The Journal of Supercomputing (2025) 81:427
https://doi.org/10.1007/s11227-024-06907-y

Intelligent energy pairing scheduler (InEPS) for heterogeneous HPC clusters

Marta López[1] · Esteban Stafford[1] · Jose Luis Bosque[1]

Accepted: 27 December 2024

Abstract

In recent years, energy consumption has become a limiting factor in the evolution of high-performance computing (HPC) clusters in terms of environmental concern and maintenance cost. The computing power of these clusters is increasing, together with the demands of the workloads they execute. A key component in HPC systems is the workload manager, whose operation has a substantial impact on the performance and energy consumption of the clusters. Recent research has employed machine learning techniques to optimise the operation of this component. However, these attempts have focused on homogeneous clusters where all the cores are pooled together and considered equal, disregarding the fact that they are contained in nodes and that they can have different performances. This work presents an intelligent job scheduler based on deep reinforcement learning that focuses on reducing energy consumption of heterogeneous HPC clusters. To this aim it leverages information provided by the users as well as the power consumption specifications of the compute resources of the cluster. The scheduler is evaluated against a set of heuristic algorithms showing that it has potential to give similar results, even in the face of the extra complexity of the heterogeneous cluster.

Keywords Task scheduling · Deep reinforcement learning · High-performance computing · Heterogeneous clusters · Energy consumption

Marta López, Esteban Stafford and Jose Luis Bosque have contributed equally to this work.

✉ Esteban Stafford
esteban.stafford@unican.es

Marta López
marta.lopez@alumnos.unican.es

Jose Luis Bosque
joseluis.bosque@unican.es

[1] Department of Computer Engineering and Electronics, Universidad de Cantabria, Avda. los castros, s/n, 39005 Santander, Spain

Published online: 24 January 2025

1 Introduction

High-performance computing (HPC) systems are constantly evolving and reaching greater heights in computing power, efficiency and scalability [1]. Along the way, energy consumption has always posed a significant obstacle in terms of environmental impact [2] and operational cost. The costs of running an HPC cluster for two years can compare to the cost of purchase [3]. Currently, data centre and data transmission networks account for roughly 1.5% of global electricity use and 0.9% of energy-related greenhouse gas emissions [4]. This represents only a moderate increase with respect to 2010 thanks to improvements made in energy efficiency, but CO_2 emissions must still be halved by 2030 to achieve the United Nations net zero objective.

In the context of high performance computing, *job scheduling* is a key aspect of cluster operation that deeply influences its energy consumption and efficiency. This task, which is carried out by a piece of software usually called *Workload Manager* or *scheduler*, determines the manner in which free resources are assigned to incoming jobs. The workload manager may seek different objectives through its scheduling policy, such as minimising response time or energy consumption. To achieve this, it must take several considerations into account, like job requirements and cluster properties. Optimal job scheduling is an NP-complete problem [5], meaning that there is no known algorithm that can solve it in polynomial time. For this reason, several methods have been proposed that can provide approximate solutions, initially heuristic algorithms and, more recently, artificial intelligence approaches [6–10]. The scheduling problem becomes more complex when considering that the compute resources, the cores, are grouped into nodes. And it is even more challenging when these nodes have different characteristics, like in heterogeneous clusters. In these cases, assigning a job to one resource or another produces different results depending on their hardware specifications. This work focuses on heterogeneous clusters, that have nodes with different capabilities, not on clusters of heterogeneous nodes, that contain nodes with compute accelerators.

Machine learning techniques aim to train a system, or *model*, to reproduce behaviours or make predictions in the context of a specific problem [11]. Recent advances have led to *deep learning*, which employs artificial neural networks with multiple layers to progressively extract features from an input at multiple levels. One of these techniques, called reinforcement learning, can be applied to the scheduling problem. Contrary to heuristic approaches, reinforcement learning does not require a prior knowledge of the problem, but rather it learns from experience. A deep reinforcement learning (DRL) scheduler can appreciate the complex behaviour of a cluster and learn to improve some metric, like execution time or energy consumption [12–14]. However, these approaches have been developed for simplistic homogeneous clusters, where all the cores are equal and pooled into one large group, eluding the fact that they are contained in nodes and the impact this has on parallel applications. This simplifies the scheduling problem into one of choosing the next job. However, in heterogeneous clusters, allocating a job

to a faster node or a slower one has a significant impact in the performance and energy consumption. Therefore, the scheduler must consider the characteristics of the nodes and the jobs to make a decision.

Many approaches have been tested to reduce the energy consumption in the HPC field, some of which focus on making a better use of resources through scheduling techniques [15–17]. However, these contributions use exclusively time-related parameters to guide the scheduler into improving energy consumption. It is worth considering that energy consumption is not only related to the time a job takes to execute, but also which node it is executed on. For instance, a job that is executed in a node with a higher performance might consume less energy than the same job executed in a slower node, even if it takes longer to complete. This is because most performant node can complete the job faster and enter a low power state, while the slower node will be consuming energy for a longer time.

This work follows the above lines of thought by proposing an intelligent energy-aware workload manager based on deep reinforcement learning, with the objective of either reducing the total energy consumption or the energy-delay product (EDP), for heterogeneous HPC cluster architectures. The novelty of this approach is the addition of an energy consumption estimation of the jobs to the neural network, and considering the heterogeneous nature of the cluster, where varying properties of nodes lead to different energy consumption. Through a set of experiments the scheduler is evaluated to determine if it is capable of learning the intricacies of scheduling in these conditions to reduce energy consumption.

The main contributions of this work are:

- Proposal of a workload manager based on a DLR agent for heterogeneous clusters that improves the energy consumption or efficiency by considering energy estimates of jobs and nodes.
- Implementation and public release of the proposed scheduler in the IRMaSim open-source cluster simulator [18].
- Experimental evaluation of the scheduler, comparing its behaviour with other well-known heuristic scheduling algorithms.

As far as the authors are aware, this is the first instance of a DRL scheduler being proposed for heterogeneous clusters with a focus on energy consumption. This proposal paves the way to a new generation of workload managers that can not only adapt to job properties, but also considering node characteristics to make better decisions. This will lead to intelligent schedulers that can reduce energy consumption in heterogeneous clusters, which is a key factor for the evolution of HPC systems.

The remainder of the paper is organised as follows. Section 2 gives some background on machine learning concepts employed in the article. Section 3 presents the intelligent energy-aware workload manager. Section 4 establishes the experimental setup used in the validation carried out in Sect. 5. A literary review appears in Sect. 6 and finally Sect. 7 summarises the findings of this article.

2 Background

This section delves into the basic ideas and theory behind reinforcement learning. To this aim, it introduces the main concepts of reinforcement learning that revolve around the agent-environment interaction, and the neural networks that are used to approximate the behaviour of the agent.

2.1 Reinforcement learning

Reinforcement learning (RL) is a strategy that involves two main elements: an *agent* and an *environment*. The first is the entity that learns to improve some metric by interacting with the second, which is the system that the agent is trying to control. The goal of RL is to teach the agent what *action* to take after having observed a particular *state* in the environment, so as to optimise a given objective. The complexity of this model resides in the fact that each action taken by the agent modifies the environment and therefore influences future behaviour [19].

More specifically, the agent learns a *policy* that maps states to actions. This policy may be deterministic, always choosing the same action for a given state; or stochastic, assigning a probability to each possible action based on the current state. Neural networks are often used to approximate the policy in the latter case. Every time the agent chooses an action and applies it to the environment, the state of the environment changes and a *reward* is generated indicating how beneficial that particular action was towards the end goal. These rewards are employed by the agent to gradually refine the policy through successive iterations, as shown in Fig. 1.

In the case at hand, the agent-environment interaction happens in discrete time steps $t = 0, 1, 2, \ldots$ in each of which the following occurs. The agent receives an observation of the state of the environment $s_t \in \mathcal{S}$. Based this state observation, the agent selects an action $a_t \in \mathcal{A}$. As a consequence of applying the chosen action, the agent receives a reward $r_{t+1} \in \mathcal{R}$ and the environment advances to state s_{t+1}. Note that s_t and r_t are random variables. The final goal of the agent is to maximise the *expected return* at each time step. The return G_t is a function of the reward sequence given by $G_t \doteq r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$, where T is a final time step.

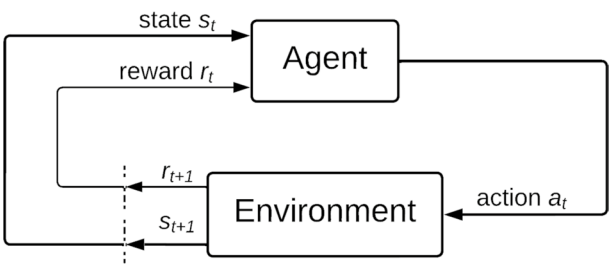

Fig. 1 Agent-environment interaction in reinforcement learning

2.2 Neural networks

Neural networks can be considered universal function approximators, which makes them a powerful tool for pattern recognition, classification and regression tasks. They comprise a series of *layers* of interconnected artificial *neurons*, a structure inspired by brain anatomy. Feed-forward neural networks, also known as multilayer perceptrons (MLPs), are the simplest kind of neural network models.

Each of the m neurons of a layer generates a linear combination, called an *activation*, of the n input variables $\mathbf{x}$ according to its parameters, called weights $\mathbf{w}$ and biases $\mathbf{b}$. For a neuron j, the activation a_j is calculated as follows:

$$a_j = \sum_{i=1}^{n} w_{j,i} x_i + b_j, \quad 1 \leq j \leq m$$

where i would be the index of the input variable. Then, a differentiable, nonlinear *activation function* h is applied to the *activation* to give the final output of each neuron as $z_j = h(a_j)$. These outputs can then be used as inputs to a following layer of neurons defined in the same manner [11].

Neural networks have to be trained in order to approximate a given function. This process involves comparing the output of the network for a series of input vectors $\mathbf{x}$ with the corresponding known correct values and adjusting the weights and biases of the network in order to minimise the error E_n. This is usually done by making use of the information provided by the gradient of the error function, often called *loss* or *objective* function. This gradient can be computed efficiently through *backpropagation*, which involves the following steps [11]:

1. *Forward propagation*: an input vector $\mathbf{x}_n$ is propagated through the network to obtain the activations of all the hidden and output neurons, referred to as a_j and a_k, respectively, from here on.
2. The errors $\delta_k \equiv \frac{\partial E_n}{\partial a_k}$ are computed for all the output units.
3. *Backpropagation*: the errors of the output layer are backpropagated through the network to obtain the corresponding δ_j error terms for each hidden unit using the chain rule for partial derivatives: $\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$.
4. The required derivatives are evaluated using $\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$.

3 Intelligent energy pairing scheduler (InEPS)

Developing a DRL scheduler requires the definition of the environment, which is the interface between the agent and the cluster, and the agent itself, which is the neural network that will make the scheduling decisions. This section describes the main components of the InEPS scheduler, including the environment, the agent, and the training process.

3.1 Overview

Applying machine learning to optimise time-related parameters in HPC cluster scheduling is not a new idea [9]. However, the authors believe that these techniques can be leveraged to produce a scheduler for heterogeneous clusters, which can reduce energy consumption metrics. For this purpose, the scheduler takes into account an estimation of the energy consumption of jobs in the different nodes of the cluster. Therefore, a simple energy model is proposed, based on the requested time and cores of the jobs and the static and dynamic powers of the nodes. This novel Intelligent Energy Pairing Scheduler (InEPS) incorporates both the static and dynamic power specifications of each node into its observations. Additionally, it allows users to specify maximum energy constraints for jobs.

This scheduler is based upon the idea of pairing jobs to the compute nodes of the cluster, which was also used to develop a scheduler aimed at reducing time-related metrics in heterogeneous clusters [20]. In both cases, the core idea is assigning a score to each possible job-node pair, and then selecting a pair based on these scores. To limit the complexity of the neural network, the jobs can be parallel, but the processes or threads they might spawn must execute in a single node. Thus, the observations of the platform are divided in nodes, concealing their internal architecture, but including its impact.

If in this previous work time-related metrics were considered for optimisation, InEPS is exclusively focused on improving energy consumption or EDP. This is achieved by modifying the features that compose the observation, for instance, by adding an estimate of energy consumption of job-node pairs. The neural network is also adapted to this new observation space by changing its internal architecture. Since the objective is to reduce energy consumption or EDP, two new reward functions have been defined.

3.2 Main components

The InEPS scheduler can be described through two main aspects: the environment, which describes what information flows it receives and generates, and the agent, which is the machine learning core that will make scheduling decisions.

These components are interrelated as shown in Fig. 2. It can be seen that the environment acts as an interface between the agent and the heterogeneous cluster and job queue.

The training of the agent is performed with the IRMaSim simulator, which presents it with a set of jobs and nodes [18]. Before proceeding on to the specification of each of the components, it is necessary to consider how the training sequence progresses. In addition to a description of a computer cluster, each training process is characterised by a *workload*, which is a list of job descriptions. This is a list of job descriptions logged at a real computer cluster. Job descriptions include arrival times and user-specified parameters, such as requested time or number of cores. The training is divided into a number of *trajectories*, which

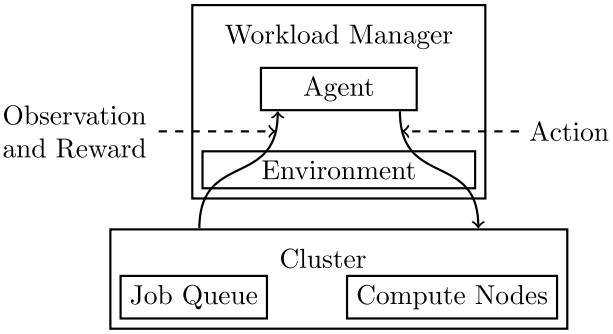

Fig. 2 Main components and their relations in InEPS

are fixed-length sequences of jobs taken at random from the workload. Processing a trajectory involves analysing the impact of many *events*, like job submissions or completions. The agent evaluates the situation for each event and determine the appropriate next action. This process is illustrated in Fig. 3.

3.3 Environment

The environment is mainly responsible for providing the agent with the necessary information to make a decision, and updating the cluster state after the decision of the agent. The observation gives a view of the state of the cluster and the job queue, whereas the action encapsulates the decision that the scheduler must carry out. This section describes observation and action spaces which define the conveyed values, together with the reward functions and how the scheduler estimates energy consumption.

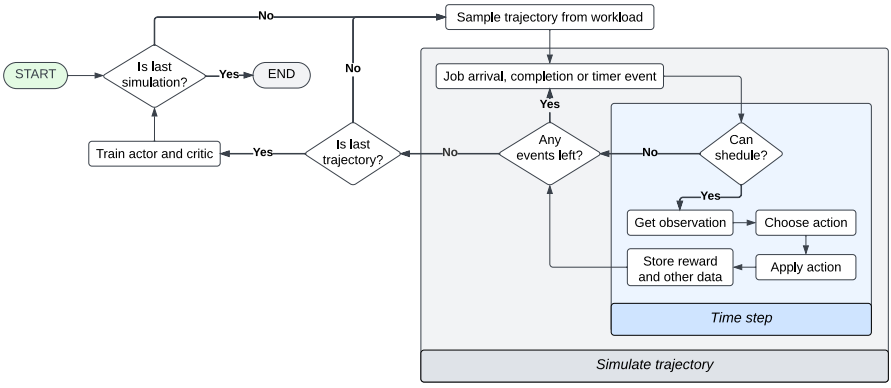

Fig. 3 Flowchart depicting a high-level view of the main simulation loop using a reinforcement learning agent in IRMaSim

3.3.1 Observation space

Observations are created by recording features for each jobs-nodes pairing, as defined in [20]. It must contain a fixed number of values or features. This is a requirement of the neural network that will be used by the agent. Since the job queue does not meet this condition, a *eligible job vector* is defined. This is a list of the first `NUM_JOBS` jobs from the job queue, that is padded with null jobs if necessary.

The features of the observations are values bounded between 0.0 and 1.0 organised in an array of dimensions (`NUM_NODES` × `NUM_JOBS`, `NUM_FEATURES`). Where each row contains the features that describe a given pairing of node and job. A list of the considered features considered is presented next, with summary in Table 1. The features that were added to those introduced in [20] are highlighted with a *.

- The *submit time** of a job serves as an indicator of its position in the queue.
- The *static power** of a node is the sum of the static power of its cores if it is running at least one task. Otherwise, this value is set to the idle power of the node, which is a percentage of its static power.
- The *dynamic power** of a node is the sum of the dynamic powers of the cores in said node that are currently running a task.
- The *availability* is defined as the ratio of free cores to total cores in the node and is meant to provide useful information in the presence of memory contention.
- The *energy estimate** is the amount of energy that the job could consume if it is scheduled in the node. This is obtained from the model described later in this section.
- The *can schedule* feature tells if the node has enough free cores to fit the job, and the energy estimation is below the maximum energy constraint of the job.

3.3.2 Action space

An action identifies a job and a node, directing the scheduler to allocate the node to the job. Thus, the action space is defined as a discrete space of size (`NUM_NODES` × `NUM_JOBS`), referred to as `ACTIONS_SIZE`. Actions are encoded as indices to the rows in the observation space, mapping nodes to the jobs in the eligible queue.

It is worth pointing out that some job-node pairs are not feasible and should not be chosen by the agent. Specifically, when the job is a null used to pad the

Table 1 Observation features considered for each job-node pair in Energy scheduler

Job	Wait time, Requested time, Submit time*, Requested cores
Node	Availability, Static power*, Dynamic power*, Average clock rate
Common	Energy estimate*, can schedule

eligible job array or when the job does not fit in a node. As neural networks operate probabilistically, invalid pairs must be masked before selecting the action.

3.3.3 Reward functions

The environment also contains the current reward function that the agent will try to optimise. This is a function that takes the state of the cluster and the action taken by the agent as input, and returns a scalar value, that will be used by the agent in the training process. The definition of the reward function will determine which objective the agent will try to minimise.

Only two reward functions are implemented in InEPS, one to reduce energy consumption and the other to improve energy-delay product (EDP) [21]. The first function returns the *negative* increment in energy consumption, and the other is the same but multiplied by the time increment. This is so because the agent is programmed to *minimise* the sum of the reward values. The increments are obtained using marker variables, which are updated each time a reward is generated. It should be noted that the sum of the EDP rewards throughout a trajectory is not equivalent to the EDP of executing its jobs, since this value is impossible to calculate beforehand. Instead, these incremental rewards are exclusively envisioned to offer guidance during training.

3.3.4 Energy estimation

One observation passed to the agent is the estimated energy consumption for scheduling a job on a node. This is computed as the product of the estimated *runtime* and *power consumption* when the job is run on the node, taking into account the current status of the cluster. The resulting formula can be expressed as follows:

$$t_{req} \cdot \frac{f_{min}}{f} \cdot \left(\frac{p_s}{j+1} + n \cdot p_d \right) \tag{1}$$

In equation 1, $t_{req} \cdot \frac{f_{min}}{f}$ is the *expected runtime*. The t_{req} user-specified job parameter indicates the expected runtime of executing the job on the *slowest node* of the cluster, running at a frequency f_{min}, so this value has to be adjusted to the frequency of the target node, f.

The parenthesised term corresponds to the *power estimate* for the job-node pair. p_s is the static power consumption of the target node, whereas p_d represents the dynamic power consumption of a single core. The static power is distributed among the j jobs that are currently running on the node, plus one for the target job. This gives the fraction of the static power consumed by the incoming job. The dynamic power estimate is obtained by adding the dynamic powers of the n cores needed to allocate the n tasks of the target job.

3.4 Agent

The agent implements the PPO algorithm [22] in an actor-critic paradigm and therefore contains three key components: two deep neural networks, called an actor and a critic, and a buffer used to store the collected training data. In addition, the agent contains two Adam optimisers [23] to update the parameters of both networks. The initial value of the network weights and optimisers is set randomly.

Stochastic weight averaging (SWA) [24] is applied exclusively to the actor network, starting after 75% of the training epochs. The learning rate decreases linearly over 15% of the remaining epochs before stabilising. This addition has only been implemented for the actor, since it is directly responsible for scheduling.

During training, the agent receives an observation of the state of the environment and produces the action predicted by the actor, the value predicted by the critic for the observed state and the log probability of the chosen action in the probability distribution over actions output by the actor.

Both networks are trained for a number of epochs with independent loss functions. The loss function corresponding to the actor includes a term corresponding to the entropy of the policy, which may be set to a nonzero value to encourage exploration [25]. In each epoch, the collected training data is sampled in minibatches of a fixed size and used to obtain the gradients of the loss functions and apply a step of the optimisers to the parameters of the networks.

3.4.1 Actor

The actor network is a deep neural network with six hidden layers, starting with an input size of `NUM_FEATURES` and decreasing dimensions of 16, 16, 8, 8, 4, and 4, before producing a single output. The initial weights are normalised to zero mean and unit variance and the Scaled Exponential Linear Unit (SELU) is used as the activation function between layers, making the actor a self-normalising neural network. It is defined as follows:

$$\mathrm{SELU}(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

After forwarding the observation through the network and flattening it to obtain a tensor of dimension (`ACTIONS_SIZE`), a mask is applied to artificially lower the scores of invalid job-node combinations, preventing the model from selecting them in most scenarios. The masked output is then used to initialise a categorical distribution from which an action is sampled. The actor computes the Proximal Policy Optimisation (PPO) loss function for a minibatch of collected data, optimising the policy to improve performance in subsequent actions.

3.4.2 Critic

The critic neural network contains a total of ten hidden layers divided into two phases. The input layer is identical to that of the actor network. The following five hidden layers have decreasing sizes 32, 16, 8, 4 and 1. At this point, the result is squeezed to a shape of (`ACTIONS_SIZE`) for a single observation. The five following hidden layers further reduce the dimensionality from `ACTIONS_SIZE` to 1 in the sequence 128, 64, 32, 16, 8 and 1 for the output layer, resulting in a single numerical value. SELU is also used as the activation function in this network, and the weights are initialised with zero mean and unit variance as well.

The loss function is the standard mean squared error (MSE) comparing the prediction supplied by the network to the expected return values collected during training, which is defined as follows

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where n is the number of observations, y_i are the actual values, and $\hat{y}_i$ are the predicted values.

3.4.3 Buffer

The agent contains a storage area to collect values after each time step, necessary for computing the losses and training the actor and critic. The values include observations, actions, rewards, values, action log probabilities, advantages and expected returns. The first five are stored after each time step, whereas the last two are computed after each complete trajectory as discounted sums from the collected data, using generalised advantage estimate (GAE) in the case of the advantages.

4 Methodology

The evaluation process has been carried out on the IRMaSim cluster simulation framework, which makes training possible without the need to deploy the new scheduler in a real HPC environment [18]. InEPS can be configured to optimise a single metric, that can be either energy consumption or EDP.

For comparison, a set of heuristic algorithms is used. Since the proposed scheduler targets heterogeneous clusters, these algorithms must be adapted to this kind of platform. Thus, they are based on two sorting criteria, one for the jobs and one for the nodes. The operation of these heuristic schedulers is as follows. Every time there are free resources in the cluster, a list of jobs that could fit in the available nodes is generated from the job queue. This list is sorted according to the job selection policy. For instance, under the *shortest* policy, jobs are sorted by requested time, with the shortest job first. Next, the available nodes are sorted according to the node selection policy. If the policy is *high_gflops*, the nodes are sorted by the peak

Table 2 Job and node sorting criteria for heuristic schedulers

Job selection policies	
Random	Any job
First	Oldest job in the job queue
Shortest	Job with the lowest expected execution time
Smallest	Job with the least requested cores
Low_mem	Job with the lowest requested memory
Low_mem_ops	Job with the lowest memory access volume
Node selection policies	
Random	Any node
High_gflops	Node with the highest peak compute capability
High_cores	Node with the most currently available cores
High_mem	Node with the highest currently available memory
High_mem_bw	Node with the highest currently available memory bandwidth
Low_power	Node with the lowest power consumption

compute capability, with the most powerful node being the first in the list. Then the first job in the eligible job queue is allocated to the first node in the available node list. This process is repeated until there are no more jobs to allocate. The available policies are given in Table 2.

Although all the heuristics are used in the experiments, the graphs only include the one with the best performance and the following four, which are considered relevant: *first-high_gflops*, *first-high_cores*, *first-high_mem_bw*, and *random-random*. The first three focus on node selection rather than job selection, while the last is included as a worst-case scenario.

5 Experimental results

This section evaluates the InEPS scheduler in two different scenarios, the first validates the convergence and the behaviour of the proposal, while the other tests how the scheduler copes with situations different from those used during training. Also, depending on the target metric there are two alternative models of the scheduler: Energy InEPS tries to minimise energy consumption, whereas EDP InEPS strives to improve energy efficiency.

5.1 Convergence analysis experiment

This experiment has been designed to analyse the convergence and behaviour of both versions of InEPS. The details of the used platform and workload are shown in Tables 3 and 4, with a total of 1040 cores and 180 jobs.

Both models have been trained with the hyperparameters shown in Table 5. In each simulation, five random sets of up to ninety jobs were selected from the

Table 3 Platform details for convergence analysis experiment

Node type	0	1	2	3
Number of Nodes	10	10	10	10
Cores	8	16	32	48
Clock rate (GHz)	4.2	3.8	3.4	3.0
Static power (W)	68.81	56.33	45.09	35.11
Dynamic power (W)	6.49	5.32	4.26	3.31
Min. power (%)	39.59	39.59	39.59	39.59

Table 4 Workload details for convergence analysis experiment

Profile	A	B	C
Number of Jobs	10	10	10
Req. cores	8	8	4
Req. ops ($\times 10^{10}$)	1.375	6.25	12.5
Req. time (s)	4.6	20.8	41.7
Mem. vol. (MB/s)	1	10^3	10^6

Table 5 Training hyperparameters for convergence analysis experiment

Simulations	100
Trajectories	5
Trajectory length	90
Minibatch size	32
Epochs	50
SWA	Yes
Seed	1024
ϵ	0.1
γ	0.99
λ	0.95
Actor lr	0.001
Critic lr	0.001

workload and scheduled with the same policy. The graphs shown in Fig. 4 represent the evolution of the loss and reward functions throughout the training of Energy InEPS.

Despite the spikiness of the policy and value loss plots, it can be seen that they tend to zero, which leads to the conclusion that the model is converging. The spikes can be attributed to the change in the features of the jobs and the usage of the cluster during the different trajectories used for training. Simulating shorter trajectories reduces training time, but introduces variance because the sampled workload fragments may not be well-balanced. The dips in the reward can be explained by the fact that only fragments of the complete workload are being scheduled in each simulation, and these may have contained heavy jobs.

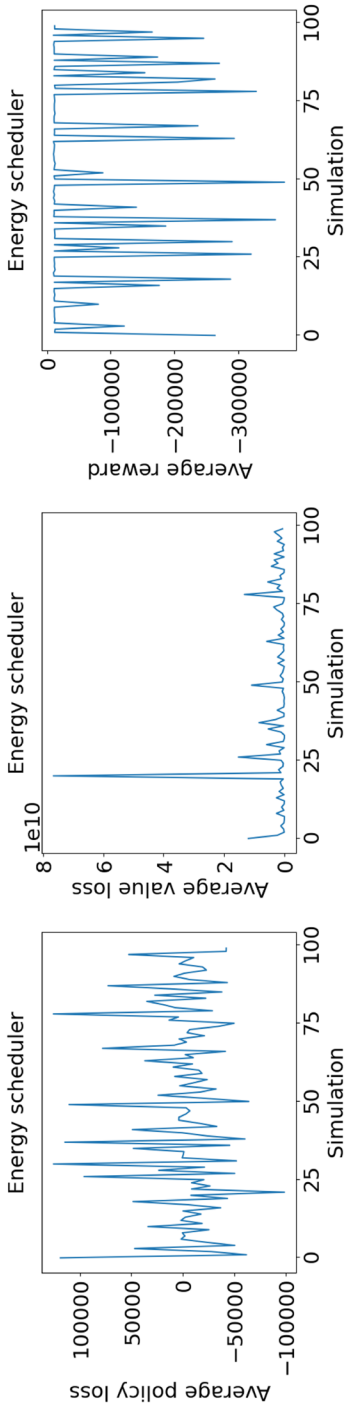

Fig. 4 Convergence Analysis Experiment - Loss and reward plots for Energy InEPS

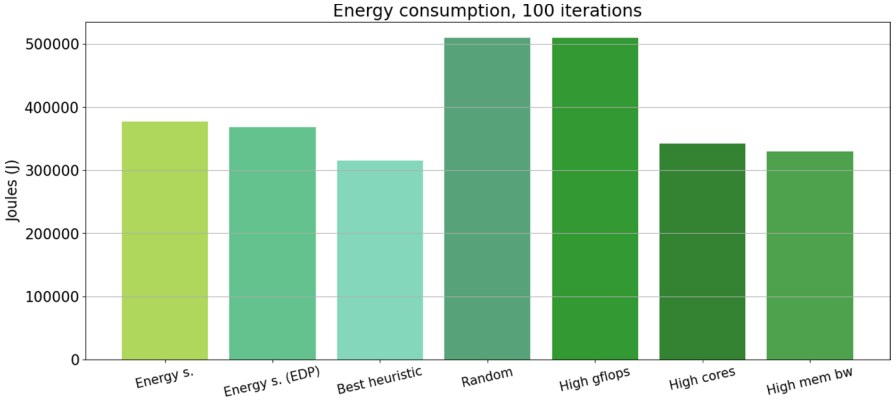

Fig. 5 Convergence Analysis Experiment - Energy consumption comparison

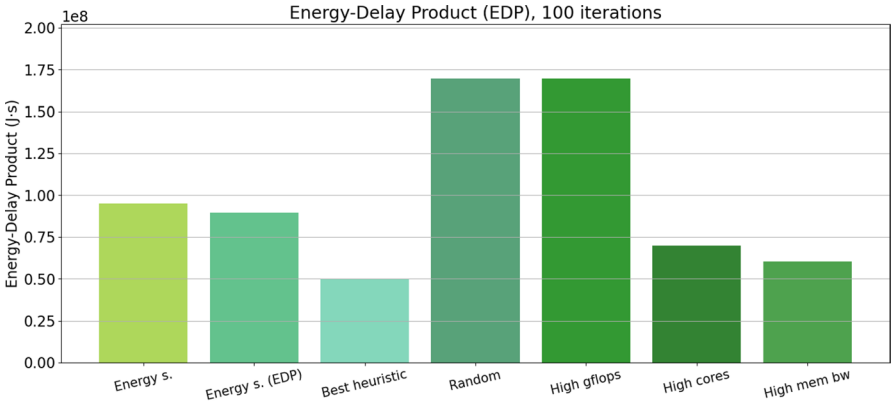

Fig. 6 Convergence analysis experiment—EDP comparison

The loss and reward data of the EDP InEPS training closely resemble those of Energy InEPS as shown in Fig. 4.

Once both models have been trained, they are evaluated by applying them to the complete workload. The results given by the models and the heuristic schedulers for energy consumption and efficiency are shown in Figs. 5 and 6, respectively. The best heuristic in this experiment is *smallest - high_mem*.

Considering the random heuristic as the baseline, the best heuristic reduces the energy consumption by 38% and the EDP by 70.6%. The results of the models vary depending on the simulation, therefore the corresponding bars show the average consumption and EDP. The models obtained for both objectives behave in a similar manner regardless of the evaluated metric. In both metrics, the models have results close to the best heuristics. The Energy InEPS model reduces the energy in 26% and the EDP in 44% on average, and the EDP InEPS model shows average reductions of 28% and 47% in energy consumption and EDP, respectively. It is worth pointing out

that the EDP InEPS gives slightly better results than the Energy InEPS. The slight advantage of the former cannot be clearly attributed to the chosen objective, though it is be a reasonable hypothesis, as execution time and energy consumption appear correlated in this problem.

The above results show that the proposed scheduler is capable of giving fair results in some instances, close to the best heuristics, but does not give consistent results. Three out of five heuristics have beaten the Energy InEPS model. It can be said that the new scheduler has been capable of offering reasonable performance, but adjustments to the training hyperparameters or the actor and critic models should be made to achieve more stable convergence.

5.2 Workload adaptability experiment

This second scenario has been designed with the objective of determining how well the InEPS scheduler is capable of adapting to different workloads. To this end, the scheduler has been trained on a generic workload and then tested on two other workloads, one is memory-intensive and the other is computation-intensive.

To avoid long training times, the size of the platform and the number of jobs have been reduced with respect to the previous experiment. The target platform is composed of ten instances of Node 0 and two instances of Node 3, following the descriptions given in Table 3. In this way, multiple jobs may be assigned to the same 48-core node or single jobs can be allocated to faster 8-core nodes.

Two workloads containing one hundred jobs each have been defined using the job profiles introduced in Table 6: a memory-intensive workload using A1, B1 and C1; and a computation-intensive workload using A2, B2 and C2. In each workload, jobs arrive in one-second intervals in a random order. A separate mixed workload of four hundred jobs combining all six profiles has been used to train the scheduler, mostly using the hyperparameters from the Convergence Analysis Experiment, except for the number of trajectories, the trajectory length and minibatch size, which have been adjusted to 3, 50 and 25, respectively.

The training process of the Energy InEPS model with the mixed workload is illustrated in Fig. 7. Only the critic seems to have converged to an adequate degree, according to the value loss plot. The policy loss and the rewards, on the other hand, show a slight improvement over time, but still oscillate considerably at the end of the training execution.

The results provided by the Energy InEPS model for the memory-intensive and computation-intensive workloads over one hundred test iterations have been

Table 6 Workload details for Workload Adaptability Experiment

Profile	A1	B1	C1	A2	B2	C2
Req. cores	8	8	8	8	8	8
Req. ops ($\times 10^{10}$)	2.75	6.25	12.5	5.5	8.25	12.5
Req. time (s)	9.2	20.8	41.7	18.4	27.5	41.7
Mem. vol. (MB/s)	10^5	10^6	10^8	1	1	1

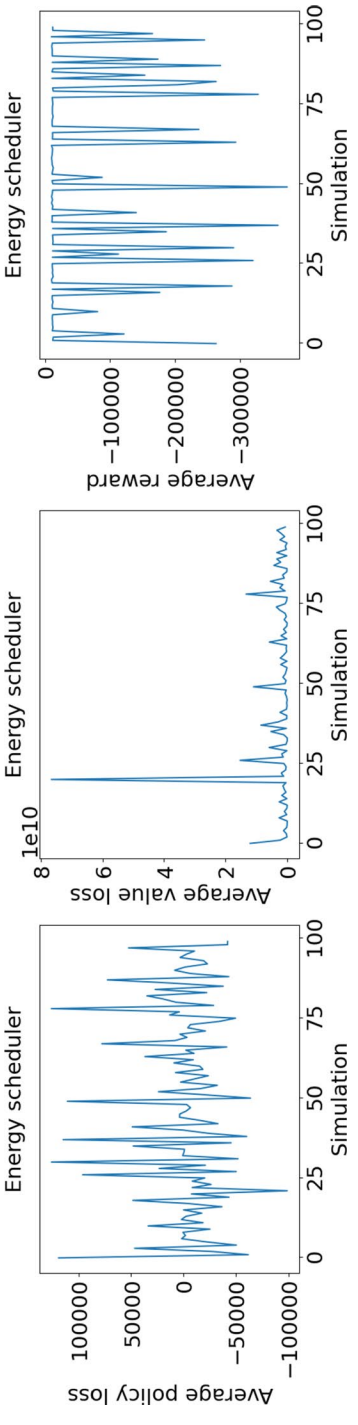

Fig. 7 Workload adaptability experiment—loss and reward for the Energy InEPS

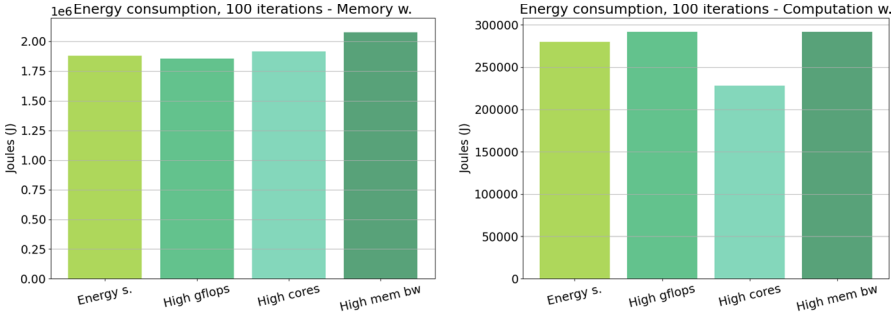

Fig. 8 Workload adaptability experiment—Energy consumption comparison for the memory-intensive (left) and computation-intensive (right) workloads

Table 7 Workload adaptability experiment—Percentage reductions in energy consumption obtained by Energy InEPS with respect to the three tested heuristics for both workloads

Energy cons.	*High_gflops*			*High_cores*			*High_mem_bw*		
reduction (%)	*Min*	*Avg*	*Max*	*Min*	*Avg*	*Max*	*Min*	*Avg*	*Max*
Mem. workload	12.66	−1.31	−12.87	15.32	1.77	−9.43	21.98	9.50	−0.83
Comp. workload	23.36	4.02	−0.35	2.28	−22.37	−27.95	23.36	4.02	−0.35

compared to the energy consumption produced by three of the heuristics. It can be argued that the *first - high_gflops* heuristic gives the best results with the computation-intensive workload, since jobs are sent to the fastest node available. The *first - high_cores* prioritises nodes with high free core counts. Finally, the *first - high_mem_bw* heuristic should be able to reduce energy consumption in the presence of memory contention, assigning jobs to the nodes with the highest available bandwidth. Figure 8 illustrates the differences in energy consumption of the different schedulers for each of the two workloads. Table 7 presents the corresponding reduction percentages.

Unexpectedly, *high_gflops* is the best-performing heuristic for the memory-intensive workload. *High_mem_bw*, on the contrary, has achieved the worst performance in this case. A possible explanation could be that it estimates the currently available bandwidth as the negative sum of the requested bandwidths of the jobs currently running on the target node. *High_cores* offers an intermediate solution. When compared to these three heuristics for the same workload, the Energy InEPS model is closest to *high_gflops*.

As for the computation-intensive workload, in this case *high_gflops* and *high_mem_bw* produce similar energy consumption, and it is *high_cores* that manages to offer a considerable reduction, of around 22%. The Energy InEPS only provides a reduction of 4% on average with respect to the worst heuristic.

In summary, the Energy InEPS model is capable of giving good results in some cases, showing a promising perspective. Furthermore, it should be taken into account that even if the tested model did not converge completely, and was trained

on trajectories of half the test-time length, it still was able to produce reasonable results for both workloads. It is also important to note that its behaviour did not vary as much as that of the *high_gflops* and *high_cores* heuristics across different workloads.

6 Related work

The scheduling problem has been exacerbated in recent years owing to several factors including, but not limited to, the steady rise in the processing power of supercomputers, the emergence of new hardware resources and novel hardware structure and the appearance of increasingly diverse workloads that combine compute-intensive and data-intensive applications [26]. Furthermore, the majority of modern HPC cluster systems display a heterogeneous architecture combining resources with different frequencies, number of processing units, bandwidth or memory capacity [27]. This scenario creates a need to consider a variety of trade-offs in a dynamic environment where new jobs may be received at any given time [28]. As an example, resources with higher clock rates complete jobs faster, but may consume more energy during execution due to their greater power consumption [29].

Traditional scheduling approaches have relied on *heuristic schedulers* that assign priorities to jobs waiting in the queue to the cluster based on their attributes, effectively determining which job shall be scheduled next. These range from simple one-parameter heuristic priority functions such as FCFS (First Come First Served) or SJF (Shortest Job First) to complex metrics combining multiple parameters such as WFP, UNICEF or F1 [30].

In [17], the authors address the challenge of scheduling in heterogeneous computer clusters by proposing heuristic algorithms that prioritise node properties over job properties. Experimental results show that these algorithms outperform traditional heuristic approaches that focus on job properties.

Another frequently employed approach is *backfill scheduling*, which relies on user-defined job runtime estimates to schedule lower-priority jobs, located later in the queue, on already allocated resources if this will not result in an already running job missing its deadline or breaching any other kind of restriction. Backfilling provides good results on production systems, but relies heavily on user-established job time estimates, which usually are not very accurate, strongly impacting on the quality of the scheduler [31].

Machine learning approaches have been applied to job scheduling in HPC clusters before, especially in academia. DeepRM [9] was one of the first initiatives to apply reinforcement learning to the scheduling problem as an alternative to heuristic approaches. It assumes a single large resource pool, abstracting away machine boundaries and resource fragmentation. It can learn to optimise objectives such as average job slowdown or completion time through the application of a standard policy gradient reinforcement learning algorithm. Years later, DeepRM2 and DeepRM_Off [10], continuous online resource scheduling and offline resource scheduling designs, were proposed as improvements upon DeepRM. All of them assume very simplistic homogeneous clusters. Decima [32] employs reinforcement learning

techniques and neural networks to learn workload-specific scheduling algorithms given a high-level objective such as minimising average job completion time. It is capable of scheduling while taking into account job dependencies represented as a DAG.

RLScheduler [12] combines a reinforcement learning resource manager with a simplistic data centre simulator to accelerate the training process. In RLScheduler, an observation represents the state of the environment by means of a vector that contains the attributes of all the eligible jobs. However, RLScheduler only considers homogeneous cluster architectures, so it was used as a starting point in [20], where it was adapted to heterogeneous architectures.

RLSchert [13] is a job scheduler based on deep reinforcement learning and remaining runtime prediction. On the one hand, it estimates the state of a cluster by means of a dynamic job remaining runtime predictor that employs a recurrent neural network to encode time series information. DRAS (Deep Reinforcement Agent for Scheduling) [14] focuses on improving upon heuristic scheduling strategies in HPC environments with dynamic application workloads, preventing resource underutilisation and job starvation. This is achieved through *resource reservation*, combining a reinforcement learning approach with backfilling and employs a hierarchical neural network for decision-making. DRAS does not consider heterogeneous cluster architectures.

Other works that specifically tackle energy consumption through job scheduling, can also be found in the literature. ExpREsS [28] is a scheduler for the Apache Spark processing framework. Its main goal is to orchestrate the execution of multiple big data applications in distributed processing systems while satisfying performance requirements and minimising energy consumption. This is achieved through adaptive dynamic voltage and frequency scaling (DVFS) and accurate models capable of predicting the execution time and power consumption of an application before it starts executing. The HEA-PAS scheduling algorithm [33] minimises the response time of parallel DAG applications subject to energy constraints in heterogeneous HPC systems. In HEA-PAS, the energy that can be allocated by an application is quantified and distributed among its tasks according to their energy demand rate. The energy allocated to each task is used to find the optimal processor and frequency combination to execute the entire application. Finally, the work presented in [16] explores the use of application signatures instead of full dynamic power profiling to obtain application data for use in conjunction with energy-aware task scheduling in data centres, including HPC systems. Signatures can therefore be used to estimate the energy consumed by applications without having to monitor and profile their execution until completion.

7 Conclusion

This work presents a scheduler for heterogeneous HPC systems that focuses on reducing either energy consumption or EDP, depending on how it is configured. It is based on deep reinforcement learning agent, but its novelty lies in that it uses power specifications of the compute resources and jobs, to improve its

decision-making, among other job and node metrics. This implies the definition of observation and action spaces, that take into account the energy consumption of jobs and nodes.

Both the training and validation has been done using the IRMaSim cluster simulation framework using a series of synthetic experiments that explore various aspects of the implementation. The results have been compared to various heuristic schedulers. The observed results show potential regarding the possibilities of energy-focused job scheduling based on deep reinforcement learning.

The design space of these schedulers is vast, as there are many alternatives in the structure of the agents, and a huge amount of parameters that have to be determined to tune its behaviour. Therefore, future work could consider alternative configurations of the agent components as well as the parameters considered in the observation and action spaces.

Author contributions All authors contributed equally to the manuscript.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. This work has been supported by the Spanish Science and Technology Commission under contract PID2022-136454NB-C21, the Ministerio de Ciencia e InnovaciÃ³n; Proyectos de TransiciÃ³n EcolÃ³gica y Digital 2021 under grant TED2021-131176B-I00 and the European HiPEAC Network of Excellence.

Data availability The trace data used in this study is available at https://www.cs.huji.ac.il/labs/parallel/workload/l_kit_fh2/index.html.

Code availability The simulator used in this study, IRMaSim, is available at https://github.com/irmasim/IRMaSim.

Declarations

Conflict of interest The authors declare no competing interests.

Ethical approval Not applicable.

References

1. Bosque JL, Robles OD, Toharia P, Pastor L (2011) Evaluating scalability in heterogeneous systems. J Supercomput 58(3):367–375
2. Uddin M, Rahman AA (2012) Energy efficiency and low carbon enabler green it framework for data centers considering green metrics. Renew Sustain Energy Rev 16(6):4078–4094
3. Witkowski M, Oleksiak A, Piontek T, Weglarz J (2013) Practical power consumption estimation for real life hpc applications. Futur Gener Comput Syst 29(1):208–217

4. Agency IE (2022) Data Centres and Data Transmission Networks
5. Ullman JD (1975) Np-complete scheduling problems. J Comput Syst Sci 10(3):384–393
6. Mu'alem AW, Feitelson DG (2001) Utilization, predictability, workloads, and user runtime estimates in scheduling the ibm sp2 with backfilling. IEEE Trans Parallel Distrib Syst 12(6):529–543
7. Brucker P (2013) Scheduling Algorithms. Springer, Berlin, Heidelberg
8. Sun H, Elghazi R, Gainaru A, Aupy G, Raghavan P (2018) Scheduling parallel tasks under multiple resources: list scheduling vs. pack scheduling. In: 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp 194–203
9. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM Workshop on Hot Topics in Networks. HotNets '16, Association for Computing Machinery, New York, NY, USA, pp 50–56
10. Ye Y, Ren X, Wang J, Xu L, Guo W, Huang W, Tian W (2018) A New Approach for Resource Scheduling with Deep Reinforcement Learning. arXiv: 1806.08122
11. Bishop CM (2006) Pattern Recognition and Machine Learning. Springer, New York
12. Zhang D, Dai D, He Y, Bao FS, Xie B (2020) RLScheduler: An Automated HPC Batch Job Scheduler Using Reinforcement Learning. arXiv: 1910.08925
13. Wang Q, Zhang H, Qu C, Shen Y, Liu X, Li J (2021) Rlschert: An hpc job scheduler using deep reinforcement learning and remaining time prediction. Appl Sci 11(20):9448
14. Fan Y, Li B, Favorite D, Singh N, Childers T, Rich P, Allcock W, Papka ME, Lan Z (2022) Dras: deep reinforcement learning for cluster scheduling in high performance computing. IEEE Trans Parallel Distrib Syst 33(12):4903–4917
15. Maroulis S, Zacheilas N, Kalogeraki V (2019) A holistic energy-efficient real-time scheduler for mixed stream and batch processing workloads. IEEE Trans Parallel Distrib Syst 30(12):2624–2635
16. Salinas-Hilburg JC, Zapater M, Moya JM, Ayala JL (2022) Energy-aware task scheduling in data centers using an application signature. Comput Electr Eng 97:107630
17. Stafford E, Bosque JL (2024) Enhancing heterogeneous cluster efficiency through node-centric scheduling. J Supercomput 80(10):13738–13753
18. Herrera A, Ibáñez M, Stafford E, Bosque JL (2021) A simulator for intelligent workload managers in heterogeneous clusters. In: IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), pp 196–205
19. Sutton RS, Barto AG (2018) Reinforcement Learning: An Introduction. A Bradford Book, Cambridge
20. Fomperosa J, Ibañez M, Stafford E, Bosque JL (2023) Task scheduler for heterogeneous data centres based on deep reinforcement learning. In: Parallel Processing and Applied Mathematics. Springer, Berlin, Heidelberg, pp 237–248
21. Castillo E, Alvarez L, Moretó M, Casas M, Vallejo E, Bosque JL, Beivide R, Valero M (2018) Architectural support for task dependence management with flexible software scheduling. In: IEEE International Symposium on High Performance Computer Architecture, HPCA 2018, Vienna, Austria, February 24–28, pp 283–295
22. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal Policy Optimization Algorithms. arXiv: 1707:06347
23. Kingma DP, Ba J (2017) Adam: A Method for Stochastic Optimization. arXiv: 1412.6980
24. Izmailov P, Podoprikhin D, Garipov T, Vetrov D, Wilson AG (2019) Averaging Weights Leads to Wider Optima and Better Generalization. arXiv: 1803.05407
25. Bick D (2021) Towards Delivering a Coherent Self-Contained Explanation of Proximal Policy Optimization. Master's thesis, Rijksuniversiteit Groningen, Netherlands
26. Fan Y (2021) Job Scheduling in High Performance Computing. arXiv: 2109.09269
27. Stafford E, Bosque JL (2020) Improving utilization of heterogeneous clusters. J Supercomput 76(11):8787–8800
28. Maroulis S, Zacheilas N, Kalogeraki V (2019) A holistic energy-efficient real-time scheduler for mixed stream and batch processing workloads. IEEE Trans Parallel Distrib Syst 30(12):2624–2635
29. Bosque JL, Toharia P, Robles OD, Pastor L (2013) A load index and load balancing algorithm for heterogeneous clusters. J Supercomput 65(3):1104–1113
30. Tang W, Lan Z, Desai N, Buettner D (2009) Fault-aware, utility-based job scheduling on blue, gene/p systems. In: 2009 IEEE International Conference on Cluster Computing and Workshops, pp 1–10
31. Tsafrir D, Etsion Y, Feitelson DG (2007) Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans Parallel Distrib Syst 18(6):789–803

32. Mao H, Schwarzkopf M, Venkatakrishnan SB, Meng Z, Alizadeh M (2019) Learning scheduling algorithms for data processing clusters. ACM Special Interest Group on Data Communication. SIGCOMM '19. Association for Computing Machinery, New York, pp 270–288
33. Peng J, Li K, Chen J, Li K (2022) Hea-pas: A hybrid energy allocation strategy for parallel applications scheduling on heterogeneous computing systems. J Syst Architect 122:102329