UNIVERSITY OF CALIFORNIA
RIVERSIDE

Safety-Aware Deep Reinforcement Learning in Job Scheduling

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Science

by

Junnan Wang

September 2023

Acknowledgments

I am grateful to my advisor, without whose help, I would not have been here.

To my mother, whose unimaginable courage in the battle against cancer inspired me through this journey.

A special thanks to Eiji Aonuma and his team, who created Legend of Zelda: Breath of the Wild, the game that kept me sane.

ABSTRACT OF THE THESIS

Safety-Aware Deep Reinforcement Learning in Job Scheduling

by

Junnan Wang

Master of Science, Graduate Program in Computer Science
University of California, Riverside, September 2023
Dr. Hyoseung Kim, Chairperson

Resource allocation in computing clusters presents an NP-hard problem, with existing solutions often employing generalized heuristics that overlook job stream specificities. Past applications of machine learning to this task have shown promise, yet the lack of robustness and reliability guarantee often results in limited confidence in produced scheduling decisions. Our work introduces a novel reinforcement learning-based scheduling model, uniquely designed to incorporate job stream characteristics and with verified decision robustness. Our findings underline the potential for deep reinforcement learning to produce quality-controlled scheduling decisions, laying grounds for future research towards safe and verifiable reinforcement learning.

Contents

List of Figures

Chapter 1

Introduction

As the world becomes more connected, the demand for efficient and reliable computing clusters continues to escalate. These computing systems are integral to many sectors of our digital society, supporting services from data processing to cloud computing. Resource allocation within these clusters is a pivotal task, ensuring optimal use of computing resources and maintaining system performance. This task, however, presents a considerable challenge owing to its nature as an NP-hard problem. Existing solutions often rely on generalized heuristics, making rigid decisions that fail to account for the unique specifics of different job streams.

The incorporation of machine learning approaches into this field has indicated promise, offering a more adaptable and potentially efficient solution. However, the transition to machine learning-based solutions has been impeded by their lack of robustness and reliability, characteristics crucial to maintaining confidence in scheduling decisions. Therefore, there is a pressing need for a machine learning-based approach that not only considers job stream characteristics but also ensures decision robustness, thereby enhancing the reliability of the scheduling decisions made.

This study is an endeavor to address this gap in the research landscape. We introduce a reinforcement learning-based scheduling model, designed specifically to incorporate the unique characteristics of job streams. By leveraging the strengths of reinforcement learning – its ability to learn optimal behaviours through trial and error and adapt to changing environments – our model offers a potential solution to the reliability issue.

Our approach capitalizes on the ability of reinforcement learning to navigate complex decision spaces effectively, thereby enabling the creation of scheduling decisions that are both quality-controlled and cognizant of job stream specificities.

This research work is divided into sections detailing the background of the study, the design and implementation of our reinforcement learning-based model and its verification, a thorough analysis of our results, and an exploration of future work. By the end of this paper, we aim to have demonstrated the potential of deep reinforcement learning in producing quality-controlled scheduling decisions, thus paving the way for future research in this promising field.

Chapter 2

Background and Related Works

The application of reinforcement learning (RL) in complex decision-making tasks has seen significant progress in recent years. Its notable successes include mastery of computer games, where agents navigate intricate state spaces, a feature relevant to resource allocation tasks [24].

One of the groundbreaking works in applying RL to resource management is the DeepRM model [17]. DeepRM introduced a novel approach by representing resource consumption in pixelized state images and employing a single RL agent to process these images and produce scheduling decisions for dynamically arriving job streams. It evaluates scheduling decisions based on average job slowdown, demonstrating a marked improvement in performance compared to traditional scheduling methods such as shortest job first (SJF) and Tetris [10].

Inspired by DeepRM's approach, we constructed the state space to be compatible with multi-processor environment. When designing our RL agent, instead of using a naive single fully connected hidden layer, we experimented with different network structure as well as training algorithms.

The first architecture we adopted was the classical design deep Q-network(DQN)[5]. DQN's success in handling high dimensional input makes it a great candidate for resource management. For instance, successful application of DQN is deemed valuable in industrial production management[25]

The second approach we utilized was Proximal Policy Optimization(PPO)[15]. Policy-Gradient algorithms maintain policy updates within a proximate range in the parameter space. However, even minor variations in the parameter space can greatly impact and worsen the policy performance [22]. PPO aims to take the largest improvement step that doesn't cause a performance breakdown[23], which makes it a promising method.

We also want to ensure in the quality of our DRL models. Safety, liveness and fairness are general classes that cover majority of the model properties[3]. Our study focuses on the safety class, where no unfavorable events occur within the system. Different approach for safe RL has been developed throughout the years[7]. One approach is to adjust the optimization criterion. This can be done by introducing heavy penalty on the uncertainty in the model[11], or introduce a dynamic risk factor into the reward calculation[19], or it can adding conditions that limit the growth of the reward in a constrained region[2]. The second approach is limiting the exploration process within the safe scope. This approach can be implemented by incorporating external knowledge into the training process[6][1][4][21], or it can be realized by carefully quantifying the system risk, and use it to guide the learning process of the agent[9]. Our method falls under the Risk-directed exploration criteria, in which the action choice during exploration is guided by risk. Our work tries to enforce a hard constraint during the RL learning process.

On top of ensuring constraint during the learning phase, we also attempts to verify the reliability of the trained model. The main idea we used is known as bounded model check in formal verification[3]. The model checking problem can be solved quite easily using standard search algorithms if the components of the problem are well defined [3]. For instance, in a finite transition system, it is possible to run a depth first search (DFS) to determine if there exist a reachable bad state. Obviously the model we have at hand is not quite as naive, yet the bounded model check is still able to provide a limited qualitative assessment of a Reinforcement Learning model with infinite transitions. We utilized the Marabou framework as the back-end engine of verification[14]. Marabou is an SMT-based tool that can answer queries about a network's properties by transforming these queries into constraint satisfaction problems. It out performs its predecessors[16][8][12] in terms of run time and accuracy, providing a efficient way to verify deep neural networks which suits our need in this study.

Chapter 3

Proposed Method

3.1 Problem Formulation:

This work focuses on the identical machine scheduling problem, where a stream of independent jobs j are scheduled on a cluster of machines m. The goal of our scheduler is to ensure i) minimum average delay of all jobs ii) adequate utilization rate of system resources iii) minimal number of missed deadlines.

3.2 Model Design:

$$j_i = \{jid, cost, t_{arr}, d, F\}$$

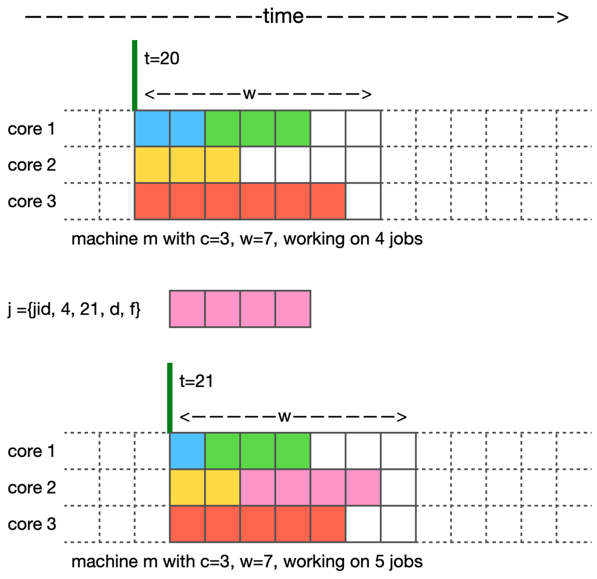
Each incoming job has a unique job id (jid) requires a fixed number of CPU time share ($cost$). It is dispatched at a timestamp t_{arr} and is expected to complete before d units of timestep after arrival. A flag F is used to denote whether the job has been assigned to a machine, and the default value is false f. A job is handled by a single machine m_j for the entire execution, and its *cost* is identical regardless of the machine it is assigned to.

$$m_j = \{mid, c, w\}$$

Each machine in the cluster has a unique id (mid). It has c identical CPU cores, and a scheduling window of length w. The scheduling window denotes the machine's view of CPU resource usage from current timestamp onward. For instance, a machine with an empty load at time T has c available CPU cores in the time window $[T, T + w]$.

$$P_T = \{j_i | t_{arr} < T, F = f\}$$

At each discrete timestamp T, the incoming jobs arrive and are put in the job pool P. The scheduler picks from the job pool and assign jobs to available machines. A machine is available for scheduling if the current job can fit in its resource window. If no such machine exists, the scheduler does nothing and proceed to the next timestamp. The figure below shows an example scheduling system with one machine in action.



------------time------------->
t=20
<------w------->
core 1
core 2
core 3
machine m with c=3, w=7, working on 4 jobs
j ={jid, 4, 21, d, f}
t=21
<------w------->
core 1
core 2
core 3
machine m with c=3, w=7, working on 5 jobs

Figure 3.1: Progression of Scheduling window from t to $t + 1$

3.2.1 State Space

We construct a finite state space for our scheduling system consists using two main parts. First is the resource consumption matrix of all machines, and second is the job pool. The first part is rather straightforward. Each machine has a resource consumption matrix of dimension $c_i \times w$ as demonstrated in figure 4.3, hence we can easily construct a system-wide resource consumption matrix of size $\sum_{i=1}^{m} c_i \times w$. However to represent the job pool is more tricky. The number of jobs in pool can be arbitrarily large, yet we cannot have a state space subjected to infinite growth. Our solution is to use a job queue Q with fixed size $q \times w$ to represent a subset of the job pool, and we record the rest number of jobs in the job pool using a single row. An example of the job pool representation is shown in the figure below.

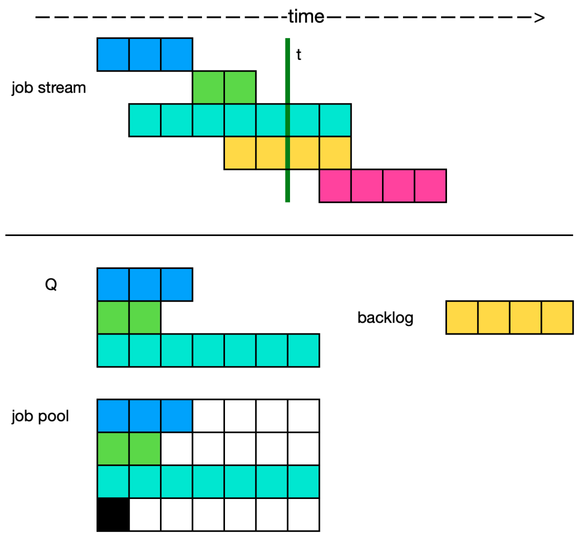


Figure 3.2: Components of job pool for a job stream at time t, where $q = 3, w = 7$

In the end, the state space has dimension of $(\sum_{i=1}^{m} c_i + q + 1) \times w$.

3.2.2 Action Space

The action space for the scheduling system is relatively compact. The action the scheduling system takes is to repeatedly pull job in queue to assign to a finite number of machines, and to stop and proceed in time when no machine is available. Hence we can describe action a as following:

$$a(j_i \in Q) = \begin{cases} m_j & \text{if } j_i \text{ can be scheduled on some machine } m_j \\ null \end{cases} \tag{3.1}$$

Because for each job in queue there are $m + 1$ possible actions, the size of action spaces $m + 1$.

3.2.3 Reward Function

The reward is designed to ensure machine usage and minimize the total slow down.

$$Reward = \alpha \sum_{j_i \in Q} (d_i - T_i - c_i) + \beta \sum_{j_k \notin Q, j_k \in P} (d_k - T_k - c_k) + \gamma \cdot \mu \tag{3.2}$$

Here,α, β, γ are positive weights adjusting the emphasis of the reward. T_i represents the current wait time of job j_i. Hence the earliest possible completion time for job j_i $T_i + c_i$, c_i being the CPU time needed. If it is not possible to complete job by deadline d_i the reward becomes negative acting as penalty. The reward encourages efficient usage of system resources by introducing system utilization rate μ, which is calculated by the sum of resource utilization rate in the scheduling window.

3.3 Learning Method

3.3.1 Feature Extraction

In order to make the model more robust[26], we prepossessed the state matrix using a convolutional neural network. The extracted feature signal is then passed into the DRL models.

3.3.2 Safety Guard

We define the scheduler to be unsafe if a job in the system were to miss its deadline. Although the reward function3.2.3 push the DRL models towards this goal, there is a possibility that at some time the DRL will recommend a scheduling decision with high reward yet leading to a missed deadline. Hence we introduce a safety guard in both of the models. The safety guard performs checks if the current action that DRL decides to take will lead the system into a bad state (missed deadline). The result of this safety check will be utilized by the DRL to modify the action decision. The pseudo code of the safe guard is shown in algorithm 1.

Algorithm 1 SafeGuard

```
Input: action $a$, system state $S$
Output: boolean value indicating whether current action is safe
1: compute the next state $S' \leftarrow < a, S >$
2: for all jobs $j$ in $S'$ do
3:     if $j$ misses deadline then
4:         return False
5:     end if
6: end for
7: return True
```

3.3.3 DQN With Experience Replay

In our deployment of DQN, we employ two neural networks: the online network and the target network. Although their structures are identical, they function with distinct parameters. The online network utilizes the most recent parameters, which are updated in accordance with the loss function. Conversely, the target network is updated using experience replay, where parameters are systematically transferred from the online network after a predetermined N steps. Through gradient descent, DQN is capable of limiting the loss within a specified boundary. This ensures that both the value function and gradient values stay within a normal range, thereby promoting the stability of the algorithm.

Experience replay is implemented by storing the agent's experiences, represented by $e_t = (s_t, a_t, r_t, s_{t+1})$, at every time-step t in a replay memory $D_t = (e_1, \ldots, e_t)$. Throughout the training process, we randomly select empirical data from this replay memory D to update the network's parameters via stochastic gradient descent. This particular mechanism of experience replay assists in breaking data correlations, thereby enhancing the efficiency of neural network updates. The algorithm operates as follows:

Algorithm 2 Training DQN with Experience Replay

```
Input: system state $S$
1:  Initialize replay memory $D$ to size $N$
2:  Initialize action-value function $Q$ with random weights $\theta$
3:  Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$
4:  for episode = 1, $M$ do
5:      Initialize system state $S_1$
6:      for t = 1, $T$ do
7:          initialize a safe action set $A$
8:          for all possible action $a$ do
9:              if $SafeGuard(a, S_t) = True$ then
10:                 add $a$ to $A$
11:             end if
12:         end for
```

```
13:         if $A$ is empty then
14:             add all possible actions into $A$
15:         end if
16:         With probability $\epsilon$ select a random scheduling action $a_t$ from $A$
17:         otherwise select $a_t = \arg\max_a Q(S_t, a; \theta); a \in A$
18:         Execute scheduling action $a_t$ in system and observe reward $r_t$ and new state $S_{t+1}$
19:         Store transition $(S_t, a_t, r_t, S_{t+1})$ in $D$
20:         Sample random mini-batch from $D$: $(S_j, a_j, r_j, S_{j+1})$
21:         Set $y_j = r_j$ for terminal $S_{j+1}$
22:         Set $y_j = r_j + \gamma \max_{a'} \hat{Q}(S_{j+1}, a'; \theta^-)$ for non-terminal $S_{j+1}$
23:         Perform a gradient step on $(y_j - Q(S_j, a_j; \theta))^2$ with respect to the network parameters $\theta$
24:         Every $C$ steps reset $\hat{Q} = Q$
25:     end for
26: end for
```

After training is completed, the usage of the trained DQN model is quite straight forward. It simply loads parameters from the trained model, calculate the action value for all actions that are checked to be safe, and return the action with maximum reward.

3.3.4 Proximal Policy Optimization(PPO)

The algorithm based on PPO adopts the Actor-Critic framework, alternating between gathering data through interacting with the environment, and optimizing a surrogate objective function using stochastic gradient ascent. The fundamental principle of PPO is to confine the update span of the previous and new policies to enhance stability. The algorithm works as following:

Algorithm 3 Proximal Policy Optimization for Scheduling

```
 1: Initialize policy parameters $\theta$, old policy parameters $\theta_{old}$, and value function $V(s;\phi)$
 2: for iteration = 1, $M$ do
 3:     Collect set of trajectories $D = \{s, a, r\}$ by running policy $\pi_{\theta_{old}}$
 4:     Compute rewards-to-go $R_t$
 5:     Compute advantage estimates $A_t = R_t - V(s_t;\phi)$
 6:     for epoch = 1, $N$ do
 7:         Update policy by maximizing $PPO - Clip$ objective:
 8:         $\theta \leftarrow \theta + \alpha\nabla_\theta \frac{1}{|D|}\sum_{t\in D} \min\left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)$
 9:         where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$
10:         Update value function by fitting to new data:
11:         $\phi \leftarrow \phi - \alpha\nabla_\phi \frac{1}{|D|}\sum_{t\in D}(V(s_t;\phi) - R_t)^2$
12:     end for
13:     $\theta_{old} \leftarrow \theta$
14: end for
```

3.4 Verification

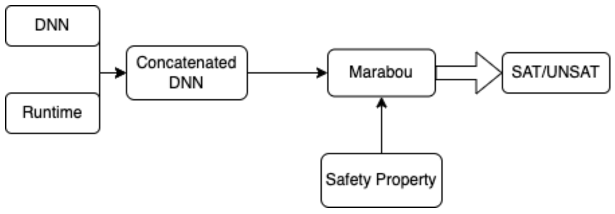
In order to verify the safety properties of the trained scheduler, we will need to check if a bad state is reachable throughout the execution of our scheduler. Note that for each scheduling decision, there are $m+1$ possible actions. If proceeding with a naive search, we will be search in a tree with fanning factor of at least $m+1$. The term tree here is used rather loosely, as the possibility of repeating state throughout execution is valid. The size of tree grows exponentially with the depth, which will render the naive search meaningless very quickly in the process of verification.

Hence we simplify the problem with some constraints. Instead of trying to perform a general verification on a model with infinite transitions, we perform a iterative bounded model check. More specifically, we try to solve the query $< M, p, t >$ that given a trained DNN N and a property p, is N guaranteed to satisfy p in some given runtime t? Through out the process we gradually relax l to push towards a more general assessment.

Now, because Marabou supports the verification query with $< N, p >$, we need to encode the perform the transformation $< N, p, t > \longrightarrow < N', p >$, or rather

$$< N, t > \longrightarrow N' \tag{3.3}$$

This transformation is done by concatenating t duplication of N to formulate a larger DNN that can encode t successive states. Note that the input is also concatenated so that the input stream for N within period of t becomes a single input vector of size t for the DNN N'. The figure below demonstrates the workflow of the verfication process.



Chapter 4

Experimental Results

4.1 Experiment Setup

Predominant techniques in reinforcement learning today comprise value-based and policy-based algorithms. In our study, we first opt for and construct two representative algorithms, namely DQN and PPO. Subsequently, we contrast their performance within an identical experimental setting, maintaining the same configuration for each experiment. All experiments below were performed in single thread on 1.1 GHz Quad-Core Intel Core i5 CPU with 16GB of memory.

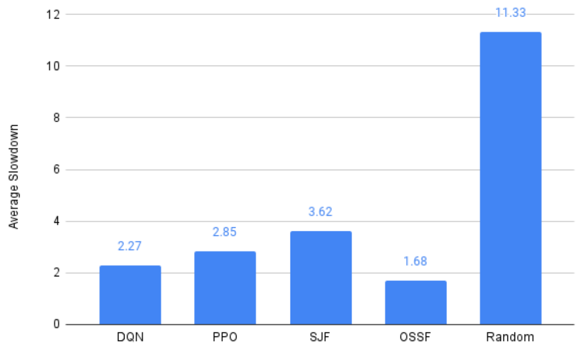
4.2 Scheduling Efficiency Evaluation

For scheduling efficiency evaluation, we utilized a workload where jobs arrive in a Bernoulli process. Jobs are classified into two types. 90% of the jobs are *small*, which takes 1 to 5 CPU times and has deadline of 5 to 10 unit times after dispatch. The rest 10%

are *large* jobs that take 15 to 20 CPU times, and has deadline of 30 to 50 unit times after dispatch. The mean interval of arrival is set such that the average load ranges from 50% to 150% of the overall system capacity. We simulated the job stream for 8000 units of time. The first 6000 units are used for training, and the last 2000 units are used for testing.

Baseline: we utilized three baseline scheduling system to evaluate our model. The first one is the classical scheduling algorithm shortest job first (SJF). The second is an optimal scheduling solution finder(OSSF) we built using Google's OR-Tools[20]. The third one is a random scheduler that assigns jobs to random machines.

Figure 4.1: Average slow down of different scheduling algorithms

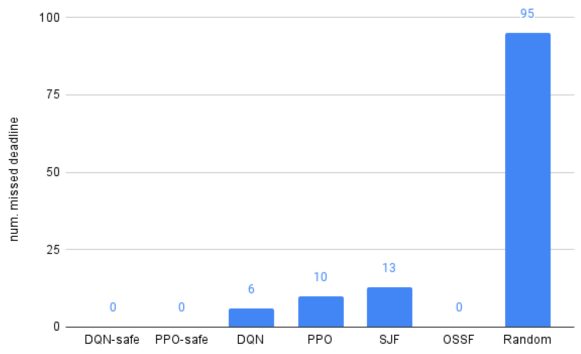


We compared the performance of the schedulers based on average job slow down, which is what our models are trained on. We can see from the figure above that Both DRL models outperform SJF, with DQN having a greater advantage and coming close to the optimal solution.

4.3 Safety Evaluation

Here we compared the number of missed deadlines using the same experiment setup. To demonstrate the improvement in safety, we trained schedulers using the same DRL models without the safeguard.

Figure 4.2: Number of missed deadlines for different scheduling algorithms



We can see that the safeguard noticeably reduced the number of missing deadlines for moth DQN and PPO methods, and as expected, outperforms SJB and random as well. We can hence conclude that in this scenario, our DRL scheduling model is safe.

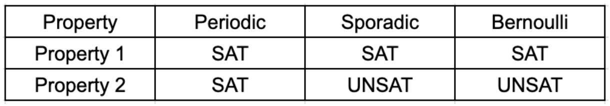
4.4 Verification Results

We verified the trained scheduler with the following properties:

- **Property 1:** When the system has an empty load and there are jobs in the job pool, the scheduler should not wait.
- **Property 2:** When the system is at full capacity and there are jobs in the job pool, the scheduler should wait.

For both properties, we performed bounded model check with $t = w$, the scheduling window size. Three different type of job stream is used in this experiment: periodic, sporadic, and Bernoulli. For each job streams, the scheduler is trained separately from beginning and the trained model is then passed into the verifyer. The verifyer found that property 1 holds regardless of the nature of the job stream. However, property 2 only holds when the job at hand is period. The different learning methods showed no difference in results. The results are shown in the figure below.

Figure 4.3: verification results on different job streams



Property	Periodic	Sporadic	Bernoulli
Property 1	SAT	SAT	SAT
Property 2	SAT	UNSAT	UNSAT

Chapter 5

Discussion and Future works

This study is built on several assumptions. First we assume the specifications of jobs are known at arrival time. However, in reality this is rarely true. The processing time required for a job is often subjected to change in the system. It is possible to use extra padding on the expected duration to give more leniency for the job to complete. Yet such overestimate may undermine the accuracy of the state representation. How will a less-accurate state representation affect the RL model is a problem worth investigating.

Our scheduling system also relies on certain characteristic of each job. It requires prior knowledge of the maximium job duration time. To be more specific, the maximum number of CPU time slices a job can require must be less than the length of the scheduling window w (3.2). Although we can set w to be large for the scheduler to accommodate most scenarios, this is a rather brute force solution. A large w can lead to a much larger state representation which makes the training of DRL models more costly.

The design and usage of the verifyer also have much room for improvement. Using concatenated neural networks to represent the consecutive states lacks scalability. The verification will become expensive when we try to explore the safety property in a longer period of time. Also, the property we used to verify the scheduler is rather relaxed. In fact, the lack of granularity ties back to lack of scalability. If we were to use a strict, specific constraint indicating some incident is bad, the total runtime required to sufficiently check for this constraint may become astronomical.

One other limitation lies in the problem statement itself. The scheduling system is built on the assumption that all jobs are independent, yet in real-world scenarios, dependency is a common property in job stream. The current job abstraction and state representation are unable to encode dependency among jobs. These designs, as well as the implementation of the safe guard, will need to be modified in order to put RL-based scheduler into a more generalized use.

In summary, the future directions of this work is to redesign the scheduler to be less reliant on prior knowledge of the job stream, to be able to handle tasks with greater level of complexity, and to create a scalable design for the verification of RL models.

Chapter 6

Conclusions

Our research shows that the DRL methods can serve as a comparable and above substitute to traditional heuristics. And our effort in safety-aware DRL design demonstrate the potential of safe Reinforcement Learning while preserving adequate reward.

Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *Int. J. Rob. Res.*, 29(13):1608–1639, nov 2010.

[2] Eitan Altman. Asymptotic properties of constrained Markov decision processes. Research Report RR-1598, INRIA, 1992.

[3] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking.* MIT Press, 2014.

[4] Jeffery A. Clouse and Paul E. Utgoff. A teaching method for reinforcement learning. In *Proceedings of the Ninth International Workshop on Machine Learning*, ML '92, page 92–110, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[5] Mnih V;Kavukcuoglu K;Silver D;Rusu AA;Veness J;Bellemare MG;Graves A;Riedmiller M;Fidjeland AK;Ostrovski G;Petersen S;Beattie C;Sadik A;Antonoglou I;King H;Kumaran D;Wierstra D;Legg S;Hassabis D;. Human-level control through deep reinforcement learning.

[6] Kurt Driessens and Saso Dzeroski. Integrating guidance into relational reinforcement learning, 2004-12-01.

[7] Javier García, Fern, and o Fernández. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16(42):1437–1480, 2015.

[8] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2018.

[9] Clement Gehring and Doina Precup. Smart exploration in reinforcement learning using absolute temporal difference errors. In *Proceedings of the 2013 International Conference on Autonomous Agents and Multi-Agent Systems*, AAMAS '13, page 1037–1044, Richland, SC, 2013. International Foundation for Autonomous Agents and Multiagent Systems.

[10] Robert Grandl, Ganesh Ananthanarayanan, Srikanth Kandula, Sriram Rao, and Aditya Akella. Multi-resource packing for cluster schedulers. In *ACM SIGCOMM*, August 2014.

[11] Matthias Heger. Consideration of risk in reinforcement learning. In *International Conference on Machine Learning*, 1994.

[12] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks, 2017.

[13] Guy Katz, Derek Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, David Dill, Mykel Kochenderfer, and Clark Barrett. *The Marabou Framework for Verification and Analysis of Deep Neural Networks*, pages 443–452. 07 2019.

[14] Guy Katz, Derek A. Huang, Duligur Ibeling, Kyle Julian, Christopher Lazarus, Rachel Lim, Parth Shah, Shantanu Thakoor, Haoze Wu, Aleksandar Zeljić, and et al. The marabou framework for verification and analysis of deep neural networks, Jul 2019.

[15] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[17] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, page 50–56, New York, NY, USA, 2016. Association for Computing Machinery.

[18] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[19] Teodor Moldovan and Pieter Abbeel. Risk aversion in markov decision processes via near optimal chernoff bounds. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[20] Laurent Perron and Vincent Furnon. Or-tools.

[21] M.T. Rosenstein and A.G. Barto. Reinforcement learning with supervision by a stable controller. In *Proceedings of the 2004 American Control Conference*, volume 5, pages 4517–4522 vol.5, 2004.

[22] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[24] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games, 2019.

[25] Bernd Waschneck, André Reichstaller, Lenz Belzner, Thomas Altenmüller, Thomas Bauernhansl, Alexander Knapp, and Andreas Kyek. Deep reinforcement learning for semiconductor production scheduling. In *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*, pages 301–306, 2018.

[26] Thomas Wiatowski and Helmut Bölcskei. A mathematical theory of deep convolutional neural networks for feature extraction, 2017.