Advantage Estimation (GAE) algorithm [30], as follows:

$$\hat{A}_t = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+1}^V \tag{13}$$

$$\delta_t^V = r_t + \gamma V_\emptyset(s_{t+1}) - V_\emptyset(s_t) \tag{14}$$

where γ is the discount factor that balances the importance of current rewards and future rewards, λ is the GAE parameter that controls the trade-off between bias and variance, and δ_t^V is the temporal difference error. In this work, we set γ and λ to 1 and 0.97, respectively.

In each training episode, we sample $trajNum$ trajectories, each containing scheduling decisions for $SeqNum$ consecutive jobs. Experience tuples (s_t, a_t, r_t) are stored in buffer D. After sampling, we calculate R_t and $\hat{A}_t$, then update the actor and critic networks in multiple rounds using mini-batches. Next, we clear D and start a new episode, repeating until we complete $IterNum$ iterations. In our setup, $trajNum$, $SeqNum$, and $IterNum$ are set to 100, 256, and 300, respectively. Algorithm 1 provides the pseudocode for the GAS-MARL training process.

Algorithm 1: GAS-MARL Algorithm

input : Initial actor network π_θ and critic network $V_\emptyset$
output: π_θ and $V_\emptyset$

```
for Iteration ← 1 to IterNum do
    for traj ← 1 to trajNum do
        Reset the environment;
        for t ← 1 to seqNum do
            Sample action a_t(a_t^job, a_t^delay) based on π_θ(a_t|s_t);
            Execute a_t(a_t^job, a_t^delay), get reward r_t and next state
              s_(t+1);
            Store (s_t, a_t, r_t) into memory D;
        end
    end
    Compute rewards-to-go R_t and advantage estimates Â_t;
    for epoch ← 1 to epochsNum do
        Split the training batch into K mini-batches;
        for each mini-batch do
            Compute L^clip(θ) and L^V(∅) by Eqs. (10), (11);
            Update π_θ using Adam(∇_θ L^clip(θ));
            Update V_∅ using Adam(∇_∅ L^V(∅));
        end
    end
    clear memory D;
end
```

4.3. Green-backfilling

In HPC clusters, to prevent large jobs from starving, the resource reservation mechanism is often used, which reserves available resources for the job that cannot be executed immediately to ensure it can be executed as soon as possible. At the same time, to reduce resource waste caused by reserving resources for high-priority job, HPC clusters generally combine the use of backfilling policy [16]. Backfilling enables jobs in the waiting queue to execute earlier if they meet certain conditions, improving resource utilization while preventing starvation of large jobs. To better cooperate with GAS-MARL for job scheduling, we have designed a new backfilling policy called Green-Backfilling.

During backfilling, the priority rules for selecting candidate jobs significantly impact scheduling effectiveness. To prioritize jobs with low power consumption, few requested processors, and short running time, we have designed the following priority function:

$$L_j = re_j \times q_j \times p_j \tag{15}$$

where L_j is the priority value of job j, re_j is the requested runtime of job j, q_j is the number of processors requested by job j, and p_j is the power consumption of job j.

Additionally, to mitigate the impact of backfilling on renewable energy utilization, we have introduced a job acceptance criterion, which is to estimate the brown energy consumption caused by the job if it starts at the current timestamp, and check whether it exceeds the threshold σ (set at $50\,000J$). If not, the job can be executed.

Lastly, conventional backfilling policies are only triggered when a job is blocked, but to accommodate the delay decision action of the GAS-MARL, our proposed Green-Backfilling is also triggered when the delay decision action is of type 2 or type 3, with specific rules as follows:

(1) When the delay decision action is of type 2 and needs to wait until RN currently running jobs are completed, first estimate the completion timestamp ts_{RN} for the RN jobs and the earliest executable timestamp ts_{ES} for the blocking job. Notably, to avoid excessive wait times for the blocking job, if $ts_{RN} > t_c + 3600$ (where t_c is the current timestamp), set ts_{RN} to $t_c + 3600$. Finally, set the maximum allowable completion timestamp ts_{LM} for the backfilling to the greater of ts_{RN} and ts_{ES}, and perform the backfill accordingly. For this action type, terminate the backfilling when the blocked job can run and is delayed until the RN currently running jobs are completed.

(2) When the delay decision action is of type 3 and the delay time is DT, first estimate the earliest executable timestamp ts_{ES} for the blocking job. Then, set the maximum allowable completion timestamp ts_{LM} for the backfilling to the greater of ts_{ES} and $t_c + DT$, and perform backfilling based on this. For this action type, terminate the backfilling when the blocked job can run and the delay is at least DT.

The pseudocode of Green-Backfilling is shown in Algorithm 2. Line 1 checks if the backfilling meets the termination condition. Line 2 sorts the waiting job queue WQ according to Eq. (15). Line 3 calculates the maximum allowable completion timestamp ts_{LM} for the backfilling. Lines 4–11 select and execute jobs from queue WQ that can start immediately, with an expected completion timestamp less than ts_{LM}, and an estimated brown energy consumption below the threshold σ. Line 12 lets the scheduler wait until the next timestamp for backfilling, which occurs at the timestamp of any of the following three events: a new job submission, completion of a running job, or change in current renewable energy generation.

Algorithm 2: Green-Backfilling

input : Waiting job queue WQ
output: New system state

while *backfilling termination condition is not met* **do**
  Sort WQ according to Eq. (15);
  Compute the maximum allowable completion timestamp ts_{LM} for the backfilling;
  for each job j in WQ **do**
    Estimated the brown energy E_j^B consumed by job j if it starts now;
    if the remaining resources are sufficient for job j **then**
      if $pe_j + ts_c < ts_{LM}$ *and* $E_j^B < \sigma$ **then**
        Start executing job j;
      end
    end
  end
  Wait until the next timestamp for backfilling;
end

5. Experiment

5.1. Datasets and data pre-processing

Table 1 lists the job trace datasets used in this work, selected because they have been widely used by many researchers [7,12,31]. Lublin-256, Cirne, and Jann are synthetic traces generated based on workload models proposed in [32,33], and [34], respectively. For Cirne, the job arrival pattern is configured as ANL. Since these workload models do not provide power consumption for each job, we will generate these data ourselves. Note that randomly generating power may lead to unrealistic scenarios where jobs occupying many processors have very low power consumption. Therefore, we first randomly generate power consumption per processor for each job and then calculate the total power consumption. The method for calculating job power consumption is as follows:

$$pow_j = q_j \times pow_j^{per} \tag{16}$$

where pow_j represents the power consumption of job j, q_j represents the number of processors requested by job j, and pow_j^{per} represents the power per processor for job j, which is obtained through random generation within the integer range of $[5, 50]$.

For modeling the power generation data of solar and wind energy, we first establish models between power generation and key weather variables, and then input real historical weather data to calculate the power generation at each timestamp. For modeling solar and wind energy, we adopt the method mentioned in [35], specifically as follows:

$$P_{solar}(s) = \alpha \times A \times irr(s) \tag{17}$$

$$P_{wind}(s) = \begin{cases} 0, & \text{if } v(s) \le v_{in} \text{ or } v(s) \ge v_{out}; \\ P_r \times \frac{v(s)-v_{in}}{v_r - v_{in}}, & v_{in} < v(s) < v_r; \\ P_r, & v_r < v(s) < v_{out} \end{cases} \tag{18}$$

where α and A represent the photovoltaic conversion efficiency and effective irradiated area of the photovoltaic panel, respectively, $irr(s)$ is the solar irradiance at timestamp s, $v(s)$ is the wind speed at timestamp s, P_r is the rated power of the wind turbine, v_r is the rated wind speed, v_{in} is the cut-in wind speed required for turbine to start generating electricity, and v_{out} is the cut-out wind speed at which the turbine stops to avoid damage. Table 2 lists the parameters of the photovoltaic panels and wind turbines used in this work. Additionally, due to variations in cluster sizes across job traces, we will proportionally scale the power generation of wind and solar energy based on the number of processors (with 256 of Lublin-256 as the reference value). Our solar irradiance and wind speed data are sourced from NSRDB [36] and WRD [37], measured in San Francisco, USA, from January to December 2016, with hourly intervals.

Table 1
List of job traces.

Job trace	Number of processors	Number of jobs
Lublin-256	256	10 000
Cirne	256	10 000
Jann	322	10 000

Table 2
The parameters of photovoltaic panels and wind turbines.

α	A (m^2)	P_r (watt)	v_r (m/s)	v_{out} (m/s)	v_{in} (m/s)
0.2	200	7200	15	30	2.5

5.2. Simulation environment

This study evaluates GAS-MARL and Green-Backfilling using simulation experiments. Our simulator, developed based on SchedGym [12], can read job trace datasets in the Standard Workload Format (SWF) [38], simulate the corresponding clusters and job requests, and assess the performance of various algorithms by scheduling submitted jobs. Notably, since SchedGym cannot simulate power consumption or renewable energy, we added a cluster power consumption module and a renewable energy module. The power consumption module maintains a record of the cluster's total power consumption over time based on job power data during scheduling. The renewable energy module calculates solar and wind energy generation over time using irradiance and wind speed data and computes the renewable energy utilization of the cluster over a given period.

The following describes the details of the simulation process of the simulator: At the beginning of a simulation, the cluster starts in an idle state, and jobs are loaded one by one from the specified starting point in the dataset. When a new job arrives or a running job finishes and releases resources (based on the job's runtime information), the simulator schedules jobs according to the actions returned by the scheduling algorithm module, continuing until there are insufficient available resources to execute the selected job, leading to a blockage. If a blockage occurs and backfilling is enabled, the simulator supports selecting jobs from the waiting queue to execute early, based on the backfilling policy. During job scheduling, the simulator continuously maintains records of the system's remaining resources, the start time of scheduled jobs, and the changes in total cluster power consumption. At the end of the simulation, the simulator calculates two metrics: the average bounded slowdown and the renewable energy utilization, based on the data collected throughout the simulation.

Notably, to facilitate the simulation study, all machines used in the clusters for each dataset are set to the same model, with each machine having 8 processors and an idle power consumption of 50 W.

5.3. Comparison methods

To evaluate the performance of GAS-MARL, we compared it with four other algorithms, namely First Come First Serve (FCFS), F2, Largest Processing Time times Power Need first (LPTPN), GA, and PPO. The following is a brief description of these algorithms:

(1) FCFS [4]: This algorithm is often combined with EASY-Backfilling and is the default scheduling strategy in many production supercomputing systems [16]. Adopting it as a baseline can demonstrate the potential improvements of the proposed method in practical scenarios. The algorithm sorts the waiting queue by job submit time in

ascending order and attempts to execute jobs in sequence until a blockage due to insufficient resources.

(2) F2 [5]: This algorithm is a superior heuristic HPC job scheduling algorithm for optimizing average bounded slowdown. It uses a priority function derived from simulation and nonlinear regression to sort the waiting queue, and then employs a process similar to FCFS for scheduling. In the experiments, its primary role is to provide a reference for GAS-MARL in determining penalty factor η, enabling the model to optimize renewable energy utilization while ensuring that the trained model achieves a competitive average bounded slowdown.

(3) LPTPN [19]: The algorithm is a heuristic HPC job scheduling algorithm that considers renewable energy. It is relatively close to the problem studied in this work, and therefore, we use it as a baseline. The original algorithm focuses on HPC clusters powered solely by renewable energy. To adapt it for this study, we modified it to the following version: At the start of each scheduling period, jobs in the waiting queue are first sorted in descending order based on the product of their requested runtime and power consumption. Then, the job queue is traversed from front to back, and a job is scheduled only if its resource requirements are met and its estimated brown energy consumption at the current time is below a threshold σ (set to $50\,000J$, as in Green-Backfilling). After traversing all jobs in the queue, scheduling halts until the next scheduling timestamp. Notably, if no jobs can be executed in the queue at a given timestamp and no jobs are running on the cluster, the first job in the queue will be executed to prevent high-power jobs from being indefinitely delayed. Additionally, unlike other algorithms, this algorithm lacks resource reservation and delay mechanisms, making backfilling infeasible. Therefore, in the experiments, we will only evaluate the performance of this algorithm without backfilling.

(4) GA [3]: This algorithm is a metaheuristic-based HPC job scheduling algorithm that considers renewable energy optimization. Using this metaheuristic algorithm as a baseline allows for a better analysis of GAS-MARL's performance advantages. The algorithm employs a genetic algorithm to generate the execution order of jobs in the current waiting queue, using roulette wheel selection, chunk mutation, and order crossover (details in [3]). We set the population size to 30 and the maximum iterations to 30. For fairness, the optimization objective is similar to the reward function in Eq. (9) of GAS-MARL, with the penalty factors set consistently with GAS-MARL.

(5) PPO [17]: A Maskable PPO-based job scheduler selects jobs from the waiting queue at each decision time step. Its key difference from GAS-MARL is the absence of delay decision action, allowing it to control only the execution order without delaying selected jobs. We use this algorithm as a baseline to evaluate the performance differences between GAS-MARL and the previous PPO-based HPC job scheduling algorithms, particularly in terms of renewable energy utilization, thus demonstrating the effectiveness of our innovations. For fair comparison, the network structure is the same as the job selection network in GAS-MARL, using the same reward function and model training parameters.

To comprehensively evaluate the performance of Green-Backfilling, we test its combination with all scheduling algorithms in the experiments. Since Green-Backfilling includes a triggering mechanism specifically designed for the delay decision action of GAS-MARL, to adapt it for other algorithms, we have modified it as follows: the triggering condition is changed to trigger only when a blockage occurs due to insufficient resources to execute the selected job, while keeping other settings unchanged. Additionally, EASY-Backfilling [7] is used as a baseline for comparison with Green-Backfilling. This backfilling policy iterates through jobs in the waiting queue in the order of submission and allows jobs to execute early if they do not delay the frontmost job.

5.4. Experimental results

5.4.1. Determining the appropriate penalty factor η

As shown in Section 4.2.1, in GAS-MARL, we use the penalty factor η to balance the two optimization objectives, and its value has a significant impact on the algorithm's performance. In this section, we will determine appropriate values for η through experimentation. The experimental setup is as follows: we first train the model using different η values (ranging from 0.001 to 0.01, with an interval of 0.001, totaling 11 values) under each job trace. After training, we tested each model by scheduling ten independent job sequences of length 1024 and analyzed the impact of different η values on the two optimization objectives based on the average of the ten sampling results. The reason for setting the sequence length for one evaluation sampling to be greater than that during training was to assess the model's generalization ability. Additionally, Green-Backfilling was enabled in both the training and evaluation stages of the experiment for GAS-MARL. Fig. 3 shows the impact of different η values on the two optimization objectives across various job traces.

Analysis of Fig. 3 shows that the impact of the same η value on different job traces varies. Therefore, selecting the appropriate η value for each job trace is crucial. Additionally, the figure reveals that it is difficult to select a single η value that allows renewable energy utilization and average bounded slowdown to simultaneously reach their optimal levels. Thus, the selection of the penalty factor is a trade-off between the two optimization objectives. In practical production environments, the choice of η should be based on the preferences and needs of the HPC cluster managers and users. In this work, we use the results of F2 (a heuristic algorithm that optimizes bounded slowdown) under the same experimental settings as a reference. We select the η values with average bounded slowdown comparable to or better than F2, and then choose the η value with the highest renewable energy utilization. According to the selection strategy mentioned above, this work sets the η value for Lublin-256 to 0.002, for Cirne to 0.008, and for Jann to 0.003. Experimental data shows that the selected η values enable the GAS-MARL algorithm to train scheduling models that achieve competitive performance in average bounded slowdown, comparable to or better than F2, while also attaining relatively high renewable energy utilization. These parameter settings will be used in the following experiments.

5.4.2. Impact of Green-backfilling on GAS-MARL training and performance

In GAS-MARL, we have the option to enable or disable backfilling policy during training, which may impact final performance. Therefore, we will experimentally analyze the effect of our proposed Green-Backfilling on algorithm training. The experimental setup is as follows: during the training phase, two types of GAS-MARL models are trained on each job trace with Green-Backfilling enabled and disabled. After the training phase, we assess the performance under three configuration: (A) Green-Backfilling enabled during both training and evaluation; (B) Green-Backfilling enabled only during evaluation; (C) Green-Backfilling disabled during both training and evaluation. Performance is assessed by sampling 10 job sequences of size 1024 for scheduling. Figs. 4 and 5 present detailed comparison results of renewable energy utilization and average bounded slowdown for GAS-MARL under these three configurations and different job traces, using box plots. In these charts, the middle line represents the median, the box range reflects the 25% to 75% quantile interval, and the square points represent the average value of the data.

Analyzing Figs. 4 and 5, it can be seen that in the three experimental configurations, Setting A demonstrates a significant advantage in renewable energy utilization, with its average value increased by 15.92% to 31.77% compared to Setting B, and by 17.59% to 31.50% compared to Setting C. While Setting A's average bounded slowdown is higher than Setting B's, it shows a significant decrease compared to Setting C. These results suggest that enabling Green-Backfilling during

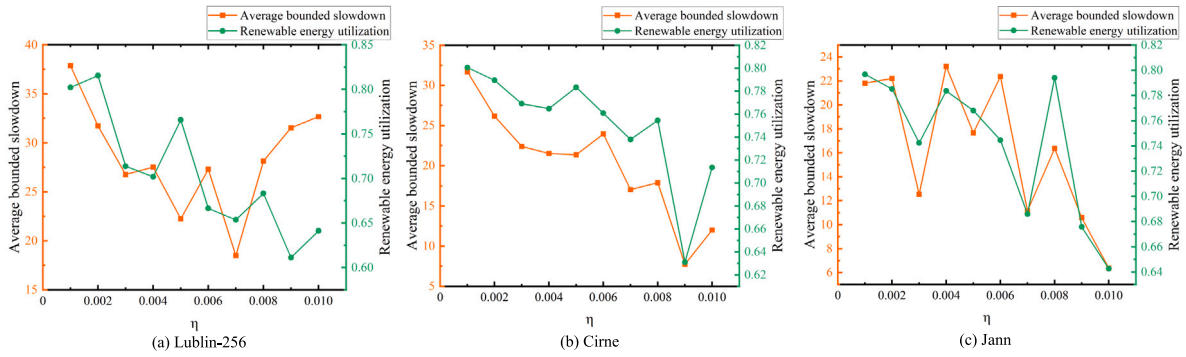

Fig. 3. Average bounded slowdown and renewable energy utilization for different values of η.

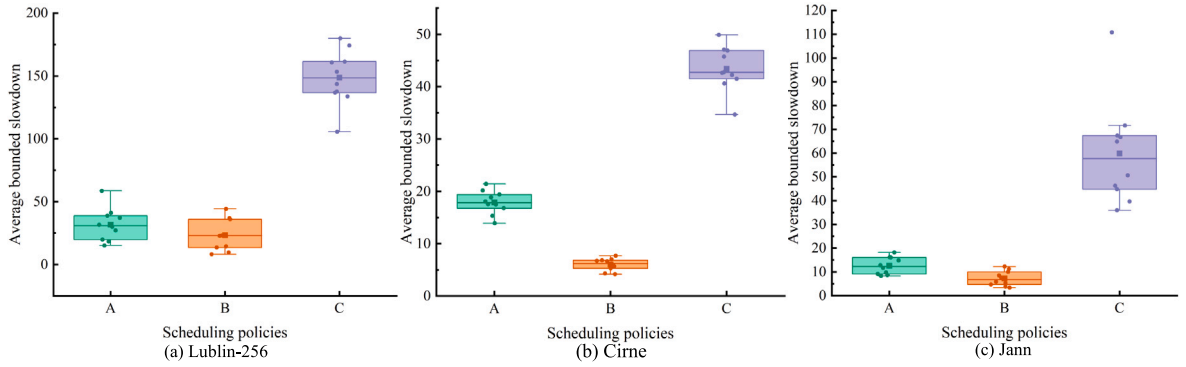

Fig. 4. Performance of each set of experiments for the two algorithms in terms of average bounded slowdown.

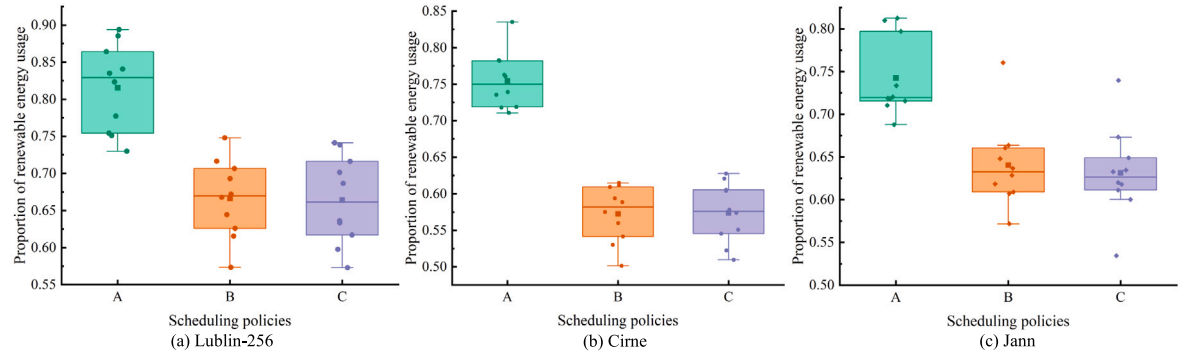

Fig. 5. Performance of each set of experiments for the two algorithms in terms of renewable energy utilization.

model training is beneficial for achieving better overall scheduling performance, as it enables the model to learn the impact of backfilling on job scheduling. Therefore, we will adopt Setting A for subsequent experiments, enabling backfilling during both training and testing. Additionally, compared to Setting C, Setting B reduces the average bounded slowdown by 84.41% to 87.89%, while their average renewable energy utilization is quite close. This suggests that even if Green-Backfilling is enabled only during the evaluation phase, it can effectively reduce the average bounded slowdown without sacrificing renewable energy utilization.

5.4.3. Comparison of training efficiency and convergence between GAS-MARL and PPO

In this section, we compare the training efficiency and convergence of the GAS-MARL and PPO algorithms. The objective of both algorithms during training is to maximize cumulative rewards, so we can evaluate their training efficiency and convergence by observing changes in cumulative rewards across training iterations. The experimental setup is as follows: with Green-Backfilling enabled, both algorithms are trained for 300 epochs on three job traces, and we collect performance data over the training process. Fig. 4 illustrates the training curves of both algorithms on each job trace. The Y-axis in the figure, labeled "performance", represents the mean value of cumulative reward per iteration, which is the mean value of the total reward for 100 sampled job sequences in an iteration (the reward function is outlined in Eq. (9), which represents a weighted sum of two optimization objectives). A higher value indicates better model performance.

As shown in Fig. 6, PPO demonstrates poor training effectiveness and even fails to converge on the Jann job trace. In contrast, GAS-MARL is able to converge after 300 training iterations and outperforms PPO significantly across all job traces. This is because the introduction of delay decision action greatly enhances scheduling flexibility, allowing jobs to be executed during periods with relatively abundant renewable energy. This reduces the difficulty of optimizing renewable energy utilization, thereby improving the training efficiency of the algorithm and enabling it to converge to a higher-performing policy.

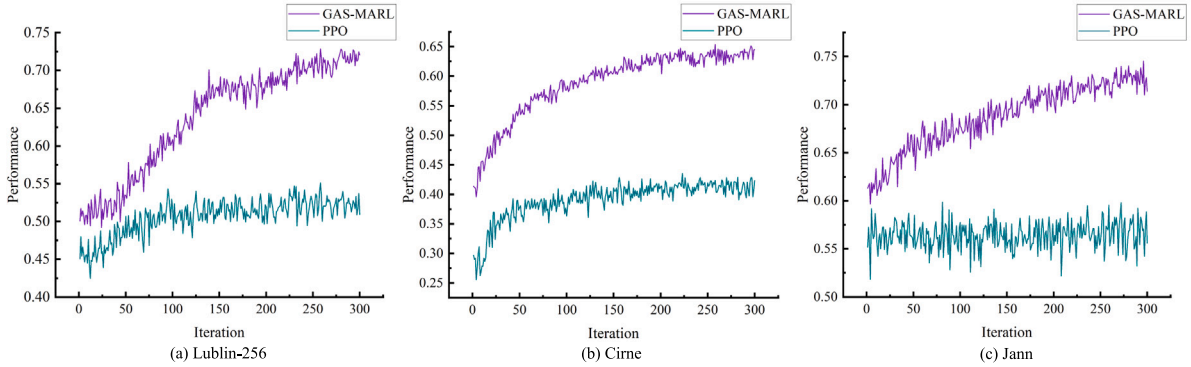

Fig. 6. Training curves of GAS-MARL and PPO under different job traces.

Table 3
Scheduling performance comparison of mean value in average bounded slowdown for each algorithm with different backfilling policies (LPTPN evaluated only under no backfilling).

Backfilling policy	Trace	FCFS	F2	LPTPN	GA	PPO	GAS-MARL
Disable	Lublin-256	5772.113	271.447	341.040	3320.075	303.051	**148.687**
	Cirne	1156.781	64.699	352.140	633.584	82.972	**43.398**
	Jann	237.898	78.365	205.073	175.817	74.645	**59.833**
EASY-Backfilling	Lublin-256	211.380	57.400	–	181.471	**55.262**	63.627
	Cirne	87.489	**20.824**	–	70.116	23.125	34.556
	Jann	14.495	**11.514**	–	15.090	13.751	31.356
Green-Backfilling	Lublin-256	91.001	55.667	–	100.975	56.358	**31.738**
	Cirne	81.564	18.682	–	64.696	22.357	**17.895**
	Jann	16.976	**11.332**	–	19.999	15.374	12.534

Table 4
Scheduling performance comparison of mean value in renewable energy utilization for each algorithms with different backfilling policies (LPTPN evaluated only under no backfilling).

Backfilling policy	Trace	FCFS	F2	LPTPN	GA	PPO	GAS-MARL
Disable	Lublin-256	0.5973	0.5808	**0.6845**	0.6877	0.5926	0.6640
	Cirne	0.5252	0.5179	**0.7872**	0.5798	0.5149	0.5738
	Jann	0.5416	0.5425	**0.7853**	0.5744	0.5387	0.6313
EASY-Backfilling	Lublin-256	0.5635	0.5807	–	0.6166	0.6072	**0.6218**
	Cirne	0.5003	0.5082	–	**0.5365**	0.5122	0.5332
	Jann	0.5389	0.5401	–	0.5442	0.5431	**0.5820**
Green-Backfilling	Lublin-256	0.6186	0.5918	–	0.7505	0.6767	**0.8154**
	Cirne	0.5341	0.5231	–	0.6125	0.5311	**0.7545**
	Jann	0.5513	0.5487	–	0.5790	0.5454	**0.7424**

5.4.4. Performance comparison of each algorithm with different backfilling policies

In this section, we will compare the performance of each scheduling algorithm under different backfilling policies. The experimental setup is as follows: each scheduling algorithm, except for LPTPN, will be evaluated under three configurations: no backfilling, enabling EASY backfilling, and enabling Green backfilling. The LPTPN algorithm, due to its inability to support backfilling, will be evaluated only under the no backfilling configuration. The performance will be evaluated by scheduling 10 independent job sequences, each with a length of 1024, on the three job traces. It is worth noting that, for experiments with EASY-Backfilling and Green-Backfilling enabled, both GAS-MARL and PPO will apply the corresponding backfilling policies during both the training and evaluation phases. Tables 3 and 4 show the average bounded slowdown and renewable energy utilization of each algorithm under different backfilling policies across three job traces, representing the average of 10 sampling results.

From Table 3, it can be observed that GAS-MARL consistently exhibits the lowest average bounded slowdown across all job traces, both without backfilling and with Green-Backfilling enabled. When Green-Backfilling is enabled, GAS-MARL reduces the average bounded slowdown by 26.17% to 78.06% compared to FCFS, by 37.33% to 72.34% compared to GA, and by 18.47% to 43.69% compared to PPO. Compared to F2, while the average bounded slowdown increases by 10.61% on Jann, it decreases by 42.99% on Lublin-256 and 4.21% on Cirne. However, with EASY-Backfilling, GAS-MARL shows suboptimal performance in average job slowdown across all job traces. Notably, when Green-Backfilling is enabled, GAS-MARL experiences a significant reduction in average bounded slowdown, ranging from 48.21% to 79.05%, compared to both no backfilling and EASY-Backfilling, indicating that Green-Backfilling effectively lowers the average bounded slowdown for GAS-MARL.

Table 3 reveals an interesting observation: compared to PPO, GAS-MARL improves renewable energy utilization by introducing delay decision action, and its delay mechanism does not significantly negatively impact average bounded slowdown. On the contrary, with either no backfilling or Green-Backfilling enabled, it performs better than PPO in terms of average bounded slowdown. By analyzing the scheduling trajectories of the two algorithms under different backfilling policies, we identified the following reasons for this outcome: In the no backfilling case, although proactive delays may temporarily pause the start of more jobs at the current timestamp, moderate pauses provide an opportunity to wait for the arrival of additional new jobs. This allows the agent to gain more scheduling options, enabling the earlier execution of jobs with higher 'cost-effectiveness' (e.g., jobs requiring very few processors and a short runtime) that arrive during the delay, thereby facilitating the optimization of average bounded slowdown. When Green-Backfilling is enabled, in addition to the above reason (in

the no backfill case), the backfilling policy introduces a mechanism that backfills during the delay period, effectively utilizing the delay time to execute small jobs from the waiting queue that do not interfere with the delayed job, thereby benefiting the optimization of the average bounded slowdown.

Another interesting observation in Table 3 is that in Green-Backfilling, a limit is imposed on the brown energy consumption of jobs during backfilling, reducing the range of jobs eligible for backfilling compared to EASY-Backfilling. However, experimental results show that the average bounded slowdown of the two heuristic algorithms (FCFS and F2) with Green-Backfilling enabled is lower than that of EASY-Backfilling. We hypothesize that this is mainly due to differences in the priority functions used to select jobs for backfilling in the two backfilling policies. To test this, we replaced the priority function of EASY-Backfilling with that of Green-Backfilling and evaluated its impact on FCFS and F2. The results show that the average bounded slowdown of FCFS on Lublin-256, Cirne, and Jann are 140.955, 65.250, and 14.152, respectively, while for F2, they are 54.170, 18.888, and 11.016. It can be observed that after replacing the priority function of EASY-Backfilling, both algorithms show a significant reduction in average bounded slowdown across all job traces, with F2 outperforming its results with Green-Backfilling on all traces, and FCFS outperforming its results with Green-Backfilling except on Lublin-256. These results suggest that the lower average bounded slowdown of the two heuristic algorithms with Green-Backfilling enabled compared to EASY-Backfilling is primarily due to the differences in their priority functions.

Analysis of the data in Table 4 clearly shows that GAS-MARL exhibits a significant advantage in renewable energy utilization. Specifically, when Green-Backfilling is enabled, GAS-MARL demonstrates an increase in renewable energy utilization ranging from 31.82% to 41.26% compared to FCFS, 35.29% to 44.23% compared to F2, 8.65% to 28.20% compared to GA, and 20.50% to 42.04% compared to PPO. Similarly, GAS-MARL shows varying degrees of advantage in renewable energy utilization over other algorithms when EASY-Backfilling is enabled and when no backfilling is used. Notably, without backfilling, GAS-MARL's renewable energy utilization is lower than that of LPTPN and GA on some datasets, but the average bounded slowdown of these two algorithms is much higher than that of GAS-MARL, highlighting GAS-MARL's overall performance advantage. Furthermore, enabling Green-Backfilling significantly increases GAS-MARL's renewable energy utilization by 17.59% to 41.50% compared to no backfilling and EASY-Backfilling, indicating its beneficial impact on enhancing renewable energy utilization for the GAS-MARL.

A comprehensive analysis of the data in Tables 3 and 4 reveals that, in terms of average bounded slowdown, the combination of GAS-MARL and Green-Backfilling is the best among all combinations on Lublin-256 and Cirne, and it performs only slightly worse than F2 with Green-Backfilling or EASY-Backfilling on Jann. In terms of renewable energy utilization, the combination of GAS-MARL and Green-Backfilling is the best on Lublin-256 and is only slightly lower than LPTPN on Cirne and Jann. Notably, F2's renewable energy utilization is much lower than GAS-MARL across all experimental setups. LPTPN also shows a much higher average bounded slowdown than the GAS-MARL and Green-Backfilling combination. Therefore, considering both optimization objectives, the combination of GAS-MARL and Green-Backfilling outperforms any other scheduling algorithm and backfilling policy combination. Specifically, compared with the combination of FCFS and EASY-Backfilling, which is the most widely used scheduling policy [16], our proposed algorithm can increase renewable energy utilization by 37.74% to 50.80% and reduce the average bounded slowdown by 13.53% to 84.99%. Additionally, the experimental results indicate that all algorithms exhibit considerable variations in renewable energy utilization and average bounded slowdown across different job traces, primarily due to significant differences in job arrival patterns, job characteristics, and cluster sizes.

To further analyze scheduling characteristics, we collected data from various algorithms sampling the same job sequence of size 1024 in the Lublin-256 job trace with Green-Backfilling enabled (except for LPTPN, which was evaluated without backfilling). We plotted changes in cluster power consumption and renewable energy generation power during scheduling for different algorithms (using 48-h data), as shown in Fig. 7. The figure shows that under this experimental setup, GAS-MARL consumes less brown energy (i.e., the blue area exceeding the green line) within 48 h compared to other algorithms. It maintains lower cluster power consumption during low renewable energy generation due to: (1) optimizing job execution order to minimize execution of high-power jobs during low renewable energy generation periods; (2) The introduction of delay decision action enhances the algorithm's flexibility, enabling it to proactively delay high-power jobs and execute them during periods of relatively abundant renewable energy generation.

5.5. Computational cost

The computational cost of the proposed algorithm in terms of time overhead can be divided into two parts: GAS-MARL training and scheduling decision-making. We will analyze these costs using our evaluation platform (Intel Xeon Gold 5318Y CPU, RTX 4090 GPU, 256 GB RAM) as an example.

With Green-Backfilling enabled and 300 training epochs, the total training times for the GAS-MARL algorithm are approximately 4.25 h on the Lublin-256, 6.25 h on the Cirne, and 6.50 h on the Jann. These differences in training times are primarily due to significant variations in cluster sizes, job characteristics, and arrival patterns across the job traces. Although model training requires some time, once the model training is completed, its decision-making speed is extremely fast, requiring only 2 ms for a single scheduling decision, which is fully acceptable in real-world production environments.

Next, we discuss the computational energy costs of the algorithm. The energy consumed by the algorithm is minimal for several reasons. First, both model training and scheduling decision-making only require the use of a single cluster management node, without the need for extensive computational resources, meaning the energy consumed by the algorithm is negligible compared to the total energy consumption caused by the large number of HPC jobs running on the cluster. Second, while relatively high-power computation is required during the training phase, the scheduling decision-making phase only involves quick inference upon receiving a request, which consumes significantly less power. In summary, the energy overhead of the algorithm is negligible for HPC clusters with thousands of nodes in production environments, and as a result, the energy consumption of model training and inference is not considered when calculating renewable energy utilization in the experiments.

6. Conclusion and outlook

This paper first analyzes the issue of rapidly increasing carbon emissions due to the expansion of HPC cluster scale, as well as the challenges of using renewable energy to reduce emissions, including its variability and intermittency. To address these issues, we propose a PPO-based job scheduling algorithm for HPC clusters, named GAS-MARL. Compared to existing work, this algorithm stands out with its multi-action space design, which not only determines job execution order but also decides if and how long jobs should be delayed, providing more flexibility. We also developed Green-Backfilling, a new backfilling policy for GAS-MARL, aimed at reducing average bounded slowdown without sacrificing renewable energy utilization. Experimental results show that the combination of GAS-MARL and Green-Backfilling offers significant advantages in renewable energy utilization and average bounded slowdown. Compared to the combination of FCFS and EASY-Backfilling, it enhances renewable energy utilization by 37.74% to

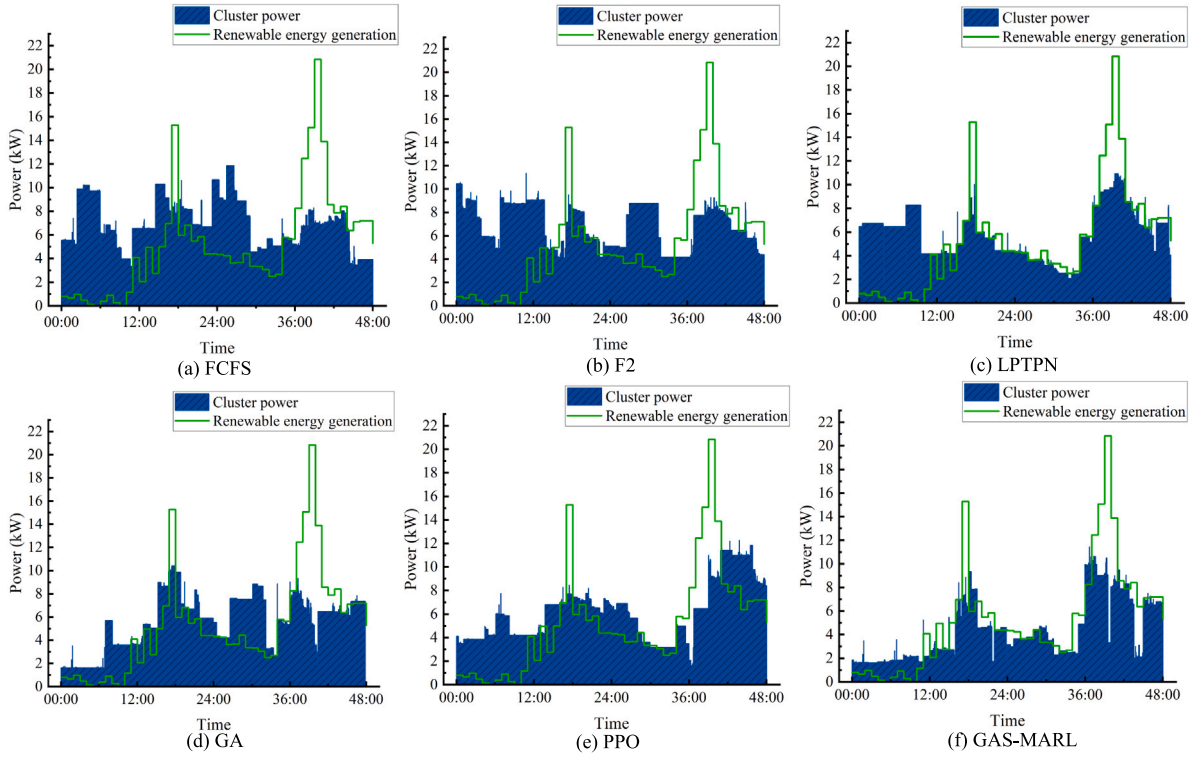

Fig. 7. Cluster power consumption and renewable energy generation power in each time slot for different scheduling algorithms under Lublin-256 job trace.

50.80% and reduces average bounded slowdown by 13.53% to 84.99%. Notably, if we can tolerate a certain increase in average job slowdown, we can further enhance renewable energy utilization by adjusting model training parameters.

The limitation of this study is that the results obtained were entirely based on simulations, using synthetic traces and randomly generated power consumption information. Therefore, future work will focus on collecting and analyzing power consumption data from various HPC applications, and using these data along with job traces from real clusters to create more reliable and diverse datasets. These datasets will be used to validate and refine the proposed algorithms, with the ultimate goal of deploying and testing them in production environments on real clusters. Additionally, our work is currently focused on job scheduling optimization in homogeneous HPC clusters. We plan to extend the scheduling algorithm to heterogeneous HPC clusters with CPUs and GPUs to enhance its versatility. We also plan to explore power systems with energy storage, integrating them into our scheduling algorithms. This will enable energy storage when the renewable energy generation are abundant and energy release when it is scarce, tackling intermittency and cutting carbon emissions.

CRediT authorship contribution statement

Rui Chen: Writing – original draft, Software, Methodology. **Weiwei Lin:** Writing – review & editing, Methodology, Funding acquisition. **Huikang Huang:** Writing – review & editing, Methodology. **Xiaoying Ye:** Writing – review & editing. **Zhiping Peng:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is supported by Guangdong Major Project of Basic and Applied Basic Research (2019B030302002), National Natural Science Foundation of China (62072187, 62273109), and the Major Key Project of PCL, China under Grant PCL2023A09.

Data availability

The source code and datasets used in this research are publicly available at https://github.com/sioncr/green-rl-sched.

References

[1] TOP500 Team, Top500 list - 2024, 2024, https://www.top500.org/lists/top500/2024/06/ (Accessed 17 2024).
[2] A. Jones, Digital goes green: More data centers migrating to renewable energy, 2023, https://www.ispartnersllc.com/blog/data-centers-renewable-energy/ (Accessed 17 2024).
[3] A. Kassab, J.-M. Nicod, L. Philippe, V. Rehn-Sonigo, Green power aware approaches for scheduling independent tasks on a multi-core machine, Sustain. Comput. Inf. Syst. 31 (2021) 100590.
[4] U. Schwiegelshohn, R. Yahyapour, Analysis of first-come-first-serve parallel job scheduling, in: Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, 1998, pp. 629–638.
[5] D. Carastan-Santos, R.Y. de Camargo, Obtaining dynamic scheduling policies with simulation and machine learning, in: SC17: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM/IEEE, 2017, pp. 1–13.
[6] A. Legrand, D. Trystram, S. Zrigui, Adapting batch scheduling to workload characteristics: What can we expect from online learning? in: 2019 IEEE International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2019, pp. 686–695.
[7] A.W. Mu'alem, D.G. Feitelson, Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling, IEEE Trans. Parallel Distrib. Syst. 12 (6) (2001) 529–543.
[8] C. Galleguillos, Z. Kiziltan, A. Sîrbu, O. Babaoglu, Constraint programming-based job dispatching for modern HPC applications, in: Principles and Practice of Constraint Programming: 25th International Conference, Springer, 2019, pp. 438–455.

[9] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Scheduling-based power capping in high performance computing systems, Sustain. Comput. Inf. Syst. 19 (2018) 1–13.
[10] Y. Kessaci, N. Melab, E.G. Talbi, A Pareto-based metaheuristic for scheduling HPC applications on a geographically distributed cloud federation, Clust. Comput. 16 (2016) 451–468.
[11] H. Hafsi, H. Gharsellaoui, S. Bouamama, Genetically-modified multi-objective particle swarm optimization approach for high-performance computing workflow scheduling, Appl. Soft Comput. 122 (2022) 108791.
[12] D. Zhang, D. Dai, Y. He, F.S. Bao, B. Xie, Rlscheduler: An automated HPC batch job scheduler using reinforcement learning, in: SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, ACM/IEEE, 2020, pp. 1–15.
[13] J. Fomperosa, M. Ibañez, E. Stafford, J.L. Bosque, Task scheduler for heterogeneous data centres based on deep reinforcement learning, in: Parallel Processing and Applied Mathematics: 14th International Conference, Springer, 2023, pp. 237–248.
[14] J. Li, X. Zhang, Z. Wei, J. Wei, Z. Ji, Energy-aware task scheduling optimization with deep reinforcement learning for large-scale heterogeneous systems, CCF Trans. High Perform. Comput. 3 (4) (2021) 383–392.
[15] J. Li, X. Zhang, J. Wei, Z. Ji, Z. Wei, GARLSched: Generative adversarial deep reinforcement learning task scheduling optimization for large-scale high performance computing systems, Future Gener. Comput. Syst. 135 (2022) 259–269.
[16] Y. Fan, et al., DRAS: Deep reinforcement learning for cluster scheduling in high performance computing, IEEE Trans. Parallel Distrib. Syst. 33 (12) (2022) 4903–4917.
[17] R.L. de Freitas Cunha, L. Chaimowicz, An SMDP approach for reinforcement learning in HPC cluster schedulers, Future Gener. Comput. Syst. 139 (2023) 239–252.
[18] D. Aikema, C. Kiddle, R. Simmonds, Energy-cost-aware scheduling of HPC workloads, in: 2011 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, IEEE, 2011, pp. 1–7.
[19] A. Kassab, J.-M. Nicod, L. Philippe, V. Rehn-Sonigo, Green power constrained scheduling for sequential independent tasks on identical parallel machines, in: 2019 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking, ISPA/BDCloud/SocialCom/SustainCom, IEEE, 2019, pp. 132–139.
[20] E. Gaussier, D. Glesser, V. Reis, D. Trystram, Improving backfilling by using machine learning to predict running times, in: SC15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, ACM/IEEE, 2015, pp. 1–10.
[21] J. Lelong, V. Reis, D. Trystram, Tuning easy-backfilling queues, in: 21st Workshop on Job Scheduling Strategies for Parallel Processing, Springer, 2018, pp. 43–61.
[22] C. Gómez-Martín, M.A. Vega-Rodríguez, J.-L. González-Sánchez, Fattened backfilling: An improved strategy for job scheduling in parallel systems, J. Parallel Distrib. Comput. 97 (2016) 69–77.
[23] E. Kolker-Hicks, D. Zhang, D. Dai, A reinforcement learning based backfilling strategy for HPC batch jobs, in: Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, ACM, 2023, pp. 1316–1323.
[24] A. Sîrbu, O. Babaoglu, A data-driven approach to modeling power consumption for a hybrid supercomputer, Concurr. Comput. Pr. Exper. 30 (2018).
[25] A. Borghesi, A. Bartolini, M. Lombardi, M. Milano, L. Benini, Predictive modeling for job power consumption in HPC systems, in: High Performance Computing: 31st International Conference, Springer, 2016, pp. 181–199.
[26] F. Antici, K. Yamamoto, J. Domke, Z. Kiziltan, Augmenting ML-based predictive modelling with NLP to forecast a job's power consumption, in: Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, IEEE, 2023, pp. 1820–1830.
[27] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. Klimov, Proximal policy optimization algorithms, 2017, arXiv preprint arXiv:1707.06347.
[28] P.C. Luo, H.Q. Xiong, B.W. Zhang, J.Y. Peng, Z.F. Xiong, Multi-resource constrained dynamic workshop scheduling based on proximal policy optimisation, Int. J. Prod. Res. 60 (19) (2022) 5937–5955.
[29] X. Song, Y. Jin, G. Slabaugh, S. Lucas, Joint action loss for proximal policy optimization, 2023, arXiv preprint arXiv:2301.10919.
[30] J. Schulman, P. Moritz, S. Levine, M. Jordan, P. Abbeel, High-dimensional continuous control using generalized advantage estimation, 2018, arXiv preprint arXiv:1506.02438.
[31] M. D'Amico, J.C. Gonzalez, Energy hardware and workload aware job scheduling towards interconnected HPC environments, IEEE Trans. Parallel Distrib. Syst. (2021).
[32] U. Lublin, D.G. Feitelson, The workload on parallel supercomputers: modeling the characteristics of rigid jobs, J. Parallel Distrib. Comput. 63 (11) (2003) 1105–1122.
[33] W. Cirne, F. Berman, A comprehensive model of the supercomputer workload, in: Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538), IEEE, 2001, pp. 140–148.
[34] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, J. Riordan, Modeling of workload in MPPs, in: Job Scheduling Strategies for Parallel Processing, Springer, 1997, pp. 95–116.
[35] H. He, H. Shen, Q. Hao, H. Tian, Online delay-guaranteed workload scheduling to minimize power cost in cloud data centers using renewable energy, J. Parallel Distrib. Comput. 159 (2022) 51–64.
[36] M. Sengupta, Y. Xie, A. Lopez, A. Habte, G. Maclaurin, J. Shelby, The national solar radiation data base (NSRDB), Renew. Sustain. Energy Rev. 89 (2018) 51–60.
[37] National Renewable Energy Laboratory, National renewable energy laboratory: WRDB: Wind resource database, 2024, Available: https://wrdb.nrel.gov/ (Accessed 17 2024).
[38] D.G. Feitelson, D. Tsafrir, D. Krakov, Experience with using the parallel workloads archive, J. Parallel Distrib. Comput. 74 (10) (2014) 2967–2982.

Rui Chen received the BEng degree in computer science from the South China University of Technology, Guangzhou, China in 2022. He is currently a master student in computer science at South China University of Technology. His research interests include cloud computing and big data.

Weiwei Lin received his B.S. and M.S. degrees from Nanchang University in 2001 and 2004, respectively, and his Ph.D. in Computer Application from South China University of Technology in 2007. He has been a visiting scholar at Clemson University from 2016 to 2017. Currently, he is a professor in the School of Computer Science and Engineering at South China University of Technology. His research interests include distributed systems, cloud computing, and AI application technologies. He has published more than 150 papers in refereed journals and conference proceedings. He has been a reviewer for many international journals, including IEEE TPDS, TSC, TCC, TC, TCYB, etc. He is a distinguished member of CCF and a senior member of the IEEE.

Huikang Huang received the BEng degree in computer science and technology from the Hanshan Normal University, Chaozhou, China, in 2018, and the MEng degree in computer science and theory from the South China Agricultural University, Guangzhou, China, in 2021. Now, he is a Ph.D. candidate at the School of Computer Science and Engineering, South China University of Technology. His research interests mainly focus on cloud computing.

Xiaoying Ye received the B.S. degree in Computer Science from the University of Electronic Science and Technology of China in 2006 and the M.S. degree in Business Administration from Guilin University of Technology in 2014. She is currently a Professor at the University of Electronic Science and Technology of China. Her research interests mainly include big data and software engineering.

Zhiping Peng received the B.S. degree from the China University of Petroleum, Huadong, in 1996, the M.S. degree from the Huazhong University of Science and Technology, in 2001, and the Ph.D. degree in computer applications from the South China University of Technology, in 2007. He is currently a Professor at the Guangdong University of Petrochemical Technology. He has published more than 50 articles in refereed journals and conference proceedings. His research interests include cloud computing, machine learning, and multi-agent systems.