Multi-Agent Deep Reinforcement Learning for Collaborative Task Scheduling

Mali Imre Gergely
Faculty of Mathematics and Computer Science, Babeș-Bolyai University, Kogălniceanu Street, Cluj-Napoca, Romania

Keywords: Multi-Agent Reinforcement Learning, Multi-Agent Systems, Resource Management, Task-Scheduling.

Abstract: Efficient, scalable and cost-efficient resource management is a multi-faceted online decision making problem, faced more and more in networking and cloud computing. More specifically, task-scheduling stands out as a complex challenge, solving which is critical for the optimal functioning of today's systems. Traditional heuristic approaches to scheduling are laborious to design and especially difficult to tune, therefore various machine-learning based methods have been proposed. Reinforcement Learning (RL) showed great results in similar decision making problems, and many existing approaches employ RL to solve task scheduling problems. Most of these works consider either single-agent scenarios (and thus suffer from scalability issues), or the existing multi-agent applications are heavily specialised. We propose a general-purpose multi-agent RL framework that can successfully learn collaborative optimal scheduling policies, making one step further towards clouds and networks that are both scalable and autonomous. Our experiments show that these agents can collaboratively learn optimal scheduling policies for dynamic workloads.

1 INTRODUCTION

Optimal resource management is a critical nuance in the operation of various real-life systems such as in logistics, computer networks, cloud-based applications, transportation, manufacturing. In particular, computer networks and cloud-based applications have experienced an immense growth in popularity recently. These systems typically must serve large numbers of users, and must manage substantial volumes of data and significant computational demands. Effective task scheduling is a vital aspect of the optimal functioning of these systems.

Task scheduling itself is a well-studied NP-hard problem, which under certain formulations is equivalent to the intensely scrutinized "Job-Shop-Scheduling" problem (Dauzère-Pérès et al., 2023). Efficient task-scheduling in computer networks ensures low makespan of tasks, thus achieving low latency, high reliability and good performance. Classical approaches to task-scheduling generally use complex hand-crafted heuristics that are not only tedious to conceive and difficult to calibrate, they also lack flexibility with respect to changes in the metrics of interest and other aspects of the environment. To combat this, RL-based solution proposals have been more and more frequent, especially since DeepRM was introduced (Mao et al., 2016).

While some approaches are directly implemented for practical use in real-world cluster management scenarios such as machine-learning clusters (Zhao and Wu, 2021), a significant portion of existing studies approach these problems by developing and using simulated environments (Mao et al., 2016; Fan et al., 2022). These simulators and testbeds offer a controllable setting that ease the process of experimentation with different algorithms and strategies, and allow for more rigorous analysis of results. This can ultimately be useful in developing efficient and robust solutions that later can be incorporated into real clusters.

Many of the existing research considers a single agent, generally handling task scheduling for an entire cluster. The existing exceptions that employ multi-agent methods (Zhao and Wu, 2021) are largely situation-specific and are heavily intertwined with the intricacies of the type of cluster or system they consider, lacking generality. Single-agent approaches are problematic since the growth in the number of machines/instances typically increases the action space for RL-based approaches, making them more and more inefficient as these systems scale.

This paper considers a novel approach that formulates multi-resource, multi-machine and multi-agent task scheduling as a collaborative Partially Observ-

Gergely, M.
Multi-Agent Deep Reinforcement Learning for Collaborative Task Scheduling.
DOI: 10.5220/0012434700003636

In *Proceedings of the 16th International Conference on Agents and Artificial Intelligence (ICAART 2024) - Volume 3*, pages 1076-1083
ISBN: 978-989-758-680-4; ISSN: 2184-433X

able Markov Game (POSG). However, we use an implementation based on a novel pattern and provide a configurable environment based on the Agent-Environment-Cycle (AEC) model (Terry et al., 2021). Since AEC games are also proved to be equivalent to POSG-s and Extensive Form Games (EFG), this abstraction is judicious.

We then apply Proximal Policy Optimization (PPO) (Schulman et al., 2017) in a decentralised way and experimentally prove that multiple homogeneous independent RL agents can jointly learn comparable to or better scheduling policies than heuristic-based approaches. Additionally, we explore the effects of local vs. global observations and rewards on performance.

2 BACKGROUND

2.1 Multi-Agent Reinforcement Learning

Reinforcement Learning is one of the main paradigms of machine learning that models decision-making problems as Markov Decision Processes and builds on the idea of learning from interaction (Sutton and Barto, 2018).

Formally, a Markov Decision Process can be characterized as a tuple (S, A, P, R, γ), where S is the set of possible states, A is the set of possible actions, P encapsulates the transition probabilities to describe the dynamics of the environment, R is a reward function and γ is called the discount factor that controls the short- or farsightedness of the agent.

The agent interacts with the environment by observing at each timestep t a state s_t (or a local observation o_t in partially observable settings), based on which it decides on action a_t and sends it to the environment. The environment then transitions to the next state according to P, and the agent receives reward r_{t+1} and observes the next state s_{t+1}. The agent's goal is to learn a policy π, according to which action selections are made in a way that maximises cumulative rewards.

In multi-agent scenarios, each agent might have a different set of actions, and might observe different subsets of the state. What makes multi-agent scenarios more challenging is that from the perspective of each agent, the rest are part of the environment. Thus, when multiple agents are learning at the same time, the environment exposes an inherent non-stationarity.

Regarding RL methods, Proximal Policy Optimisation stands out as an advanced policy-gradient method with a unique approach to the balance between exploitation and exploration. Unlike classical methods, PPO optimises a clipped surrogate objective function. The clipping mechanism is central to this approach as this effectively prevents the algorithm from committing updates so large that would change the policy acutely (thus the name "proximal"). The objective function can be written as follows:

$$L(\theta) = \hat{\mathbb{E}}[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)]$$

Here, $\hat{A}_t$ is an approximator of the advantage function (Schulman et al., 2015), $r_t(\theta)$ is the ratio between the new and the old policy, and ε is a hyperparameter, generally set to 0.2. The clip function modifies the surrogate objective and prevents it from jumping outside of the interval $[1-\varepsilon, 1+\varepsilon]$.

PPO has gained immense popularity due to its simplicity, applicability and effectiveness in various scenarios, from chip design (Mirhoseini et al., 2021) to robotics (OpenAI et al., 2018). Furthermore, PPO has proved to be surprisingly effective in cooperative multi-agent games, (Yu et al., 2022) which further validates our idea to apply multi-agent PPO to task scheduling. To our best knowledge however, this has not yet been done in the literature for this particular problem in a multi-agent setting.

2.2 Agent Environment Cycle Games

Due to recent advances, RL has gained wide recognition and popularity, especially in the field of games and game theory (Silver et al., 2017). This has led to the development of tools, frameworks and libraries such as OpenAI gym (Brockman et al., 2016), OpenSpiel (Lanctot et al., 2019), and PettingZoo along with the Agent Environment Cycle (AEC) model (Terry et al., 2021). These tools prove to be useful in rigorous modeling and experimentation in RL, and are applicable to scheduling simulators as well.

Among these, the AEC model and the corresponding library, PettingZoo stand out in that it successfully addresses common errors and provides clearer attribution of rewards by providing a more conceptually consistent approach. Agents in the AEC model receive observations, take actions, and receive rewards sequentially, which mirrors the sequential nature of decision-making in most real-world scenarios. This approach not only simplifies the attribution of rewards and actions but also implicitly prevents the introduction of race conditions, frequently encountered in models where agent actions are committed simultaneously. Moreover, AEC games are also proved to be

equivalent to POSG games, which makes the model widely applicable.

Formally, AEC games can be characterized as a tuple containing $(S, s_0, N, (A_i)_{i\in[N]}, (T_i)_{i\in[N]}, P, (R_i)_{i\in[N]}, (\Omega_i)_{i\in[N]}, (O_i)_{i\in[N]}, v)$, where:

- S is the set of states, s_0 is the initial state.
- N is the number of agents. Typically, agents are numbered from 1 to N, and the environment agent is denoted with 0.
- A_i is the set of actions for agent i. For convenience, A_0 is generally set to be void.
- T_i is a state transition function for agent i, while P is the transition function for the environment.
- R_i is the set of possible rewards for agent i.
- Ω_i is the set of possible observations for agent i, while O_i is the observation function.
- v is a special function that computes the next agent in the agent environment cycle.

Under this formulation, games start in state s_0. The first agent to act is the environment agent, after which the game continues in turns, agents receive an observation ω_i, and choose action a_i. Then the game transitions deterministically to the next state according to T_i. When $i = 0$, a stochastic transition according to P happens. Agents are taking their turns in order defined by v. Analogously, at every step agents receive a partial reward r_i which are aggregated at the end of each cycle. This model allows for turn-based as well as for synchronous games.

For our purposes, the AEC game will mean that each scheduler agent will take turn in choosing tasks to schedule, after which the environment agent will transition to the next timestep. This ensures that all agents have their inner clocks synchronized and we avoid the necessity of complex time synchronization mechanisms.

2.3 Task Scheduling

Efficient task scheduling has a pivotal role in optimizing resource utilization and maintaining high quality of service, upholding service-level agreements and meeting user requirements. These aspects play an increasingly important role in environments experiencing fluctuating workloads and changing priorities. The task scheduling problem is generally framed in terms of the following components:

1. **jobs**, which are waiting to be processed. Jobs often have known resource profiles
2. **machines**, which process jobs according to their resource constraints, and can only process tasks that fit within their resource constraints
3. **processing time**
4. **schedule**, which is a plan detailing which tasks to be processed on which machines at what times
5. **scheduler**, the entity that finds a schedule
6. **objective**, typically the minimisation of some metric like job slowdown, makespan, etc.

Under this formulation, our goal is to come up with a set of RL-based scheduler agents, each handling a subset of machines of a cluster, concurrently and collaboratively finding scheduling plans that are optimal under dynamic workloads.

3 RELATED WORK

The field of resource management has gained significant attention, focusing particularly on task scheduling in cloud environments, as well as job-shop scheduling challenges. Reinforcement learning has served as a key technique in recent advancements (Wang et al., 2022; Liu et al., 2020).

DRAS (Fan et al., 2022) aims to solve the problem of task scheduling, specialised in High Performance Computing (HPC) networks. The authors also provide a gym-like simulation environment for training called CQGym. They present multiple single-agent RL methods including Deep Q-Learning, A2C, Vanilla Policy-Gradient and PPO, and evaluate them on real HPC cluster traces. Their experiments show the PPO scheduler agent to perform the best.

The work of Tassel et al. also comes close to ours (Tassel et al., 2021) in that it both addresses job-shop scheduling, and employs PPO, but only considers a single-agent scenario. Instead of randomly generated data, this work considers benchmark datasets instead.

Another recent study (Zhu et al., 2023) presents a double-deep Q-network based Multi-Agent Reinforcement Learning (MARL) approach to solve real-time scheduling in a robot-assisted flexible job shop, using centralized training and decentralized execution on job sequencing, process planning and machine selection in a simulated environment. Their results show improvements up to 50% as compared to heuristics. Another popular approach to job-scheduling with MARL is using homogeneous agents and Graph Neural Networks to harness topology-awareness in large clusters (Zhao and Wu, 2021).

DeepMAG (Zhang et al., 2023) applies a Deep Q-Network based cooperative multi-agent method, operating over a multi-agent graph based on interoperation relationships between machines. While this work addresses manufacturing settings primarily, it

also comes close to our approach as the problem of cooperation is directly addressed.

DeepRM and DeepRM 2 (Mao et al., 2016; Ye et al., 2018) are the most prominent works in the area of task scheduling with RL, one of their most significant contribution being a flexible, intuitive and extensible simulation environment, on which our environment is based as well. DeepRM first demonstrated the efficacy of RL methods over traditional heuristics. It is also the first approach that employs Policy Gradient methods, having used the REINFORCE algorithm (Sutton et al., 1999). While other works base their simulation environment on the one described in DeepRM as well, including the extension to multiple machines (Paduraru et al., 2023), to our best knowledge ours is the first one to extend it to the multi-agent setting.

4 DESIGN

This section elaborates on the design of the main components of our approach, including the environment, agents and rewards.

4.1 Environment

We describe the simulation environment provided, along with the evaluation methods and bases of comparison, highlighting similarities and differences between our approach and existing ones. Inspired by DeepRM, we provide a simulation environment that models a multi-machine and multi-resource cluster. Resources, job slots and the backlog are modeled equivalently to DeepRM. Each machine has multiple types of resources. Then, tasks are arriving to a fixed number of slots, from which the scheduler agents choose candidates to allocate. Once a job has been allocated, it's moved to the afferent machine's running jobs, its resources are marked as used, and the job is eliminated from the job slots.

New jobs are arriving with a frequency according to a Bernoulli process. The resource demands and time lengths of the jobs are chosen randomly according to a pre-computed expected load/stress factor. Newly arriving jobs are placed in one of the available job slots. If none is available, the new job is added to the backlog, from which jobs are placed into the slots in a first-come first-served manner as soon as job slots free up.

Additionally, we provide not only one, but multiple distinct machines within the cluster. These machines are partitioned evenly across the agents, each agent being responsible for scheduling tasks on their machines. Handling not only one but possibly multiple machines by one agent further increases the generality of our simulator. Our experiments only consider homogeneous agents, meaning each handles the same number of machines of the same number of resources, capacities and having identical time horizons.

Further, we encapsulate the aforementioned system in an AEC environment for the MARL setup. The usage of the AEC model is pivotal in this setting, since this model disambiguates the possible conflicts arising between the agents from having a shared resource pool, the job slots.

4.2 Agents

We follow a decentralized training and decentralized execution pattern. We train a separate PPO model for each agent. In each epoch, we choose one agent and train it for one episode with the rest of the agents fixed, then repeat for a number of episodes before proceeding to the next epoch.

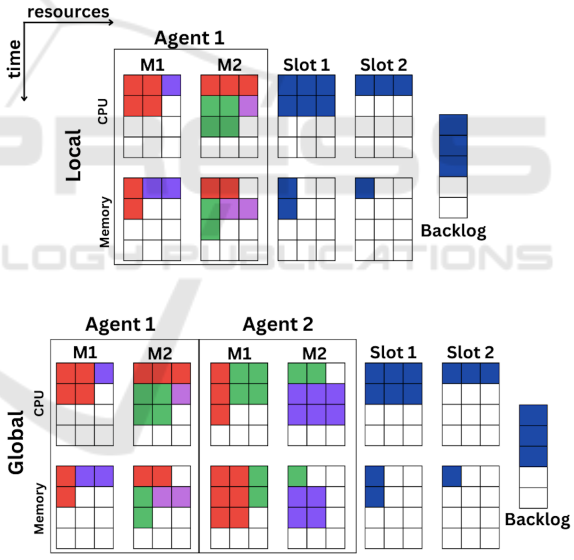

Figure 1: Sample cluster state with two resources, two pending job slots, two machines, two machines per agent, a time horizon of four discrete units and a maximum resource capacity of three units. The top row shows a possible local observation for Agent 1, while the bottom row contains the full global state. The job slot and backlog are shared between both.

The *actions* of the agents consist of numbers between 0 and NxM, where N is the number of machines per agent, and M is the size of the job slots. Having action a, the scheduler will allocate the job having index $a \mod M$ in the job slot to machine with index $a \div M$ of the scheduler. This effectively reduces the action space for agents, since a single-agent approach

would require having to deal with *NxMxK* actions, where K is the total number of agents (having *NxK* machines in the system).

Having multiple agents also has the additional possibility of reduced observation spaces. Our state representation is similar to DeepRM in that we use images to describe the current cluster state. The current state of the environment can be described by the job allocation statuses of each machine for each resource, the resource demands of the first M waiting jobs (the job slot) and the number of waiting jobs in the queue. When using local observations, each agent will only observe the state of its own machines and their job allocations, along with seeing the resource profiles of the M jobs in the job slot. It is also important to note that since the job slots and backlog are shared among all agents, this part of the observation is common to all. De-facto locality of the observations is attributable to each agent only observing its own machines. Figure 1 shows examples of global state and local observation.

4.3 Data

We simulate job arrivals according to a Bernoulli process. Jobs are generated in the following way: short, and long jobs, where short jobs have a time length between $L_{sl} = 1$ and $L_{su} = \frac{1}{2}H$, where H is the time horizon, while long jobs have a time length between $L_{ll} = \frac{2}{3}H$ and $L_{lu} = H$. Jobs are taken to be short or long with a probability p_{long}. Further, jobs have a dominant resource and a recessive one, which are chosen at random. The dominant resource demands are chosen randomly between $D_{dl} = \frac{1}{2}C$ and $D_{du=}C$, where C is the maximum resource capacity, while non-dominant resource demands are chosen between $D_{rl} = 1$ and $D_{ru}\frac{1}{2}C$. The expected job length then can be computed as follows:

$$\hat{\mathbb{E}}[Length] = (1 - p_{long})\frac{L_{su} + L_{sl}}{2} + p_{long}\frac{L_{lu} + L_{ll}}{2}$$

Analogously, in a system with two resources, the expected resource demand for resource r, $D(r)$ can be computed as:

$$\hat{\mathbb{E}}[D(r)] = p_{dom}\frac{D_{du} + D_{dl}}{2} + (1 - p_{dom})\frac{D_{ru} + D_{rl}}{2}$$

Here, p_{dom} is the probability of a resource being dominant, and the expected load on resource r on one machine then can be computed as:

$$\hat{\mathbb{E}}[Load(r)] = q^2\frac{\hat{\mathbb{E}}[D(r)]\hat{\mathbb{E}}[Length]}{B_r}$$

Here, B_r is the number of maximum available resource units for resource r, and q is the job arrival rate. Note however, that the job arrival rate is raised to the second power. This is due to the fact that whenever a job arrives, its demands and length are independently generated, and both depend on the job arrival rate. Another important detail to note is that this formula gives the expected load for one machine. To compute the total expected load on the system for a certain resource, $\hat{\mathbb{E}}[Load(r)]$ can be further divided by the total number of machines in the system. Using this formula we craft data and tweak the job arrival rate such that the expected total load varies from 10% to 120%.

4.4 Rewards

The flexibility of RL schedulers lies in the fact that they can be trained for multiple objectives. For each individual agent, we use a generic reward function that can be parameterized for several objectives:

$$R_t = \alpha\sum_{j\in A}\frac{1}{T_j} + \beta\sum_{j\in B}\frac{1}{T_j} + \gamma\sum_{j\in G}\frac{1}{T_j}$$

Here A is the set of jobs waiting in the job slot already, B is the set of jobs in the backlog and G is the set of jobs running on the machines under the current agent's management. This set can be further extended to all machines in the system, which will control the locality or globality of the rewards. Intuitively, this means that an agent will either receive penalties for the delays on its own machines, or it will be penalized for all. α, β and γ are hyperparameters, generally taken to be negative. In our experiments, we consider $\alpha = \beta = \gamma = -1$. The generality of this reward scheme allows for multiple different objectives.

5 EXPERIMENTS

Our experiments employ PPO in a decentralized multi-agent setting to the problem of task scheduling in a DeepRM-like simulator. A PyTorch-based PPO implementation is used for the agents, and the environment is provided as a Python library with an AEC interface. Our experiments are guided by the following research questions:

- **R1:** Can independent PPO agents collaboratively learn resource management solely from experience?
- **R2:** How does the locality and globality of rewards and observations influence performance?

- **R3:** How does the joint policy of multiple independent PPO agents compare to heuristics?

6 ENVIRONMENT PARAMETERS

Inspired by multiple works in the literature and the parameters used in DeepRM and DeepMAG, the parameters chosen for the experiments in our environment are summarized in Table 1.

Table 1: Environment parameters.

Parameter	Value
time horizon	20
job slot size	5
number of resources	2
resource capacity	10
backlog size	60
number of machines per agent	1,2
number of agents	2,3

6.1 Agent Parameters

Our agents use PPO to learn scheduling policies. The sizes of state and action spaces are derived from the environment parameters as described in Section 4.2. Agents are uniform in size, values of parameters and number of machines they handle. Table 2 summarizes the relevant parameters used by each PPO agent.

Table 2: PPO Agent Parameters.

Parameter	Value	Description
learning rate	$3 \cdot 10^{-3}$	
batch_size	64	
γ	0.99	discount factor
gae-λ	0.95	bias-variance tradeoff factor (Schulman et al., 2015)
clip-range	0.2	clipping parameter for the surrogate loss
entropy coeff.	0.0	Used in computing the loss
value-function coeff.	0.5	

6.2 Results

To showcase the convergence of our agents, we show the evolution of the episode mean reward as obtained through time, when co-training multiple PPO agents for job-slowdown.

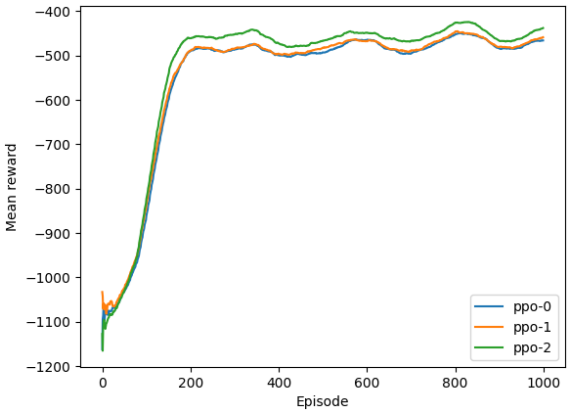

Figure 2: Mean reward obtained over time by three PPO agents, using global observations, global rewards and having one machine per agent.

Figure 2 and 3 show the difference in convergence and obtained reward when using global vs. local observations and rewards respectively. It is noticeable how considering only local information introduces a variance and slows down convergence. In contrast, incorporating global information makes agents much more homogeneous and speeds up convergence. The difference between the absolute values of rewards between global vs local runs is expected, as global reward means each agent's reward contains the penalties for all the other agents as well. This is also visible in the fluctuations of the rewards after the policies stabilised, as they are roughly aligned across all agents. The reward values on Figure 2 are not equal across agents due to the training procedure: only rewards during the learning process are included. When one agent is learning, the others are fixed for that episode and their rewards do not contribute to the learning curve. Analogously, in considering only local observations, the rewards fluctuate quite independently.

Further we explore the cumulative job slowdowns for our agents, and compare it with heuristics. We provide multi-machine generalisations of the SJF (slowest job first), packer, and Tetris agents from DeepRM. The original algorithm chooses a job to schedule according to a score. Extending these algorithms is however straightforward: the multi-machine algorithm not only chooses the best job to allocate, it also maximizes this score over the available machines.

For a fair comparison, we use the same number of heuristic agents in each case. That is, the slowdown of a system with three PPO agents is compared to the slowdown of three Packer or Random agents, for ex-

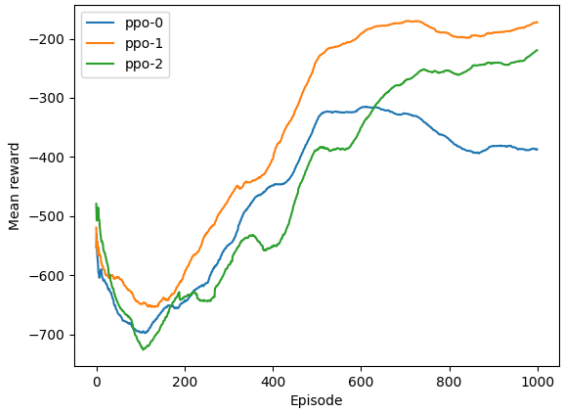

Figure 3: Running mean reward obtained over time by three PPO agents, using local observations, local rewards and having one machine per agent.

ample. We run 10 sample episodes of 200 timesteps on trained PPO agents with both local and global information, as well as heuristics, and average the slowdowns for comparison. We do this over different load factors, and the results are summarized in Figure 4. We note how there isn't a significant difference between the slowdowns of PPO agents trained with local or global information. Also, the generalisations of packer and SJF agents do not scale well to the multi-agent scenario.

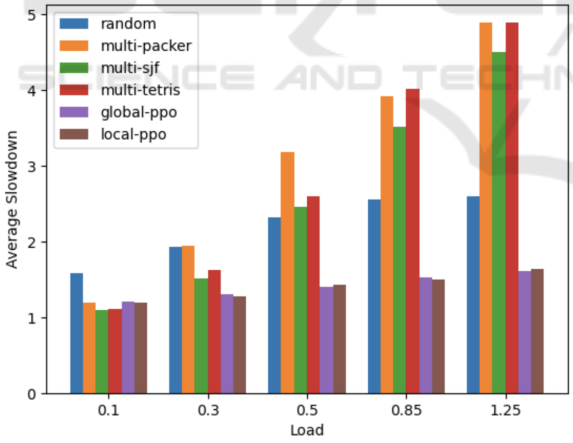

Figure 4: Average job slowdowns over load factors of different methods in an environment with 3 machines and 1 machine per agent.

7 DISCUSSION

In summary, answering **Q1** one can confidently state that independent PPO agents can successfully internalise joint scheduling policies in a multi-agent multi-machine setup that are comparable to or superseed heuristics, as measured by job slowdown. This result is particularly significant given the complexity of coordinating multiple agents and resources without predefined rules or direct communication between agents. Moreover, accessing global states and rewards by individual agents facilitates faster and more stable learning. The state and reward representations implicitly ensures that agents will not set different goals for themselves, but learn to interact productively.

In response to **Q2**, our findings reveal that the distinction between global and local observations and rewards does not significantly impact job slowdowns. This could partly be due to our experiments being conducted involving relatively small numbers of agents and machines. However, we observed a noticeable effect on the stability and speed of convergence, which varies based on whether the observations and rewards are local or global. Our experiments suggest that access to global information facilitates faster and more stable convergence, but does not imply significant performance gains. This increase in convergence speed implies that every learning update contributes to the agents' ability to learn collaboratively, and that successfully incorporating other agents's feedback proves to be useful on an individual level. In contrast, relying solely on local information seems to highlight the non-stationary aspects of the environment, impacting the speed of the learning process. In terms of performance however, both are capable of reaching comparable levels.

Moving on to **Q3**, we see how our agents superseed heuristics. This superiority is particularly evident in complex scenarios where multiple agents interact and collaborate. It is crucial to note that while heuristics provide a baseline for decision-making, they often fall short in multi-agent environments, especially as the number of agents increases.

8 CONCLUSIONS

In this paper we provided an Agent Environment Cycle game framework that successfully extends a general-purpose task scheduling simulator, able to handle multiple agents and multiple machines per agents. We also proved that in this context, Proximal Policy Optimisation is a feasible method that can be successfully applied in a multi-agent setting in order to find scheduling policies that are both plausibly good as compared to heuristics and scalable as well.

While the achieved results seem promising, our experiments are still conducted in a simulated environment, thus an obvious line of future work that we find important to outline is placing this in a practical context. One challenge of this is the assumption

of knowing resource profiles in advance, which rarely holds in practice, but tackling this is a necessity in every practical approach.

In conclusion, our findings suggest that while cluster scheduling remains a particularly difficult problem, modern MARL methods are not only effective in managing resources, they are also a step towards realizing fully autonomous and scalable cloud systems. The ability of RL agents to learn and improve over time, without human intervention or handcrafted features, aligns with the goal of developing cloud systems that can self-manage and dynamically adapt to changing conditions and requirements.

REFERENCES

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. arxiv. *arXiv preprint arXiv:1606.01540*, 10.

Dauzère-Pérès, S., Ding, J., Shen, L., and Tamssaouet, K. (2023). The flexible job shop scheduling problem: A review. *European Journal of Operational Research*.

Fan, Y., Li, B., Favorite, D., Singh, N., Childers, T., Rich, P., Allcock, W., Papka, M. E., and Lan, Z. (2022). Dras: Deep reinforcement learning for cluster scheduling in high performance computing. *IEEE Transactions on Parallel and Distributed Systems*, 33(12):4903–4917.

Lanctot, M., Lockhart, E., Lespiau, J.-B., Zambaldi, V., Upadhyay, S., Pérolat, J., Srinivasan, S., Timbers, F., Tuyls, K., Omidshafiei, S., et al. (2019). Openspiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*.

Liu, C.-L., Chang, C.-C., and Tseng, C.-J. (2020). Actor-critic deep reinforcement learning for solving job shop scheduling problems. *Ieee Access*, 8:71752–71762.

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. (2021). A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212.

OpenAI, M. A., Baker, B., Chociej, M., Józefowicz, R., McGrew, B., Pachocki, J. W., Pachocki, J., Petron, A., Plappert, M., Powell, G., et al. (2018). Learning dexterous in-hand manipulation. corr abs/1808.00177 (2018). *arXiv preprint arXiv:1808.00177*.

Paduraru, C., Patilea, C. C., and Iordache, S. (2023). Task scheduling: A reinforcement learning based approach. In *ICAART (3)*, pages 948–955.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Tassel, P., Gebser, M., and Schekotihin, K. (2021). A reinforcement learning environment for job-shop scheduling. *arXiv preprint arXiv:2104.03760*.

Terry, J., Black, B., Grammel, N., Jayakumar, M., Hari, A., Sullivan, R., Santos, L. S., Dieffendahl, C., Horsch, C., Perez-Vicente, R., et al. (2021). Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043.

Wang, M., Zhang, J., Zhang, P., Cui, L., and Zhang, G. (2022). Independent double dqn-based multi-agent reinforcement learning approach for online two-stage hybrid flow shop scheduling with batch machines. *Journal of Manufacturing Systems*, 65:694–708.

Ye, Y., Ren, X., Wang, J., Xu, L., Guo, W., Huang, W., and Tian, W. (2018). A new approach for resource scheduling with deep reinforcement learning. *arXiv preprint arXiv:1806.08122*.

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A., and Wu, Y. (2022). The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in Neural Information Processing Systems*, 35:24611–24624.

Zhang, J.-D., He, Z., Chan, W.-H., and Chow, C.-Y. (2023). Deepmag: Deep reinforcement learning with multi-agent graphs for flexible job shop scheduling. *Knowledge-Based Systems*, 259:110083.

Zhao, X. and Wu, C. (2021). Large-scale machine learning cluster scheduling via multi-agent graph reinforcement learning. *IEEE Transactions on Network and Service Management*.

Zhu, X., Xu, J., Ge, J., Wang, Y., and Xie, Z. (2023). Multi-task multi-agent reinforcement learning for real-time scheduling of a dual-resource flexible job shop with robots. *Processes*, 11(1):267.