

Interpretable Modeling of Deep Reinforcement Learning Driven Scheduling

Boyang Li

Department of Computer Science
Illinois Institute of Technology
Chicago, USA
bli70@hawk.iit.edu

Zhiling Lan

Department of Computer Science
University of Illinois Chicago
Chicago, USA
zlan@uic.edu

Michael E. Papka

Argonne National Laboratory
University of Illinois Chicago
Lemont, USA
papka@anl.gov

Abstract—In the field of high-performance computing (HPC), there has been recent exploration into the use of deep reinforcement learning for cluster scheduling (DRL scheduling), which has demonstrated promising outcomes. However, a significant challenge arises from the lack of interpretability in deep neural networks (DNN), rendering them as black-box models to system managers. This lack of model interpretability hinders the practical deployment of DRL scheduling. In this work, we present a framework called IRL (Interpretable Reinforcement Learning) to address the issue of interpretability of DRL scheduling. The core idea is to interpret DNN (i.e., the DRL policy) as a decision tree by utilizing imitation learning. Unlike DNN, decision tree models are non-parametric and easily comprehensible to humans. To extract an effective and efficient decision tree, IRL incorporates the Dataset Aggregation (DAgger) algorithm and introduces the notion of critical state to prune the derived decision tree. Through trace-based experiments, we demonstrate that IRL is capable of converting a black-box DNN policy into an interpretable rule-based decision tree while maintaining comparable scheduling performance. Additionally, IRL can contribute to the setting of rewards in DRL scheduling.

Index Terms—cluster scheduling; deep reinforcement learning; high-performance computing; interpretation; decision tree

I. INTRODUCTION

Cluster scheduling, also known as batch scheduling, is pivotal to high-performance computing (HPC). It is responsible for determining the order in which jobs are executed on an HPC system. Heuristics play a significant role in cluster scheduling, with the first-come, first-served (FCFS) policy being a widely employed scheduling approach on production systems [1]. To improve system utilization, backfilling is commonly used in cluster scheduling to enhance system utilization, which allows subsequent jobs to be moved ahead to utilize available resources [1].

Reinforcement learning, a subfield of machine learning, focuses on the automatic learning of decision-making strategies to maximize cumulative rewards through interactions with the environment [2]. Deep reinforcement learning (DRL), which combines reinforcement learning with deep neural networks, has been employed for cluster scheduling [3]–[6], demonstrating promising results. Unfortunately, very few, if any, of these approaches have been deployed on real-world production systems. One key hurdle is the *lack of model interpretability*. The superior performance of DRL scheduling stems from its

deep neural network (DNN) [7]; however, DNN appears as a black-box to system managers [8], making it challenging to comprehend, debug, deploy, and adjust in practice. As a result, system managers have reservations about using DRL scheduling on production systems. Therefore, it is essential to develop interpretable models that facilitate the practical deployment of DRL scheduling.

Many techniques have been developed to understand the behaviors of DNNs; however, there are two issues when applying these techniques to interpret DRL scheduling. First, existing techniques predominantly focus on monitoring neuron activations to identify the features that trigger them [9]. Consequently, system managers still need to possess knowledge of machine learning. Second, the current DNN interpretation methods are primarily designed for well-structured vector inputs such as images [10], [11] and sentences [12], [13], which are not applicable to the cluster scheduling problem. We believe there is a pressing need to provide a simple, deterministic, and easily understandable model for interpreting DRL scheduling.

Decision tree is non-parametric, and easy for humans to understand. However, training a decision tree in the context of DRL poses significant challenges [14]. While some studies have attempted to train some decision tree policies for reinforcement learning [15], subsequent work pointed out that these approaches struggled with relatively simple problems such as cart-pole [14]. Other efforts have sought to convert the DNNs employed in DRL to decision trees [14], [16]. This conversion is built upon a teacher-student training process, where the DNN policy serves as the teacher, generating input-output samples to construct a student decision tree for classification or regression. Compared to directly training decision tree policies for reinforcement learning, these converted decision trees can achieve better performance and handle more complex problems [14], [16].

In this work, we present IRL, an Interpretable modeling for deep Reinforcement Learning scheduling. IRL is based on imitation learning [17]. In simple terms, a decision tree is trained to mimic the DRL agent (i.e., deep neural network). Specifically, once a DRL agent (deep neural network) is built for cluster scheduling, this agent is used to generate input-output samples. These samples are used to train a decision

tree policy. However, we observe that the decision tree obtained through this straightforward conversion process does not closely resemble the original DRL agent as expected.

To overcome this issue, we employ the DAgger algorithm [18] to improve the scheduling performance of the decision tree. Furthermore, we have found that existing conversion algorithms [19] tend to produce a large decision tree with an excessive number of branches, which will greatly impact the effectiveness of the decision tree for decision making. To reduce the size of the decision tree, we introduce the concept of critical state in the scheduling environment. A critical state is defined as a system state that has a nontrivial impact on scheduling performance. By utilizing the critical state, we can decrease the size of the decision tree while still maintaining effectiveness. Specifically, we make three major contributions in this work:

- We present IRL, an interpretable modeling for DRL scheduling. IRL converts a black-box DRL policy to an easy-to-understand decision tree policy, thereby overcoming the lack of interpretability issue of DRL scheduling.
- We showcase how the interpretability of IRL can assist in the design of DRL scheduling, such as reward setting.
- Our trace-based experiments demonstrate the decision tree derived from IRL achieves comparable scheduling performance to the original DRL scheduling. In contrast to the black-box deep neural network, the decision tree is straightforward to comprehend, debug, and modify.

II. BACKGROUND

A. Cluster Scheduling in HPC

A cluster scheduler is responsible for allocating resources and for determining the order in which jobs are executed on an HPC system. When submitting a job, a user is required to provide two pieces of information: the number of compute nodes required for the job (i.e., job size) and job runtime estimate (i.e., walltime). The scheduler determines when and where to execute the job. The jobs are stored and sorted in the waiting queue based on a site's policy. The scheduler determines *when and where* to execute the jobs [20]. Once a new job is submitted, the job scheduler sorts all the jobs in the waiting queue based on a job prioritizing policy. A number of popular job prioritizing policies have been proposed, and one of the widely used policies is FCFS [1], which sorts jobs in the order of job arrivals.

In addition, backfilling is a commonly used approach to enhance job scheduling by improving system utilization, where subsequent jobs are moved ahead to utilize free resources. A widely used strategy is EASY backfilling which allows short jobs to skip ahead under the condition that they do not delay the job at the head of the queue [1].

While cluster scheduling is an active area of research in both HPC and cloud computing, these two communities target different workloads, resulting in divergent research approaches. In data centers such as those at Google or Microsoft, typical workloads include long-running services and directed acyclic

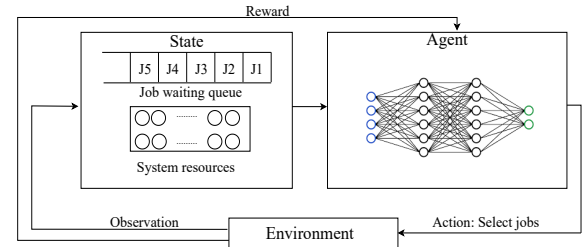


Fig. 1: The interaction between environment and an agent in reinforcement learning.

graph (DAG) jobs. Each job comprises multiple tasks, and the resource requirements for these tasks are often adjustable (i.e., malleability). In this context, the smallest scheduling unit is typically a task. In contrast, HPC is characterized by tightly-coupled parallel jobs (i.e., rigidity). Here, the scheduling unit is usually an entire job.

B. Deep Reinforcement Learning Scheduling

Reinforcement learning is an area of machine learning that is primarily focused on dynamic decision making where an intelligent agent takes actions in an environment with the goal of maximizing some reward [2]. The recent advancement of reinforcement learning enhanced with deep neural networks has yielded a number of promising performances for cluster scheduling [3]–[6]. In DRL driven scheduling, the agent is trained to learn a proper scheduling policy according to a specific scheduling objective (e.g., reward) provided by system managers. Once trained, the agent can automatically interact with the scheduling environment and dynamically adjust its policy as workload changes. Since the state space is typically enormous, memorizing all states becomes infeasible. DRL driven scheduling uses a deep neural network for approximation [21]. Figure 1 shows an overview of typical DRL driven scheduling. At each step, state is observed and fed to the scheduling agent. The agent provides job selection and receives the reward as the feedback.

C. Related Work

Interpreting deep neural networks is an active topic in the machine learning area [10], [12], [22], [23]. The interpretation methods can be broadly classified into two categories [24]. One is *local methods*, such as LIME [25]. These methods focus on explaining the prediction of a single instance in the dataset. The other is *global methods* [26], which aim to explain the average behavior of a machine learning model. In this work, we adopt global methods because our objective is to interpret the average behavior of a DRL-driven scheduling model.

Many global interpretability approaches focus on understanding the mechanism of DNNs, such as convolutional neural networks (CNN) [10], [12] and recurrent neural networks (RNN) [22], [23]. These approaches still require the system manager to have knowledge of machine learning, which is not the goal of IRL. In addition, these approaches are designed for well-structured inputs such as images and sentences, and

TABLE I: Comparison of heuristics, DRL scheduling, and IRL (Interpretable Reinforcement Learning).

Features \ Methods	Heuristics [1]	DRL scheduling [3], [4], [6]	IRL
Not black-box	✓	×	✓
Easy to comprehend	✓	×	✓
Superior scheduling performance	×	✓	✓
Rapid decision making	✓	×	✓

thus are not suitable for deep reinforcement learning driven scheduling problems.

Another way to achieve interpretability is to use only a subset of algorithms that create interpretable models. A decision tree is a commonly used interpretable model [27]. A decision tree is a decision-support hierarchical model that uses a tree-like model of decisions and their possible consequences. It can represent a complex policy. However, the main hurdle is that the decision tree is hard to train directly in reinforcement learning problems. There have been work training decision tree policies for reinforcement learning [15], but the following research pointed out that this method could not even achieve satisfactory performance in a not complicated reinforcement learning problem such as cart-pole [14]. To overcome this obstacle, some researchers refer to the idea of imitation learning to convert the DNN to a decision tree [14], [16]. The conversion is built on top of a teacher-student training process, where the DNN policy acts as the teacher and generates input-output samples to construct the student decision tree. Compared to directly training decision tree policies for reinforcement learning, the converted decision tree achieves better performance in many reinforcement learning problems [14], [16].

Interpreting DNNs to decision trees has shown promising results in many areas [28]–[30]. Meng et al. presented Metis to interpret deep learning based networking systems based on decision trees and hypergraphs [28]. Hu et al. demonstrated the decision tree converted from DRL achieved satisfactory performance in learning adaptive bitrate (ABR) algorithms [29]. Schmidt et al. applied the conversion from DNN to the decision tree in autonomous driving [30]. However, there is no effort made to interpret the DNNs in DRL driven scheduling in HPC. *To the best of our knowledge, IRL is the first attempt to interpret DRL scheduling in HPC.* Table I summarizes the key features of heuristics, DRL scheduling, and the proposed IRL — interpretable reinforcement learning scheduling.

III. IRL DESIGN

IRL, shown in Figure 2, is developed to provide an interpretable model for general DRL scheduling in HPC. The design of IRL is based on imitation learning, where the DNN policy of the DRL agent acts as the teacher and generates input-output samples to construct the student decision tree. Specifically, a trained deep neural network is obtained by training a workload trace. This neural network acts as the teacher and generate input-output samples from the original

training workload trace. These generated input-output samples are used as the training dataset to train an interpretable decision tree.

There are two issues in the above process: (1) the derived decision tree might not resemble the original deep neural network very well, and (2) the size of the decision tree could be huge. To overcome these obstacles, we employ two techniques. We integrate the DAgger algorithm [18] to address the former issue, and then introduce the critical state concept for the latter one.

The details are described in the subsequent subsections. In order to make the description of our method more straightforward, we use deep Q-network (DQN) as an illustrative example. In Section III-A, we describe DQN driven scheduling. In Section III-B, we present the conversion from the DQN policy to the decision tree policy. We describe the generation of the input-output sample dataset by DQN, and the generation of the decision tree with the sample dataset. We also describe how to integrate DAgger in our design to improve the scheduling performance of the decision tree, and how to reduce the size of the decision tree via the concept of critical state.

A. DQN Scheduling

Recent studies have employed various deep reinforcement learning (DRL) methods for cluster scheduling [5], [6]. Regardless of the specific DRL method employed, the underlying principle remains consistent. In this study, we utilize DQN [21] as a practical example. The scheduling system, driven by a DRL agent, follows the approach outlined in [6]. The DRL scheduling agent aims to optimize scheduling performance by making decisions on when and which jobs in the waiting queue should be allocated to available computer resources. At each scheduling instance, the agent encodes both job and system information into a vector, which is fed as an input to the neural network. Based on the neural network’s output, the agent selects jobs from the wait queue and then receives a reward signal from the scheduling environment.

In DQN, the deep neural network is utilized to approximate Q-value as $Q(s_k, a_k)$ (i.e., the expected cumulative reward of taking an action a_k in the state of s_k). DQN processes one job at a time and produces the expected Q-value for this job. The input of DQN is a 1-D vector containing job size (i.e., number of nodes requested), job length (i.e., estimated job runtime), and system utilization. The output is a single neuron corresponding to the expected Q-value of the job. The scheduler enforces a window of jobs at the front of the job

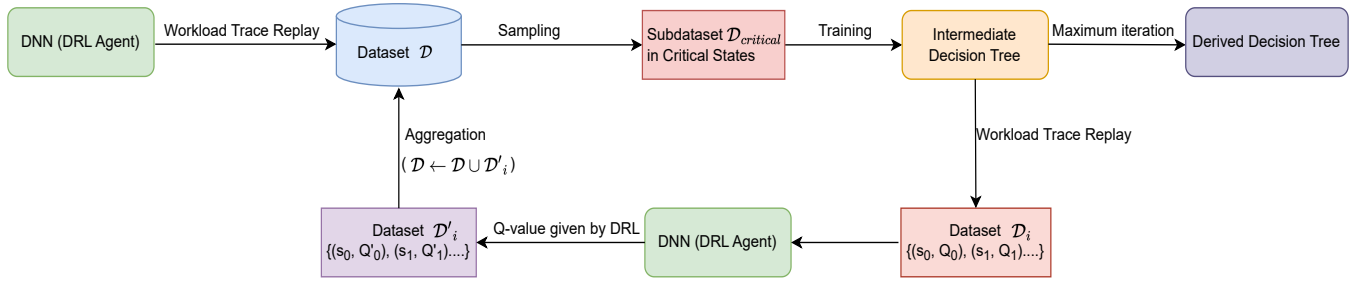


Fig. 2: Overview of IRL design. A cylinder represents a data repository $\mathcal{D}$. A rounded rectangle denotes a scheduling policy, which is either DRL or a decision tree. A rectangle represents a dataset. It is sampled from $\mathcal{D}$, or produced by a scheduling policy (decision tree or DRL).

wait queue. The window alleviates the job starvation problem by providing higher priorities to older jobs.

In order to explore various actions, during the training, the agent follows the ϵ -greedy policy (i.e., the agent randomly chooses a job instead of the job with the highest Q-value with probability ϵ). During the testing or inference time, the agent selects the job with the highest Q-value.

B. Decision Tree Conversion

Decision tree is a supervised learning method commonly used for classification or regression. In this work, we use decision tree for regression. Specifically, in order to interpret the DQN policy, our IRL works as follows. The input of the decision tree is the state, and the output of the decision tree is the imitated output (Q-value) of DQN. The DQN scheduling agent replays the workload trace to produce a trajectory of (state, Q-value) pairs. This trajectory will be used as the training dataset $\mathcal{D}$ of the decision tree so that the output of the decision tree will approximate the output of the DQN.

The process of extracting an effective and efficient decision tree to interpret the DQN policy presents several challenges. Firstly, we observed that the decision tree trained once may not resemble the DQN policy well. When replaying the same workload trace, the decision tree agent may choose a different job from the DQN agent, since the imitated Q-value by the decision tree is unlikely to precisely match the Q-value output by DQN. Consequently, due to the varying job selection, the decision tree is prone to jump to a state which it has never seen in the training dataset $\mathcal{D}$, and behaves unpredictably in these unseen states.

To address this issue, we incorporate DAgger [18] into the decision tree conversion. DAgger is an iterative training algorithm. Instead of training the decision tree one time, we train it multiple times. After each training iteration, the newly generated decision tree is used to replay the workload trace and demonstrate its policy to the DQN agent. As a result, a new trajectory of (state, Q-value) pairs following the newly generated decision tree policy is produced, denoted as $\mathcal{D}_i$. This new trajectory may contain unseen states in the training dataset $\mathcal{D}$. As a teacher, the DQN agent assigns the Q-values to the states in this new trajectory, and aggregate this newly produced trajectory $\mathcal{D}_i$ into the training dataset $\mathcal{D}$.

The updated dataset $\mathcal{D}$ is then used to train the decision tree in the next iteration. This process repeats multiple times until the maximum iteration is reached. In this work, the maximum iteration is set to 5. The middle part of Figure 2 illustrates the iterative process of generating the decision tree.

Next, another challenge of the decision tree conversion is that the generated decision tree with DAgger is normally huge [14]. The cost of decision time is proportional to the size of the decision tree. Hence a smaller-sized decision tree without sacrificing scheduling performance is desired. To overcome the issue, IRL introduces the concept of *critical state* with the goal of generating a reduced-sized decision tree.

Our design is based on a key observation, that is, when the system is relatively idle (i.e., few jobs are in the waiting queue), the job selection has less impact on the scheduling performance. There are two reasons. First, the range of job selection is limited when there are a few jobs in the waiting queue. For instance, when there is only one job in the waiting queue, this job is selected anyway. Similarly, if the available resources can accommodate only one job, it is chosen among all the waiting jobs. Second, even if a job is not selected at this scheduling instance, it is likely to be scheduled shortly thereafter.

In the design of IRL, we define *critical state* as the system state when the number of jobs in the waiting queue is greater than a threshold, and *non-critical state* as the state when the number of jobs in the waiting queue is less than or equal to the threshold. The threshold should be chosen based on the workload to balance the tree size and the scheduling performance. Only samples with critical states ($\mathcal{D}_{critical}$) in the dataset $\mathcal{D}$ are used to generate the decision tree. Put together, Algorithm 1 shows the complete pseudo code of the IRL method.

IV. EVALUATION

We have implemented IRL using TensorFlow [31]. Our deployment of IRL involves integrating it with the discrete-event driven scheduling simulator CQSim [32]. To thoroughly evaluate the performance of IRL, we conduct extensive trace-based simulations using actual workload traces.

In this section, we describe our evaluation methodology, and the experimental results are listed in the next section.

Algorithm 1 IRL algorithm

```

1: Initialize DataSet  $\mathcal{D} \leftarrow$  Trajectory (state, Q-value) gener-
   ated by DQN via workload trace replay
2: for  $i = 1$  to  $N$  do
3:    $\mathcal{D}_{critical} \leftarrow \mathcal{D}$  in critical states
4:   Train decision tree  $DT_i$  from  $\mathcal{D}_{critical}$ 
5:   DataSet  $\mathcal{D}_i \leftarrow$  Trajectory generated by  $DT_i$  via work-
   load trace replay
6:    $\mathcal{D}'_i \leftarrow$  Q-values are assigned by DQN for the states
   in  $\mathcal{D}_i$ 
7:    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}'_i$ 
8: end for

```

A. Workload Traces

In our evaluation, two real workload traces are used [33], [34]. Table II summarizes the two workload traces. For each trace, 10,000 jobs are employed for training to build both the DQN policy and the decision tree. The convergence of the DQN policy is confirmed by assessing its convergence rate. Subsequently, an additional set of 2,500 unseen jobs is utilized for inference testing.

B. Comparison Methods

We compare IRL with the following three methods.

- **FCFS** represents first come first served method, which is the default scheduling policy deployed on many production supercomputers [1]. FCFS prioritizes jobs based on their arrival times.
- **DQN** denotes the reinforcement learning scheduling policy [6]. It acts as the teacher to construct the decision tree policy.
- **Dagger** represents a generated decision tree policy without the introduction of critical state.

In addition, backfilling is adopted in each of these methods to mitigate resource fragmentation [1]. FCFS comes with EASY Backfilling [1]. In DQN and IRL, the backfilled job is selected by the agent.

C. Experiment Setup

When training the DQN agent, the reward is set to $\sum_{j \in J} -\frac{1}{t_j}$ [3], where J is the set of jobs currently in the system, t_j is the (ideal) running time of the job. This reward function aims to minimize the average job slowdown. The window size of the waiting queue is set to 20. ϵ is set to 1.0 at the beginning of the training and decays at the rate of $\alpha = 0.995$. The input layer contains three neurons, and three fully-connected hidden layers activated by rectified linear units (ReLU) [35] with 32, 16, 8 neurons are used separately. The output layer contains one neuron.

In our experiments, we set the critical state threshold to three, meaning a critical state is defined when the number of jobs in the waiting queue is more than three. Our sensitivity study indicates that for these workloads, this configuration can balance the tree size and its performance. Scikit-learn library is used to generate the decision tree [36].

TABLE II: Workload Traces

Workload	Site	System Size	Period
SP2	SDSC	128	April, 1998-April, 2000
DataStar	SDSC	1,664	March, 2004-April, 2005

D. Evaluation Metrics

Following the common practice [37], we use the following metrics to evaluate different scheduling methods.

- 1) *Average job wait time*: the average interval between job submission to job start time.
- 2) *Average job slowdown*: the average ratio of job response time (job runtime plus wait time) to the actual runtime, representing the responsiveness of a system.

V. RESULTS

In this section, we present the experimental results. Our analysis centers upon four questions:

- 1) What can IRL contribute to DRL driven scheduling? (Section V-A.)
- 2) Does the decision tree obtained through IRL exhibit comparable scheduling performance to DRL scheduling? (Section V-B)
- 3) Does the use of critical state reduce the size of the decision tree? (Section V-C)
- 4) Does IRL introduce less runtime overhead than DRL scheduling? (Section V-D)

A. Reward Setting

Neural networks used in DRL function as black-box models. However, since the decision tree obtained through IRL imitates DRL, we can indirectly understand the DRL scheduling policy by examining the resulting decision tree.

In order to illustrate the use of IRL, we conduct a case study to demonstrate the contribution of IRL to DRL scheduling in terms of reward setting. Specifically, we utilize the SP2 workload as our experimental scenario. Within this case study, we explore two different reward settings: one that aligns with our scheduling objective, and the other that fails to do so. Through this use case, we showcase the interpretability of IRL in identifying key features that significantly influence the decision-making process of the DRL agent. Furthermore, IRL enables us to explain the rationale behind whether a particular reward setting can achieve the scheduling objective or not.

In this case study, the objective of the DQN agent is to minimize the average job slowdown. Two reward settings are used: (1) *Reward A* — the reward is set to $\sum_{j \in J} -\frac{1}{t_j}$ [3], and (2) *Reward I* — the reward is set to $\sum_{j \in J} -\frac{(w_j + t_j)}{t_j}$, where J is the set of jobs currently in the system, t_j is the (ideal) running time of the job, and w_j is the waiting time of the job. Reward I appears reasonable since it aims to maximize the negative values of all the jobs' slowdown in the system, and our objective is to minimize the average job slowdown. Since job waiting time is in the reward, we also add job waiting time in the input feature.

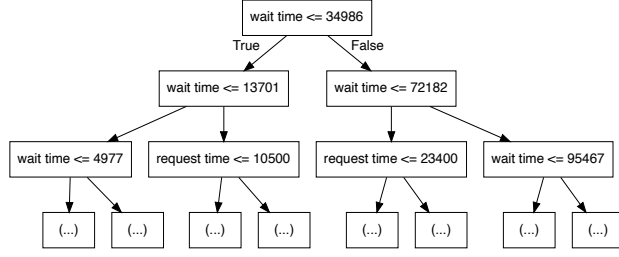


Fig. 3: Decision tree (depth=10) generated by IRL from the DQN agent with *Reward I*. Only the first two depths are presented in the figure due to the space limitation. Note that the decision tree's branches primarily revolve around job wait time for decision-making.

Figure 3 shows the decision tree generated by IRL from this DQN agent using *Reward I*. The depth of the decision tree is set to ten, and only the first two depths are presented due to space limitation. It can be observed that this decision tree primarily bases its decisions on job waiting time. Further analysis reveals that this decision tree favors jobs with longer wait times, and its approach closely aligns with FCFS. The underlying reason is that if we want to maximize $\sum_{j \in J} -\frac{(w_j+t_j)}{t_j}$, the DQN agent is trained to select (remove) the job with the minimum value of $-\frac{(w_j+t_j)}{t_j}$. A longer wait time leads to a smaller value of $-\frac{(w_j+t_j)}{t_j}$. In the end, the DQN is trained to prefer early coming jobs. If the scheduling objective is to minimize average slowdown, each scheduling decision is supposed to consider the future job wait time instead of the past job wait time. Therefore, the past job wait time should not be a factor in either the reward or input feature.

Figure 4 shows the decision tree generated by IRL from the DQN agent using *Reward A*. It can be observed that this decision tree primarily bases its decisions on the job length (the requested running time). Further analysis finds that the decision tree prefers to select short jobs. Indeed, according to the definition of job slowdown, the scheduling objective of minimizing average job slowdown tends to allow shorter jobs to wait less time. This demonstrates that the interpreted DQN policy indicates that the reward setting – *Reward A* in the DQN is appropriate.

Figure 5 shows the DQN scheduling performance under different reward settings. Appropriate reward setting (*Reward A*) can reduce the average job wait time and slowdown by up to 66% compared to inappropriate reward setting. This study clearly illustrates that IRL contributes to analyzing reward settings in DRL driven scheduling.

B. Scheduling Performance

Figure 6 and Figure 7 compare different scheduling methods under SP2 and DataStar workloads in terms of average job slowdown and average job wait time. Different tree depths are used in IRL and DAgger to observe the impact of the tree depth on the scheduling performance.

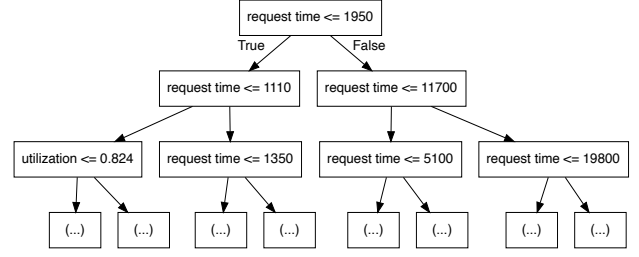


Fig. 4: Decision tree (depth=10) generated by IRL from the DQN agent with *Reward A*. Only the first two depths are presented in the figure due to the space limitation. Note that the decision tree's branches mainly revolve around requested running time for decision-making.



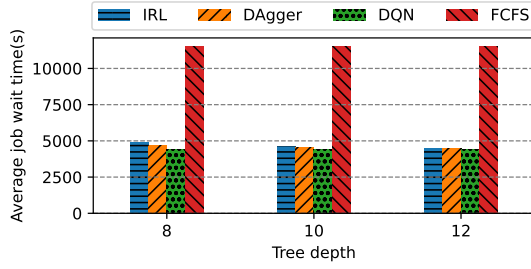
Fig. 5: Comparison of scheduling performance with DQN under different reward settings.

We observe that IRL yields much better scheduling performance than FCFS. On SP2, IRL can reduce average job wait time and average job slowdown by up to 70% compared to FCFS. On DataStar, the average job wait time and slowdown can be shortened by up to 36% in comparison with FCFS. In addition, we notice that IRL achieves comparable scheduling performance in contrast to DQN. On SP2, when the tree depth is 10 or 12, the increase in average job wait time and slowdown of IRL is within 3% compared to DQN. On DataStar, when the tree depth is 10 or 12, the increase in average job wait time and slowdown of IRL compared to DQN remains within 5%.

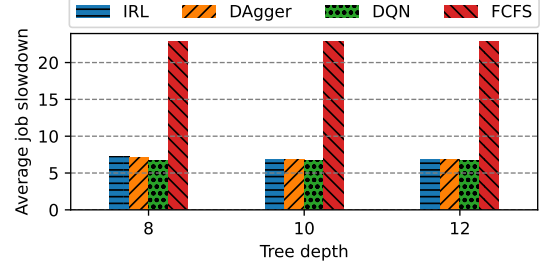
We also compare the performance between IRL and DAgger to examine the impact of critical state on the scheduling performance. On SP2, when the tree depth is 10 or 12, the scheduling performance loss between IRL and DAgger is within 1%. On DataStar, when the tree depth is 10 or 12, the increase in average job wait time and slowdown remains within 2%. This suggests that the introduction of critical state causes negligible performance loss.

C. Tree Reduction

The time cost of making decisions with the decision tree is proportional to the size of the tree. Thus, it is preferable to

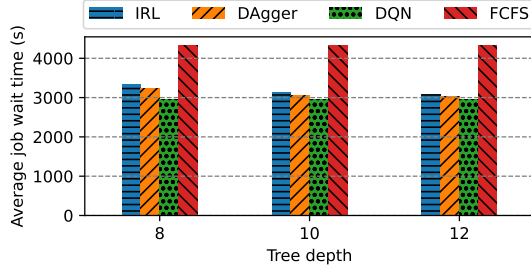


(a) Average job wait time

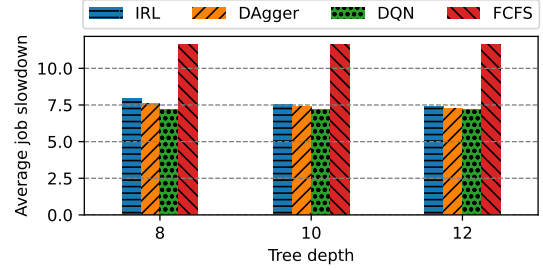


(b) Average job slowdown

Fig. 6: Scheduling performance on SP2 workload trace. The tree depth is set to 8, 10, 12 separately in IRL and DAgger.

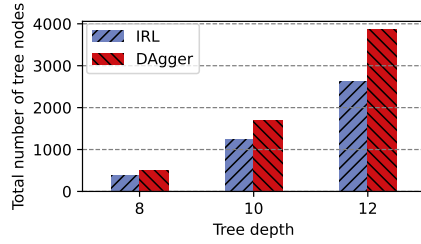


(a) Average job wait time

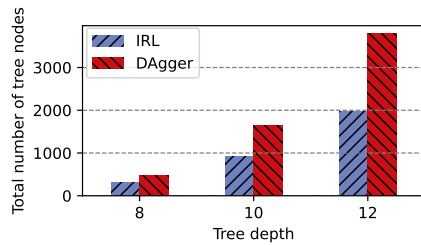


(b) Average job slowdown

Fig. 7: Scheduling performance on DataStar workload trace. The tree depth is set to 8, 10, 12 separately in IRL and DAgger.



(a) SP2



(b) DataStar

Fig. 8: Comparison of tree size generated by IRL and DAgger.

have a decision tree that is compact in size. In this subsection, we analyze the impact of introducing critical state on the size of the tree.

Figure 8 compares the sizes of the decision tree generated by IRL and DAgger. On the SP2 workload, IRL reduces the tree size by up to 34% compared to DAgger. On the DataStar workload, IRL delivers a reduction of up to 48% in tree size. The reduction of tree size on the DataStar workload is larger than that on the SP2 workload. We attribute this to the relative idleness of the DataStar system compared to the SP2 system. As demonstrated in Figure 6 and Figure 7, the average job

wait time of the DataStar workload is significantly less than that of the SP2 workload. Consequently, with the introduction of critical state, IRL stands to benefit more in terms of tree size reduction.

D. Runtime Overhead

In our experiments, with the tree depth set to 10, IRL takes approximately 0.0003 seconds for each job selection, whereas DQN requires around 0.02 seconds. Hence, IRL introduces significantly less overhead compared to DQN. All the experiments were conducted on a personal computer configured with an Intel 2 GHz quad-core CPU and 16 GB memory.

VI. CONCLUSIONS

While DRL driven scheduling exhibits impressive performance compared to heuristic and optimization methods, there are significant limitations that hinder its practical deployment, particularly the lack of model interpretability. The superior performance of DRL driven scheduling stems from the neural network employed in the design; however, the neural network appears as a black box model to system managers as it is incomprehensible to debug, deploy, and adjust in practice.

In this work, we have presented IRL, an interpretable model for DRL driven scheduling. IRL converts the black-box neural network employed in DRL driven scheduling to an interpretable decision tree model, which not only can represent a complex decision policy, but also is easy for humans to understand. The design of IRL poses several technical challenges. In this work, we have described the detailed strategies for the design of IRL. Moreover, we have shown the use of IRL via several case studies. Specifically, one case study demonstrates how IRL can contribute to the DRL scheduling design (e.g.,

reward setting), and another case study shows that IRL can deliver comparable scheduling performance as compared to the existing scheduling methods including the widely used heuristic method and the DRL scheduling method.

While this study focuses on DQN, IRL can be applied to other DRL methods as well. As part of our future work, we will investigate other interpretable models such as the regression model to IRL. To the best of our knowledge, this study represents the first exploration of interpretable reinforcement learning scheduling for HPC. We hope this work will pave the way for further investigations in the field of interpretable reinforcement learning scheduling for HPC.

ACKNOWLEDGMENT

This work is supported in part by US National Science Foundation grants CCF-2109316, CCF-2119294, and U.S. Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

REFERENCES

- [1] A. W. Mu'alem and D. G. Feitelson, "Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 6, pp. 529–543, 2001.
- [2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [3] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proceedings of the 15th ACM workshop on hot topics in networks*, 2016, pp. 50–56.
- [4] B. Li, Y. Fan, M. E. Papka, and Z. Lan, "Encoding for reinforcement learning driven scheduling," in *Workshop on Job Scheduling Strategies for Parallel Processing*. Springer, 2022, pp. 68–87.
- [5] H. Mao, M. Schwarzkopf, S. B. Venkatakrisnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, 2019, pp. 270–288.
- [6] Y. Fan, B. Li, D. Favorite, N. Singh, T. Childers, P. Rich, W. Allcock, M. E. Papka, and Z. Lan, "Dras: Deep reinforcement learning for cluster scheduling in high performance computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 12, pp. 4903–4917, 2022.
- [7] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [8] A. Dethise, M. Canini, and S. Kandula, "Cracking open the black box: What observations can tell us about reinforcement learning agents," in *Proceedings of the 2019 Workshop on Network Meets AI & ML*, 2019, pp. 29–36.
- [9] D. Bau, B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Network dissection: Quantifying interpretability of deep visual representations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 6541–6549.
- [10] M. Du, N. Liu, Q. Song, and X. Hu, "Towards explanation of dnn-based prediction with guided feature inversion," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 1358–1367.
- [11] J. Wang, L. Gou, W. Zhang, H. Yang, and H.-W. Shen, "Deepvid: Deep visual interpretation and diagnosis for image classifiers via knowledge distillation," *IEEE transactions on visualization and computer graphics*, vol. 25, no. 6, pp. 2168–2180, 2019.
- [12] Q. Liu, Y. Zhu, Z. Liu, Y. Zhang, and S. Wu, "Deep active learning for text classification with diverse interpretations," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 3263–3267.
- [13] M. Toneva and L. Wehbe, "Interpreting and improving natural-language processing (in machines) with natural language-processing (in the brain)," *Advances in neural information processing systems*, vol. 32, 2019.
- [14] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," *Advances in neural information processing systems*, vol. 31, 2018.
- [15] D. Ernst, P. Geurts, and L. Wehenkel, "Tree-based batch mode reinforcement learning," *Journal of Machine Learning Research*, vol. 6, 2005.
- [16] N. Frosst and G. Hinton, "Distilling a neural network into a soft decision tree," *arXiv preprint arXiv:1711.09784*, 2017.
- [17] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.
- [18] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings*, 2011, pp. 627–635.
- [19] W.-Y. Loh, "Classification and regression trees," *Wiley interdisciplinary reviews: data mining and knowledge discovery*, vol. 1, no. 1, pp. 14–23, 2011.
- [20] B. Li, S. Chunduri, K. Harms, Y. Fan, and Z. Lan, "The effect of system utilization on application performance variability," in *Proceedings of the 9th International Workshop on Runtime and Operating Systems for Supercomputers*, 2019, pp. 11–18.
- [21] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
- [22] T. Feng and C. Yue, "Visualizing and interpreting rnn models in url-based phishing detection," in *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies*, 2020, pp. 13–24.
- [23] H. Wu, A. Huang, and J. W. Sutherland, "Layer-wise relevance propagation for interpreting lstm-rnn decisions in predictive maintenance," *The International Journal of Advanced Manufacturing Technology*, pp. 1–16, 2022.
- [24] C. Molnar, *Interpretable machine learning*. Lulu.com, 2020.
- [25] M. T. Ribeiro, S. Singh, and C. Guestrin, "Why should I trust you?" Explaining the predictions of any classifier," in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.
- [26] M. Du, N. Liu, and X. Hu, "Techniques for interpretable machine learning," *Communications of the ACM*, vol. 63, no. 1, pp. 68–77, 2019.
- [27] S. B. Kotsiantis, "Decision trees: a recent overview," *Artificial Intelligence Review*, vol. 39, pp. 261–283, 2013.
- [28] Z. Meng, M. Wang, J. Bai, M. Xu, H. Mao, and H. Hu, "Interpreting deep learning-based networking systems," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 154–171.
- [29] Q. Hu, H. Nori, P. Sun, Y. Wen, and T. Zhang, "Primo: Practical Learning-Augmented Systems with Interpretable Models," in *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, 2022, pp. 519–538.
- [30] L. M. Schmidt, G. Kontes, A. Plinge, and C. Mutschler, "Can you trust your autonomous car? interpretable and verifiably safe reinforcement learning," in *2021 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2021, pp. 171–178.
- [31] IRL on GitHub. [Online]. Available: <https://github.com/SPEAR-UIC/IRL>
- [32] CQSim. [Online]. Available: <https://github.com/SPEAR-UIC/CQSim>
- [33] D. G. Feitelson, D. Tsafrir, and D. Krakov, "Experience with using the parallel workloads archive," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2967–2982, 2014.
- [34] PWA. [Online]. Available: <https://www.cs.huji.ac.il/labs/parallel/workload/>
- [35] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.
- [36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg et al., "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.
- [37] D. G. Feitelson and L. Rudolph, "Metrics and benchmarking for parallel job scheduling," in *Job Scheduling Strategies for Parallel Processing: IPPS/SPDP'98 Workshop Orlando, Florida, USA, March 30, 1998 Proceedings 4*. Springer, 1998, pp. 1–24.