International Journal of Advances in Computer Science & Its Applications – IJCSIA

Volume 9 : Issue 2- [ISSN : 2250-3765] - Publication Date: 27 Dec, 2019

Multiple Resource Management and Burst Time Prediction using Deep Reinforcement Learning

Vaibhav Kumar, Siddhant Bhambri, Prashant Giridhar Shambharkar
Delhi Technological University
India

***Abstract*—Resource management and job scheduling are two problems that go hand-in-hand and the solutions to which are primarily dependent on the nature of workload. With increasing demand to automate the entire process from allocating resources to scheduling jobs efficiently, deep reinforcement learning techniques have been brought into the picture which adapt to the environment and learn from experience. In this paper, we present SchedQRM which classifies burst time of jobs based on their signature and employs Deep Q-Network algorithm to find an optimal solution for any arbitrary job set. We also evaluate our proposed work against state-of-the-art heuristics to show the efficacy of our approach.**

***Keywords*— reinforcement learning, job scheduling , Deep-Q Network**

Introduction

Resource management has always been a tricky domain in the field of research and has become increasingly significant due to the rapid developments in production technologies. The problem of optimal allocation and use of resources has been dealt in the past in several ways [1-5]. Improvement in the classical measures of efficiency due to periodic rescheduling has already been addressed in the past alongside the undesirable effect of compromising stability.

Considering the heuristics on which Reinforcement Learning (RL) algorithms work, we believe that RL approaches and methodologies fit well in the domain of resource management and job scheduling since they shall allow the machine to check for the best possible order of scheduling for a set of jobs, given the resource and burst time requirements for each job.

Our approach is an extension to the idea of machines being able to handle resources on their own in a justified manner. RL has gained attention in the field of machine learning research. The concept of decision-making had been introduced earlier in the research problems of resource management and job scheduling. RL revolves around agents which interact with the environment to accomplish a task. For each action it takes during its discourse through the environment, the agent receives a reward -positive or negative, based on the result of the action it takes. The agent has no prior knowledge of the task to be performed and learns based on the reward it receives.

We design and evaluate SchedQRM, an online multi-resource job scheduler, in our approach to applying RL for solving the problem of resource management. A set of jobs are fed into the scheduler as an input along with their job signature, and no pre-emption is allowed. The scheduler aims to optimize average job slowdown or job completion time by minimizing it.

Related Work

Resource allocation problem has been addressed in various contexts such as in Radio Networks [1], Software Defined Networking [2], mobile cloud computing networks [3] and in wireless communication networks [4]. Wan et al. [5] in their paper propose a resource allocation algorithm to maximize throughput for hybrid Visible Light Communication (VLC) and Wi-Fi networks. In all these papers, we observe that increase in the throughput, whatever the requirement may be, has been the primary objective.

The objectives while scheduling jobs/tasks at hand vary in context from minimizing CPU energy [6] to reducing total completion time on a single machine [7]. With the advent of Big Data and Machine Learning approaches, Karim H. and Ahmed J. in [8] proposed an approach for scheduling tasks in Big Data Cluster and showed a comparison with the traditional task schedulers such as the First-In-First-

International Journal of Advances in Computer Science & Its Applications – IJCSIA
Copyright © Institute of Research Engineers and Doctors
Volume 9 : Issue 2- [ISSN : 2250-3765] - Publication Date: 27 Dec, 2019

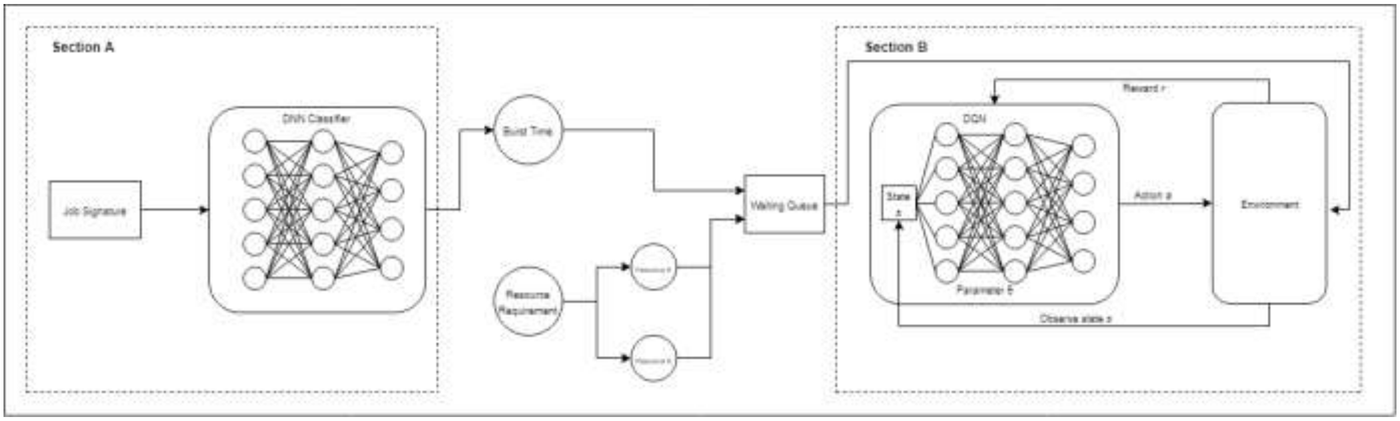

Figure 1: Flow of data in our proposed model.

Out (FIFO) scheduler and the Capacity Scheduler. Fuming et al. in [9] proposed the concept of a virtual scheduling pool whereas Zhao P. and Huang T. in [10] incorporated a genetic algorithm to solve the problem of single resource dynamic Job-Shop scheduling. Smart manufacturing domain also faces with a real-time requirement of job scheduling which has been tackled using a hybrid computing framework [11].

Resource Isolation Policy (RIP) combined with static as well as dynamic scheduling strategy was proposed by Liu et al. [12] to solve the problem of hard real-time task deadline. In [13], the authors characterize the performance of scheduling policies for wireless systems that are based on Cumulative Density Functions (CDF). Su N. et al. [14] incorporated genetic programming to propose an automatic design of scheduling policies which shows outstanding performance on unseen simulation scenarios. Particle Swarm Optimization (PSO) [15] algorithm has also been used for optimizing task scheduling in the field of cloud computing.

Survey on the past work is done which shows the use of reinforcement learning approach to designing feedback controllers for discrete as well as dynamic systems [16]. In [17], the authors proposed an adaptive Neural Net-based controller using RL for a class of nonlinear systems which does not require information about the system dynamics. RL has also been used to solve the problem of resource allocation [18] where the authors combine the strengths of RL and queuing models in a hybrid approach. However, our inspiration has been from the work of Hongzi M. et al. in [19] where the state space has been represented pictorially and fed into a deep reinforcement learning network to find the optimal scheduling policy for a given job-set.

Background

This section discusses the techniques in brief that we have worked upon in this paper.

Burst Time Classification: Prior knowledge of the burst time of a job helps exceptionally well in resource allocation and job scheduling. In a few cases, the burst time (run time) of a job is known, but mostly an approximation needs to be made. We divide the burst time of every job into a certain number of classes based on the job environment. Every job has a signature/set of attributes. These are fed to a neural net classifier to classify jobs and approximate the burst time.

Reinforcement Learning: Consider a scenario where there is an *agent* which interacts with an *environment*. The *agent* observes a state s_t and chooses an action a_t at each time step t, from the set of possible actions. Once the action is taken, a state transitions takes place from s_t to s_{t+1}, following which a reward r_t is given to the *agent*. The state transitions and rewards are assumed to have the Markov property; i.e., the action to be taken by the *agent* in the current state s_t is independent of the states that preceded s_t.

Note that the *agent* has no prior knowledge of which state of the *environment* would it transition to or what reward it may receive, once it chooses to take action a_t. It is while interacting with the

Volume 9 : Issue 2- [ISSN : 2250-3765] - Publication Date: 27 Dec, 2019

environment, during training, that the *agent* will be able to observe the value of these quantities. The expected cumulative discounted reward: $E[\sum_{t=0}^{\infty} \gamma^t r_t]$, where $\gamma \in (0,1]$ is the discount factor for future rewards, needs to be maximized through learning.

Deep Q-Network (DQN): DQN is a type of Temporal Difference (TD) learning method. With the use of TD learning methods, the estimate of the final reward calculated at every step for each state can be formally expressed as:

$$\boldsymbol{V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]}. \quad (1)$$

Where, $V(s_t)$ represents the utility value of state s_t, $\boldsymbol{\alpha}$ is the learning rate, $\boldsymbol{\gamma}$ is the discount factor and $\boldsymbol{r_{t+1}}$ is the observed reward at time $\boldsymbol{t+1}$. Compared to Monte Carlo methods [20], where the Q values are updated after the end of an episode; here, the Q values are updated after ever action. This helps in guiding the *agent* to the goal state more efficiently. The *agent* uses DQN because of the advantages that it offers in solving the scheduling problem through Experience Replay. Experience Replay is a circular queue which stores agent's experiences in form a tupple $e_t=(s_t, a_t, r_t, s_{t+1})$. Here, the *agent* takes an action a_t to move from state s_t to s_{t+1} and gains a reward r_t. This helps reducing correlation between transitions when the neural network has to be updated. The learning speed of the model increases with mini-batches that are made by DQN to update the neural network being employed. It also reuses past transitions to avoid catastrophic forgetting which speeds up learning and also breaks undesirable temporal correlations. Thus, DQN is believed to achieve stable training.

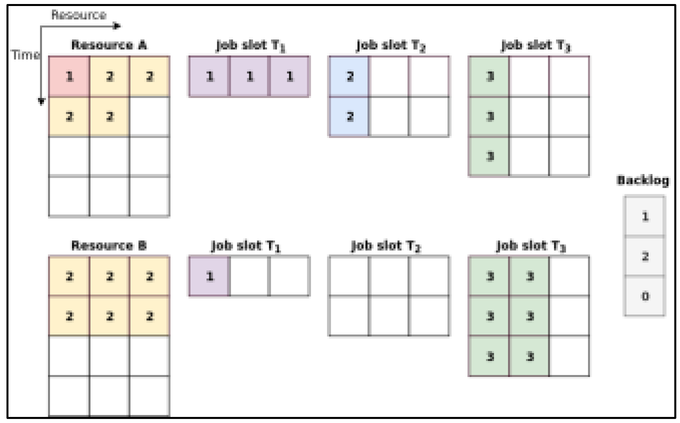

Figure 2: Pictorial representation of a sample state in SchedQRM.

Design

In this section, we present the design of SchedQRM. We describe the problem and also its formulation as an RL task. We then explain our solution to this problem based on the techniques described in the previous section.

Model

As shown in Figure 1, we divide our model into 2 sections- Section A and Section B.

Section A takes job signature as an input and predicts the burst time for the job using a Deep Neural Network (DNN). Resource requirements along with the burst time for the job are sent as an input to the waiting queue.

Section B consists of the DQN model which outputs a scheduling policy for a given job set. All the jobs from the waiting queue are fed into the *environment* as a starting state.

We consider a cluster with k resource types (e.g. CPU, I/O, memory) and it is treated as a single collection of resources. Jobs arrive at the cluster in discrete timestamps. One or more of the waiting jobs are chosen to be scheduled by the scheduler at each timestamp. The resource requirement of each job j is given by the vector $\boldsymbol{r_j = (r_1, r_2, \ldots\ldots, r_k)}$, where r_i represents the number of instances of resource i required by the job and $\boldsymbol{i \in [0, k]}$. We define T_j as the duration/execution time of the job. Given the above information, the DNN correctly places each job into a class from the set $\boldsymbol{(c_1, c_2, \ldots\ldots, c_n)}$, where c_i represents a class of jobs that requires resources for i timestamps. The jobs are assumed to be non-preemptive for the sake of simplicity. Also, r_j must be allocated to the job j continuously from the time that the job starts execution until completion.

The simplicity of the model can allow it to be used in other domains of application where a similar type of input is provided containing a set of jobs along with their resource requirements. Given this information, our design of the model will identify the execution time requirement of the job by learning from the experience. This information shall further be passed to the DQN *agent* which will schedule these jobs.

TABLE I: INPUT DATASET TO THE NEURAL NET CLASSIFIER

	interp	gnu.hash	dynsym	dynstr	rela.dyn	rela.pit	pit	text	rodata	eh_frame_hdr	eh_frame	got	got.pit	bss	symtab	strtab	filesize	time
0	28	0	5	2	0	4	1	645	32064	1	0	0	1	0	1	764	49768	1.0
1	28	0	5	2	0	4	1	645	32072	1	0	0	1	0	1	764	49768	1.0
2	28	0	5	2	0	4	1	645	32080	1	0	0	1	0	1	764	49768	1.0

Objective: Similar to the prior work shown in [19], we use the *average job slowdown* as the primary objective for our agent. Formally, the slowdown for each job i is given by $\boldsymbol{S_j = C_j/T_j}$, where C_j is the completion time and T_j is the burst time of the job. Completion time of a job is the time between arrival and completion of execution; note that $\boldsymbol{S_j \geq 1}$. If the completion time of the job is not normalized by the job's duration, the solution will be biased towards large jobs.

Burst Time Classificaion

Dataset creation: To create the dataset for burst time prediction, we use C++ object files for four programs to generate job signatures, namely: Matrix multiplication, Quicksort, Fibonacci series generator, and a random number generator. A script is deployed to create Executable and Linkable Format (ELF) files with various input sizes, and it collects the job signatures through the *readelf* bash command. We run this script until 100,000 data points are created. File size and run time are also stored for every ELF file along with the signature. Table I represents 3 out of the total data points/rows of the data-set that are fed into the neural network classifier for training. This dataset was created on an Intel core i7-6700 quad-core, 64-bit x86 processor and 8+8GB DDR4 3000mhz Corsair RAM.

RL formation

State Space: We represent a pictorial representation of a single state of the system as shown in Figure 2. This state contains information regarding the current resources' allocation, the jobs in the waiting queue and the jobs present in the *backlog*. The left-most clusters represent the allocation of resource instances to jobs which have been scheduled for service as of the current timestep. Here we have assumed two types of resources with three instances available for each of them. The resource allocation shown is present starting from the first timestamp till t timestamps, each row representing a timestamp. Jobs in the waiting queue belong to one of the time slots belonging to the vector $\boldsymbol{T_i = (T_1, T_2, \ldots\ldots, T_n)}$, where i refers to the number of timestamps required by a job to complete its execution; for example, the job in time slot T_2 requires one instance of resource A and zero instances of resource B for two timestamps. The different numbers within the resource clusters represent different jobs belonging to the respective time slots that have already been assigned resources and are undergoing or are about to begin execution; for example, 2 represents that a job belonging to T_2 has currently been assigned two instances of Resource A and three instances of Resource B for two timestamps. The first job of a certain burst time is represented in it's appropriate job slot while others wait for their turn in the *backlog*. The t^{th} box in the backlog stores the number of jobs with burst time t. For example, in Figure 2, there is one job of burst time 1 and two jobs of burst time 2 in the *backlog*.

Our state representation is a modified version of the state representation shown in [19], unlike which, we have a fixed representation of jobs based on their burst time. This fixed representation allows us to represent a state as an array of numbers and obviates the need for pictorial representation of a state and involvement of lofty convolutional neural networks. Hence, our input to the model is a flattened array representation of Figure 2.

Note: By having a fixed time representation, only a single job of a particular burst time can be represented in a state. Multiple jobs of same burst time have to wait in the waiting queue which might hinder the learning of the *agent*. However, it helps significantly in an optimized representation of any arbitrary job set which makes our algorithm very robust.

Action Space: We choose the action space to be simple, and it is given by $a_t \in \{1, 2,..,i,.., N\}$, where N is the maximum burst time and $a_t = i$ means that the *agent* should schedule the job at the i^{th} slot, which, because of our fixed time state representation, has a burst time of *i*. A valid decision is one in which the *agent* chooses to schedule a job at the non-empty slot. An invalid decision is the one where the *agent* selects an action corresponding to an empty slot as it makes no sense to schedule a job which does not exist. Once the *agent* takes a valid decision, a job is scheduled in the first possible timestamp of the resource clusters in which the resource requirements of the job can be completely satisfied till completion. A state transition is then observed: the scheduled job is allocated it's appropriate position in the resource clusters.

Rewards: Since DQN is a TD learning method, we have crafted a dynamic reward system that will guide our *agent* to the optimal policy by giving an appropriate reward at every time step. We do this by maintaining a counter *c* for the current time in the *environment*. If the *agent* decides to take action a_t, then it is given a reward $r_t = -(a_t + c/a_t)$. If no job exists at the chosen time slot, then a very high negative reward is given. We set the discount factor $\boldsymbol{\gamma} = 1$ so that the cumulative reward of an episode equates the negative of the sum of job slowdown. This way, maximizing the cumulative reward over an episode is equivalent to minimizing the average slowdown.

Evaluation

We evaluate SchedQRM to answer the following questions:

1. How accurately does the DNN time classifier predict the burst time of incoming jobs?
2. How does SchedQRM compare with state-of-the-art heuristics when scheduling online jobs having multiple resource requirement?

Classifier Training and Testing: Before training the *agent* for job scheduling, we train a DNN classifier over the job signature to predict the burst time. Rather than using burst time as a continuous variable, we classify it into ten equally spaced classes. This classification helps significantly in our RL formulation. Data points with outliers and large burst times are discarded to keep the classes

TABLE II: CLASSIFICATION REPORT FOR DNN TIME CLASSIFIER

	Precision	Recall	F1-Score
Class 1	1.00	0.99	1.00
Class 2	0.95	0.99	0.97
Class 3	0.99	0.96	0.97
Class 4	0.94	0.93	0.93
Class 5	0.89	0.86	0.88
Class 6	0.88	0.95	0.91
Class 7	0.94	0.90	0.92
Class 8	0.92	0.98	0.95
Class 9	0.98	0.81	0.89
Class 10	0.83	1.00	0.90

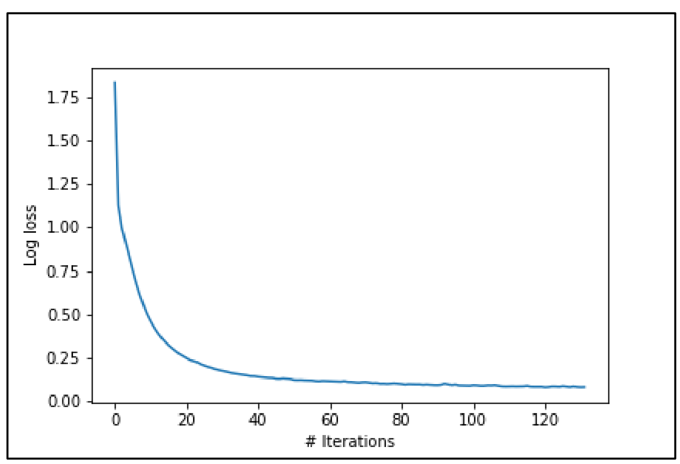

Figure 3: Training loss curve of the classifier.

balanced. A simple DNN classifier with 100 hidden layers, 10 output layers and a learning rate of 0.0001 is trained over 70,000 data points. Rest of 30,000 data points are used to test our model. Our model efficiently converges after 120 iterations by using Adam optimizer [22].

Given the signature of a job, the classifier can predict the burst time class of that job with high accuracy, and this can be seen from Table II. The table shows the values of three evaluation metrics we have calculated for each class 1-10, namely- Precision, Recall, and F1-Score. The classifier can yield satisfying results for each of the classes. Figure 3 shows the training loss curve of the classifier and it's convergence.

DQN Training and Testing: We create a complete job scheduling environment with custom states, actions, and rewards. Our *agent* explores this *environment* and learns an optimal policy with the help of two neural networks and a *replay memory* as used in the DQN algorithm. To implement Experience Replay, we have used a circular memory buffer called *replay memory* which stores new transitions by overwriting the previous ones. The purpose of using a replay memory is:

1. Remembering experience: By storing and sampling transitions from experience replay, our *agent* gets exposed to a broader set of experience and knowledge that helps the *agent* learn more efficiently.
2. De-correlation: If we merely train our *agent* in sequential order, we risk getting our *agent* influenced by the correlation between consecutive states. By randomly sampling transitions from the experience replay, we enable learning from an independent and identically distributed experience.

Job signatures are picked at random from the dataset and fed to the system in an online fashion. This way SchedQRM is trained for arbitrary jobsets and is expected to optimally schedule any set of jobs. which makes it very robust. The cluster load or the number of jobs selected are varied as a percentage of the number of time classes from 10% (1 job for 10-time classes) to 200% (20 jobs for 10-time classes). This set of job is then passed to the DNN classifier to estimate the time of each job. On random, for every job, a dominant or both equally dominant resources are chosen.

In the case of dominant resource, it's resource requirement is chosen between 50% and 100% of maximum resource instances while for the other case, it is chosen in between 0% and 50%. In the case of equally dominant resources, the resource requirement for both the resources is chosen randomly between 10%-100% of total instances. Such job sets are then used to train the SchedQRM.

Every new job set is loaded into the environment's initial state. DQN trains on these job sets to finally converge and form an optimal policy which can be used to determine a scheduling policy for any new arbitrary job set. As stated earlier, we have used two neural networks namely the *evaluate* network and the *train* network for our learning algorithm.

The weights of the *train* network are transferred to the *evaluate* network after every 1000 timestamps, and this helps in stabilizing the DQN algorithm. Each neural network has a single fully connected hidden layer and 20 output layers, one each for an action. The replay memory is a buffer of length 2000. We set the learning rate $\alpha = 0.01$, $\varepsilon = 0.9$ and $\boldsymbol{\gamma} = 1$ for training our *agent*.

Figure 4 represents the plot of the burst time of a job belonging to one of the classes from Class 1 to Class 10 versus the average job slowdown time measured over different jobsets. The extended line shows the maximum slowdown time that was observed for a job belonging to any class.

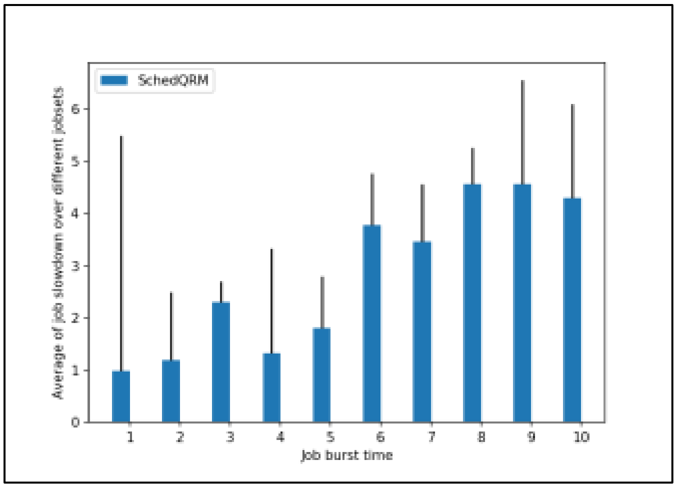

Figure 4: Performance of SchedQRM in terms of average job slowdown.

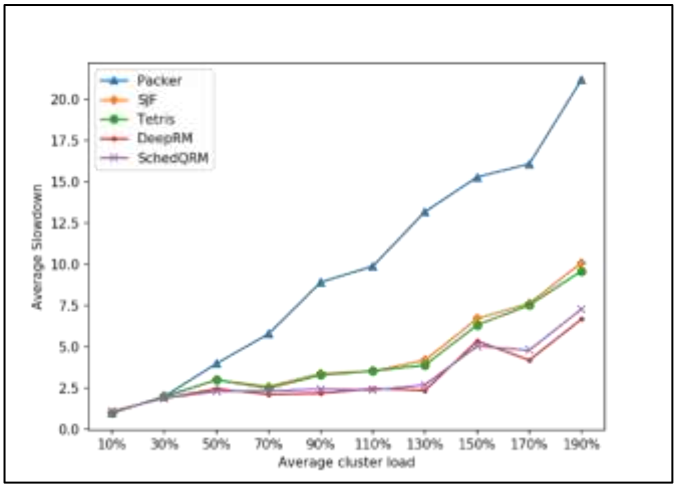

Figure 5: Job slowdown at different levels of load.

A general trend of longer jobs having higher slowdown is observed. This indicates that the *agent* is withholding longer jobs to make space for shorter jobs which helps in reducing the overall job slowdown.

Figure 5 illustrates a comparison between state-of-the-art Packer (the packing heuristic in [21]), Tetris [21], DeepRM [19] and SJF (Shortest Job First) algorithm with our proposed SchedQRM. For all cluster load values, SchedQRM performs either better or equivalent to the existing heuristics. The plot is made by averaging 100 arbitrary job sets at average cluster load value and observing the average job slowdown of all these jobs.

Conclusion

This paper presents our proposed approach for automating an end-to-end process from predicting the burst time of tasks and/or jobs until scheduling them. To achieve this, we present a 2-section model each of which performs one of the tasks stated above. Our RL *agent* focuses on the criterion of average job slowdown. The experiments show that our scheduler SchedQRM outperforms the ad-hoc heuristics. There are certain limitations faced by our model. Firstly, SchedQRM, when trained for average job slowdown, performs not as good as the DeepRM scheduler because of fixed time representation used by the authors in [19]. Our *agent* is unable to choose two jobs of the same burst time together; instead, it selects one of them and keeps the other in the *backlog*. However, such a representation makes SchedQRM much more robust and optimized, and SchedQRM is both trained and capable of working over arbitrary job sets. The second challenge is to interpret the policy used by the *agent* to reach an optimal goal. In general, it holds longer jobs to allow shorter jobs to schedule first, but interpreting the complete policy remains a challenge. We believe these challenges would further motivate research directions in the future.

References

[1] Min Z., Guodong Z., Shibing Z. and Zhihua B., "An Optimized Resource Allocation Algorithm in Cooperative Relay Cognitive Radio Networks," 2017 Signal Processing Symposium (SPSympo).
[2] Feng T., Bi J. and Wang K., "Allocation and Scheduling of Network Resource for Multiple Control Applications in SDN," China Communications, Volume: 12, Issue: 6, June 2015, pp. 85 – 95.
[3] Weiwei X. and Lianfeng S., "Joint Resource Allocation Using Evolutionary Algorithms In Heterogeneous Mobile Cloud Computing Networks," China Communications, Volume: 15, Issue: 8, Aug. 2018, pp. 189 – 204.
[4] F.A. Cruz-Perez and L. Ortigoza-Guerrero, "Equal resource sharing allocation with QoS differentiation for conversational services in wireless communication networks," IEE Proceedings - Communications, Volume: 150, Issue: 5, 14 Oct. 2003, pp. 391.
[5] Tian W., Liu L., Zhang X. and Jian C., "A Resource Allocation Algorithm Combined with Optical Power Dynamic Allocation for Indoor Hybrid VLC and Wi-Fi Network," 2016 8th International Conference on Computational Intelligence and Communication Networks (CICN), Dec. 2016.
[6] Frances Y., Alan D. and Scott S., "A Scheduling Model for Reduced CPU Energy," Proceedings of IEEE 36th Annual Foundations of Computer Science, Oct. 1995.
[7] Yarong C., Ling-Huey S., Ya-Chih T. Shenquan H. and Fuh-Der C., "Scheduling jobs on a single machine with dirt cleaning consideration to minimize total completion time," IEEE Access, Volume: 7, Feb. 2019.
[8] Karim H. and Ahmed J., "A New Approach for Scheduling Tasks and/or Jobs in Big Data Cluster," 2019 4th MEC International Conference on Big Data and Smart City (ICBDSC), Jan. 2019.
[9] Fuming L., Yunlong Z., Chaowan Y. and Xiaoyu S., "Study on the Job Shop Fuzzy Dynamic Scheduling Based on Virtual Pre-scheduling," 2006 6th World Congress on Intelligent Control and Automation, June 2016.

[10] Zhao P. and Huang T., "Research of Multi-resource Dynamic Job-Shop Scheduling based on the Hybrid Genetic Algorithm," 2009 Third International Conference on Genetic and Evolutionary Computing, Oct. 2009.

[11] Xiaomin L., Jiafu W., Hong-Ning D., Muhammad I., Min X. and Antonio C., "A Hybrid Computing Solution and Resource Scheduling Strategy for Edge Computing in Smart Manufacturing," IEEE Transactions on Industrial Informatics, pp. 1-1, Feb. 2019.

[12] Liu Y., Zhong L., Zhang J. and Fu D., "Resource Isolation Policy for Task Scheduling Strategy In Open Real-Time Systems," Proceedings of the 31st Chinese Control Conference, July 2012.

[13] PhuongBang N. and Bhaskar R., "Optimal Scheduling Policies and the Performance of the CDF Scheduling," 2014 48th Asilomar Conference on Signals, Systems and Computers, Nov. 2014.

[14] S N., Mengjie Z., Mark J. and Kay T., "Automatic Design of Scheduling Policies for Dynamic Multi-objective Job Shop Scheduling via Cooperative Coevolution Genetic Programming," IEEE Transactions on Evolutionary Computation, Volume: 18, Issue: 2, April 2014, pp. 193-208.

[15] Wu Q., "Cloud Computing Task Scheduling Policy Based on Improved Particle Swarm Optimization," 2018 International Conference on Virtual Reality and Intelligent Systems (ICVRIS), Aug. 2018.

[16] Frank L., Draguna V. and Kyriakos V., "Reinforcement Learning and Feedback Control Using Natural Decision Methods to Design Optimal Adaptive Controllers," IEEE Control Systems Magazine, Volume: 32, Issue: 6, Dec. 2012, pp. 76-105.

[17] Pingan H. and S. Jagannathan, "Reinforcement Learning Neural-Network- Based Controller for Nonlinear Discrete-Time Systems With Input Constraints," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) (Volume: 37 , Issue: 2 , April 2007), pp. 425-436.

[18] Gerald T., Nicholas J., Rajarshi D. and Mohamed B., "A Hybrid Reinforcement Learning Approach to Autonomic Resource Allocation," 2006 IEEE International Conference on Autonomic Computing, pp. 65-73.

[19] Hongzi Mao, Mohammad Alizadeh, Ishai Menachey, Srikanth Kandulay, "Resource Management with Deep Reinforcement Learning," HotNets '16 Proceedings of the 15th ACM Workshop on Hot Topics in Networks, pp. 50-56.

[20] Tom V., Spyridon S. and Branko S., "On Monte Carlo Tree Search and Reinforcement Learning", Journal of Artificial Intelligence Research, Vol 60 (2017)

[21] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. "Multi-resource packing for cluster schedulers", SIGCOMM '14, pp. 455–466, New York, NY, USA, 2014. ACM.

[22] Diederik K. and Jimmy B., "ADAM: A Method For Stochastic Optimization", 3rd International Conference for Learning Representations, San Diego, 2015.