Software Impacts 8 (2021) 100077

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

# DRAS-CQSim: A reinforcement learning based framework for HPC cluster scheduling

Yuping Fan *, Zhiling Lan

*Illinois Institute of Technology, Chicago, IL, United States of America*

ARTICLE INFO

*Keywords:*
Reinforcement learning
Cluster scheduling
High-performance computing

ABSTRACT

For decades, system administrators have been striving to design and tune cluster scheduling policies to improve the performance of high performance computing (HPC) systems. However, the increasingly complex HPC systems combined with highly diverse workloads make such manual process challenging, time-consuming, and error-prone. We present a reinforcement learning based HPC scheduling framework named DRAS-CQSim to automatically learn optimal scheduling policy. DRAS-CQSim encapsulates simulation environments, agents, hyperparameter tuning options, and different reinforcement learning algorithms, which allows the system administrators to quickly obtain customized scheduling policies.

Code metadata

| | |
|---|---|
| Current code version | v1.0 |
| Permanent link to code/repository used for this code version | https://github.com/SPEAR-IIT/CQSim/tree/DRAS |
| Permanent link to Reproducible Capsule | https://codeocean.com/capsule/7855334/tree/v1 |
| Legal Code License | MIT |
| Code versioning system used | Git |
| Software code languages, tools, and services used | Python, Tensorflow, Keras |
| Compilation requirements, operating environments & dependencies | Any OS supporting Python |
| If available Link to developer documentation/manual | http://bluesky.cs.iit.edu/cqsim/documents/Manual.pdf |
| Support email for questions | yfan22@hawk.iit.edu |

Software metadata

| | |
|---|---|
| Current software version | v1.0 |
| Permanent link to executables of this version | https://github.com/SPEAR-IIT/CQSim/tree/DRAS |
| Permanent link to Reproducible Capsule | https://codeocean.com/capsule/7855334/tree/v1 |
| Legal Software License | MIT |
| Computing platforms/Operating Systems | Any OS supporting Python |
| Installation requirements & dependencies | Python, Tensorflow, Keras |
| If available, link to user manual - if formally published include a reference to the publication in the reference list | http://bluesky.cs.iit.edu/cqsim/documents/Manual.pdf |
| Support email for questions | yfan22@hawk.iit.edu |

## 1. Introduction

Cluster scheduler is crucial in HPC. It determines when and which user jobs should be allocated to available system resources. Traditionally, scheduling policies are developed by system administrators based on their experience with specific systems and workloads. However, such a manually designing and tuning process becomes increasingly difficult to handle the increasingly complex HPC systems and highly diverse application workloads. In recent years, reinforcement learning

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
*E-mail address:* yfan22@hawk.iit.edu (Y. Fan).

https://doi.org/10.1016/j.simpa.2021.100077
Received 31 March 2021; Received in revised form 23 April 2021; Accepted 26 April 2021

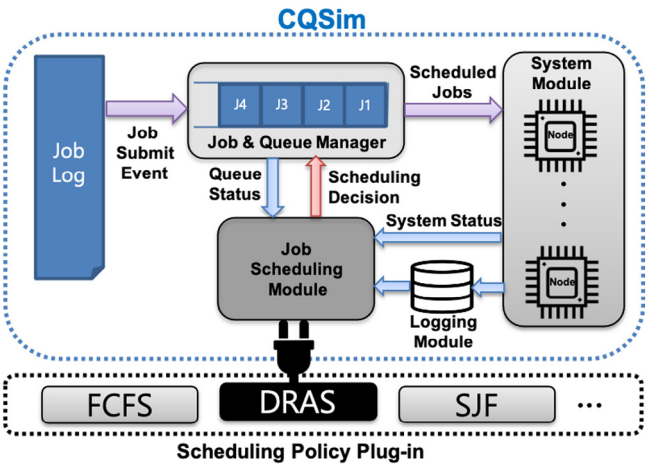


**Fig. 1.** DRAS-CQSim overview.

has been successfully employed in various fields of decision-making problems, such as gaming playing [1,2], self-driving cars [3,4], and autonomous robots [5–7]. Reinforcement learning (RL) is an area of machine learning that automatically learns to make decisions through interaction with the environment.

Inspired by the successful RL examples, we present an open-source HPC scheduling toolkit named DRAS (Deep Reinforcement Agent for Scheduling) to automatically learn customized scheduling policies [8,9]. DRAS with CQSim simulator packs together all the necessary components, i.e., training environment, agents, and RL algorithms, to train the scheduling policy model. Currently DRAS contains the two most popular reinforcement learning algorithms, deep q-learning, and policy gradient [10], and can be easily switch to other reinforcement learning or traditional scheduling algorithms. Our objective is to enable system administrators to quickly obtain the optimal scheduling policies for their specific system and workload environment and easily compare the performance of different scheduling policies.

## 2. Functionalities and key features

DRAS-CQSim, illustrated in Fig. 1, is a reinforcement learning empowered cluster scheduling framework. Rather than executing jobs on real systems, DRAS uses the event-driven scheduling simulator named CQSim [11] to train the agents. CQSim simulates the job scheduling environment by reading the job arrival event from the job log and advancing the simulation clock according to scheduling decisions and job runtime information. CQSim consists of four main components: job & queue manager, system module, logging module, and job scheduling module. The job & queue manager maintains waiting jobs and manages job lifecycle. System module simulates the status of the real systems. Each node in the system is represented as an object and the system module keeps track of each node's availability information. The logging module collects the finished job information, such as job submit time, start time, and end time. The job scheduling module makes scheduling decisions based on the queue status, system status, and historical job information retrieved from queue manager, system module, and logging module respectively. The job scheduling module provides an interface to plugin a customized scheduling policy, such as FCFS (First Come First Serve) and SJF (Shortest Job First). Our DRAS agents are implemented as reinforcement learning based scheduling policies. Our key features are:

- Easy-to-use: DRAS-CQSim requires Python, Tensorflow, and Keras to be install. These prerequisites are easily satisfied on most systems. It provides the concise command to run simulations and train agents. Listing 1 shows an example to train DRAS agents. Argument `n` and `j` are required to run DRAS. Argument `n` specifies the file containing the information of the simulated system. Argument `j` specifies the file containing job traces. The `Config` folder encapsulates all details of scheduling policies, simulated systems, neural networks, requiring no configuration from the user for most use cases. DRAS provides sensible default values, but users can customize configurations via two approaches: directly modify the configuration files under `Config` folder or add optional arguments in command.

**Listing 1:** Train DRAS agent command example

```
$ python cqsim.py -j job_log.swf
        -n node_structure.swf --is_training 1
```

- High scalability: The simulator dynamically streams in the job traces for simulation and streams out the finished jobs to disks. Hence, the memory requirements to run the simulations do not linearly increase with the size of job traces. This allows the simulator to run scalable simulations with a limited amount of memory resources.
- Hyperparameter tuning: Hyperparameters play a crucial role in model performance. To find the optimal hyperparameters, one needs to try various hyperparameter combinations. DRAS requires no source code modifications to find hyperparameters. The customized hyperparameters can be passed to source code through command arguments. The configurable hyperparameters are ranging from learning rate, mini-batch size, epsilon, epsilon decay rate, and discount factor. This allows the developers to launch multiple simulations with different hyperparameters in parallel.
- Performance comparison: The output of the simulation is in `Results` folder by default and it includes the job and system information to evaluate scheduling performance. All simulations have the same output format regardless of scheduling policies. This enables a fair performance comparison of the different scheduling policies on the same log.
- Extensibility: Modules are independent of each other. The scheduling policy plug-in only communicates with job scheduling module and is entirely decoupled from the rest of the system. DRAS employed the two most popular reinforcement learning algorithms, i.e., policy gradient and deep q-learning. In addition, there are several traditional scheduling algorithms, such as FCFS, SJF, and LJF (longest job first), to choose from. New scheduling algorithms can be implemented by creating a new scheduling policy plug-in and replacing the current plug-in.
- Rich debugging facility: We provides five-level logging options to record various events to `Debug` folder. The most detailed logging information captures sufficiently detailed information allowing the developers to quickly identify the issues in their code. To achieve the best performance, the developers can set `debug_lvl` argument to 1, which will record the minimum amount of information to reduce the I/O demand.

## 3. Impact

Reinforcement learning is a highly active research field. Advanced reinforcement learning algorithms have been proposed in recent years [12–16]. However, not all reinforcement learning algorithms are suitable for our HPC scheduling problem. A good solution is supposed to achieve good scheduling performance, such as low average job wait time and high system utilization, with the minimum scheduling overhead. Typical HPC systems tolerate 10–30 s of scheduling delay [17,18]. It is crucial to evaluate the performance and overhead of new RL scheduling methods before deployment. Fortunately, our DRAS design can easily embrace new RL algorithms.

The underlying CQSim scheduling simulator has been successfully supporting a number of projects in this field over a decade [8,19–30]. CQSim provides a unified platform to evaluate the performance of various methods with minimal overheads. For example, [8,24] used CQSim to explore advanced methods, such as reinforcement learning and plan-based methods, to scheduling HPC jobs. The advanced scheduling methods enable the HPC systems to achieve better user-level and system-level performance. [19,22,28] utilized CQSim to explore various factors that could affect the performance of HPC job scheduling, such as job runtime estimate and system utilization. By identifying these factors, the system administrators could develop new policies to minimize the impacts of these factors. [20,21,23,26] aim to find the scheduling strategies to handle multiple resources, i.e., CPU, burst buffer, GPU, and power. CQSim plays a crucial role in these multi-resource projects, because CQSim simulator provides a virtual configurable platform to identify the best scheduling policy to schedule specific resources on a given system before deployment on real systems. [25,29,30] proposed and analyzed novel job placement algorithms on HPC system to improve the efficiency of job placement. Job placement problems are difficult to be measured and conducted on real systems due to the scale of the problems, CQSim provides an easy platform to evaluate the performance of various job placement solutions. Additionally, CQSim is a mature community with many existing scheduling policies, ranging from traditional utility-based policies, optimization methods, to popular RL methods. This enables quick and accurate performance comparison between the new and existing scheduling methods.

DRAS cannot only serve the research purpose, more importantly, the trained RL model can directly deploy to real HPC systems. Thanks to the decoupled scheduling policy design, the system administrators could first train and evaluate the RL agent in the simulator using the historical job logs. Once they obtain satisfactory performance, they can transfer the trained RL agent to real HPC systems and schedule jobs in real-time.

## 4. Conclusion and future work

In this work, we present DRAS-CQSim, a reinforcement learning based HPC scheduling framework, which aims to train RL-based scheduling agents to outperform the traditional scheduling policies. In summary, DRAS-CQSim implements two RL-based scheduling methods, provides a standard platform to design and evaluate RL-based scheduling methods, and enables fair comparison of different scheduling methods. As further developments, we plan to implement and evaluate other RL methods, such as A2C, SAC, and PPO, in order to find the best RL-empowered method for HPC job scheduling.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work is supported in part by US National Science Foundation grants CNS-1717763, CCF-1618776, and the US Department of Energy, Office of Science, under contract DE-AC02-06CH11357.

## References

[1] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature (2016).
[2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. Driessche, T. Graepel, D. Hassabis, Mastering the game of go without human knowledge, Nature (2017).
[3] S. Shalev-Shwartz, S. Shammah, A. Shashua, Safe, multi-agent, reinforcement learning for autonomous driving, 2016, arXiv:1610.03295.
[4] A. Sallab, M. Abdou, E. Perot, S. Yogamani, Deep reinforcement learning framework for autonomous driving, electronic imaging.
[5] W. Smart, L. Pack Kaelbling, Effective reinforcement learning for mobile robots, in: Robotics and Automation.
[6] J. Kober, J. Peters, Reinforcement learning in robotics: A survey, in: Learning Motor Skills: From Algorithms To Robot Experiments, 2014.
[7] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. Ojea, E. Solowjow, S. Levine, Residual reinforcement learning for robot control, in: ICRA'19.
[8] Y. Fan, Z. Lan, T. Childers, P. Rich, W. Allcock, M. Papka, Deep reinforcement agent for scheduling in HPC, in: IPDPS, 2021.
[9] DRAS Github Repository, https://github.com/SPEAR-IIT/CQSim/tree/DRAS.
[10] R. Sutton, D. McAllester, S. Singh, Y. Mansour, Policy gradient methods for reinforcement learning with function approximation, in: NIPS'99.
[11] CQSim Github Repository, https://github.com/SPEAR-IIT/CQSim.
[12] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu, Asynchronous methods for deep reinforcement learning, 2016, arXiv:1602.01783.
[13] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, M. Riedmiller, Deterministic policy gradient algorithms, in: Proceedings of the 31st International Conference on Machine Learning, 2014.
[14] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, Daan Wierstra, Continuous control with deep reinforcement learning, 2019, arXiv:1509.02971.
[15] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, Proximal policy optimization algorithms, 2017, arXiv:1707.06347.
[16] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel, Trust region policy optimization, 2017, arXiv:1502.05477.
[17] W. Allcock, P. Rich, Y. Fan, Z. Lan, Experience and practice of batch scheduling on leadership supercomputers at Argonne in: JSSPP'17.
[18] L. Yu, Z. Zhou, Y. Fan, M. Papka, Z. Lan, System-wide trade-off modeling of performance, power, and resilience on petascale systems, J. Supercomput. (2018).
[19] B. Li, S. Chunduri, K. Harms, Y. Fan, Z. Lan, The effect of system utilization on application performance variability, in: ROSS, 2019.
[20] Y. Fan, Z. Lan, Exploiting multi-resource scheduling for HPC, in: SC Poster, 2019.
[21] Y. Fan, Z. Lan, P. Rich, W. Allcock, M. Papka, B. Austin, D. Paul, Scheduling beyond CPUs for HPC, in: HPDC'19.
[22] Y. Fan, P. Rich, W. Allcock, M. Papka, Z. Lan, Trade-off between prediction accuracy and underestimation rate in job runtime estimates, in: CLUSTER'17.
[23] X. Yang, Z. Zhou, S. Wallace, Z. Lan, W. Tang, S. Coghlan, M. Papka, Integrating dynamic pricing of electricity into energy aware scheduling for HPC systems, in: SC'13.
[24] X. Zheng, Z. Zhou, X. Yang, Z. Lan, J. Wang, Exploring plan-based scheduling for large-scale computing systems, in: Cluster, 2016.
[25] X. Yang, X. Zheng, Z. Zhou, W. Tang, J. Wang, Z. Lan, Balancing job performance with system performance via locality-aware scheduling on torus-connected systems, in: Cluster, 2014.
[26] Y. Fan, P. Rich, W. Allcock, M. Papka, Z. Lan, ROME: A multi-resource job scheduling framework for exascale HPC systems, in: IPDPS Poster, 2018.
[27] Y. Fan, P. Rich, W. Allcock, M. Papka, Z. Lan, Next generation workload management system for high performance computing systems, in: Greater Chicago Area System Research Workshop (GCASR), 2018.
[28] Y. Fan, P. Rich, W. Allcock, M. Papka, Z. Lan, Exploring machine learning to adjust job runtime estimate for high-performance computing, in: Greater Chicago Area System Research Workshop (GCASR), 2017.
[29] P. Qiao, X. Wang, X. Yang, Y. Fan, Z. Lan, Preliminary interference study about job placement and routing algorithms in the fat-tree topology for HPC applications, in: CLUSTER Poster, 2017.
[30] P. Qiao, X. Wang, X. Yang, Y. Fan, Z. Lan, Joint effects of application communication pattern, job placement and network routing on fat-tree systems, in: ICPP Workshops, 2018.