DRAS: Deep Reinforcement Learning for Cluster Scheduling in High Performance Computing

Yuping Fan, Boyang Li, Dustin Favorite, Naunidh Singh, Taylor Childers, Paul Rich, William Allcock, Michael E. Papka, and Zhiling Lan

Present by: An Trieu & Rosie Vuong

Introduction

Common scheduling goals include high system utilization, good user satisfaction and job prioritization.

Heuristics are the prevailing approaches in HPC cluster scheduling.

- First come, first served (FCFS) with EASY backfilling → Scheduling Policy.
- Bin packing → High Utilization.

Optimization methods focus on optimizing immediate scheduling objective(s) without regard to long-term performance.

However:!!!

In case of sudden variation in workloads, system administrators have to manually tune the algorithms and parameters in methods to mitigate performance degradation.

As HPC systems become increasingly complex combined with highly diverse application workloads, such a manual process becomes challenging, time-consuming, and error-prone.

Previous studies do not take into account two special features of cluster scheduling in HPC, that is, resource reservation to prevent job starvation and backfilling to reduce resource fragmentation.

An automated HPC scheduling agent named DRAS

The goal of the agent is twofold:

- To improve HPC scheduling performance beyond the existing approaches.
- To automatically adjust scheduling policies in case of workload changes.

TABLE 1: Comparison of cluster scheduling methods.

Features \ Methods	FCFS [1]	BinPacking [5]	Optimization [2], [3], [4]	Decima [6]	DRAS [7]
Adaption to workload changes	✗	✗	✗	✔	✔
Automatic policy tuning	✗	✗	✗	✔	✔
Long-term scheduling performance	✗	✗	✗	✔	✔
Starvation avoidance	✔	✗	✗	✗	✔
Require training	✗	✗	✗	✔	✔
Implementation effort	Easy	Easy	Median	Hard	Hard
Key objective	Fairness	Resource utilization	Customizable	Customizable	Customizable

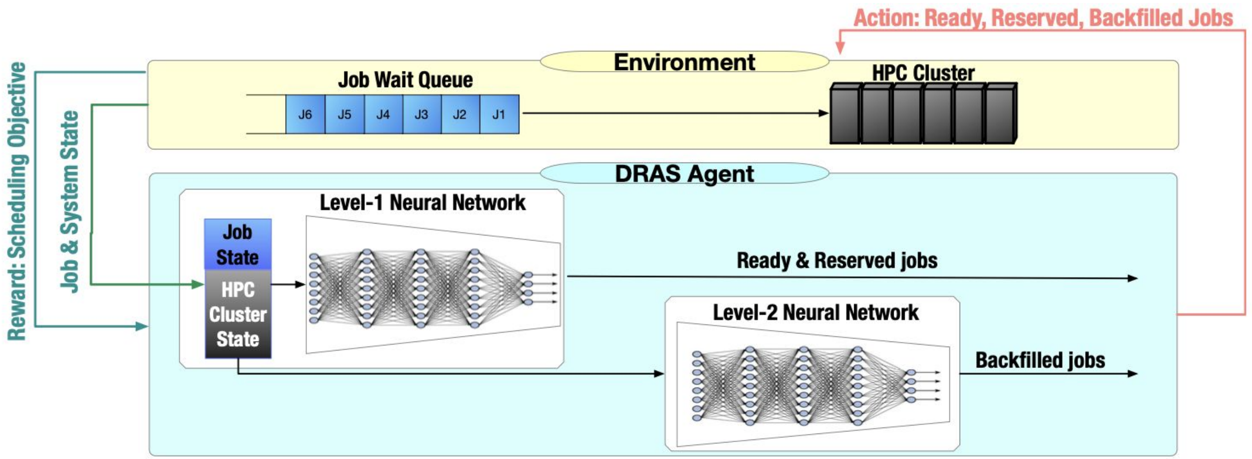
Reinforcement Learning

- Reinforcement learning: learn an optimal policy (maximize reward) through interaction with the environment (series of actions) in random situations (states).
- Interaction:
  - Agent interacts with a dynamic environment in discrete time steps.
  - At each time step (t), agent observes the state (st) and takes an action (at).
  - Environment transitions to a new state (st+1) with a given probability (P).
  - Agent receives a reward (rt) as feedback.

DRAS: HPC Scheduling Agent

DRAS

- DRAS: automated cluster scheduling leveraging reinforcement learning techniques.



DRAS reward, action

- Reward: Capability computing.
- Two types of Capability computing:
    - Type 1: balance 3 factors (prevent job starvation, promote capability jobs, improve system utilization).

$$w_1 \times \frac{\bar{t}}{t_{max}} + w_2 \times \frac{\overline{n}}{N} + w_3 \times \frac{N_{used}}{N}$$

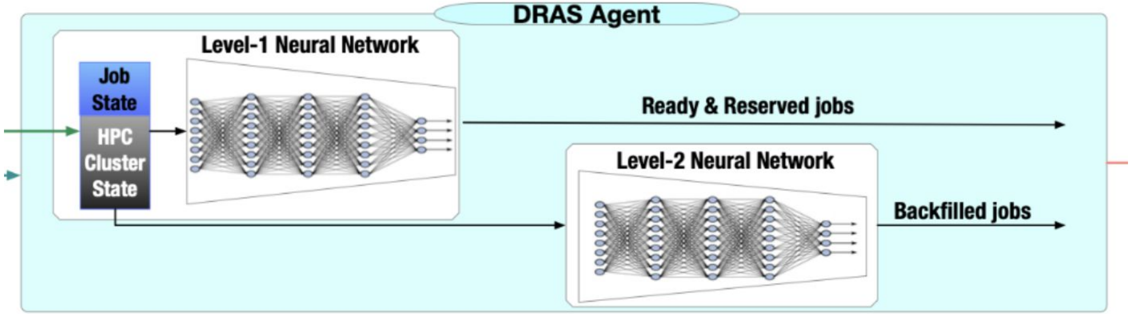
    - Type 2: minimize average job wait time

$$\frac{\sum_{j \in J} -1/t_j}{c}$$

- Action: rather than select multiple jobs at a time, leading to explosive number of actions, DRAS selects one job at a time to prevent explosive number of actions.

DRAS agent

- Goal: prevent job starvation and minimize resource waste.
- The decision making of DRAS is to select jobs and execute them in three modes: ready job, reserved job, backfilled job.
- Using hierarchical neural network structure with 2 levels to identify each mode:
    - Level-1 network (prevent job starvation): select ready jobs and reserved jobs
    - Level-2 network (minimize resource waste): identify backfilled job
- Execution:
    - Level1: Scheduler enforces window at the front of the job wait queue and provide higher priorities to older jobs → less starvation problems
    - If nodes >= job size: mark job as ready
    - If nodes < job size: mark job as reserved
    - Level2: select jobs that can fill in holes before reserved time



4 DRAS agent learning methods:

- 2 main approaches: maximize Q-learning (predict the reward of a certain action taken in a certain state) and policy gradient (directly predict the action itself)
- 4 DRAS agents using 4 reinforcement learning algorithms
  1. DRAS-DQL: maximize Q-value
  2. DRAS-PG: maximize policy gradient
  3. DRAS-A2C: reduce baseline variance in policy gradient method
  4. DRAS-PPO: address problem of large improvement steps on a policy might accidentally cause performance collapse
- Training:
  - Train agent with 3 types of jobsets: (1) a set of sampled jobs randomly selected from real job traces, (2) a period of real job traces, and (3) a set of synthetic jobs generated according to job patterns on the target system.
  - Stop training process once the performance stops significantly increasing.

CQGym

CQGym

A platform to comprehensively evaluate different HPC scheduling policies under the same setting.

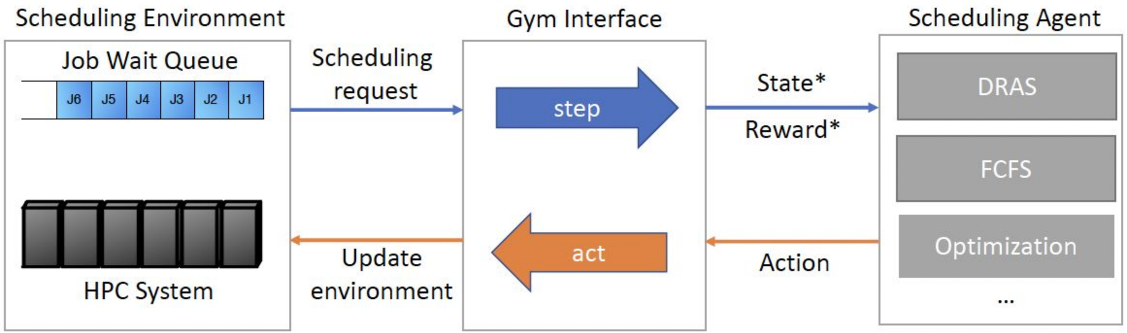
CQGym consists of three main components:

- Scheduling environment
  - an event-driven scheduling simulator that simulates job events
  - Ex: job submission, start, and end,...
- Gym interface
- Scheduling agent
  - Processes scheduling requests from the environment.

The scheduling environment and agent are running on two separate threads.

Gym provides a standard interface to bridge the environment and agent enabling their communication and coordination.

CQGym



simulates the actual scheduling environment.

provides a standard interface between scheduling environment and scheduling agent.

makes scheduling decisions.

Experimental Setup

Comparison Methods

- FCFS/B:
  - Represents FCFS with EASY backfilling.
  - Prioritizes jobs based on their arrival times.
- BinPacking:
  - Iteratively allocates the largest runnable jobs (until the system cannot accommodate any further jobs).
- Random:
  - Randomly selects runnable jobs (until no more jobs can fit into the system).
- Optimization:
  - A suite of scheduling methods that formulate cluster scheduling as an optimization problem
- Decima-PG:
  - Modify Decima by skipping the graph neural network and adopting our state representation
- DRAS:
  - Our HPC custom designs scheduling

Note!!:

- FCFS/B and DRAS are equipped with reservation and backfilling strategies.
- Optimization does not have backfilling and reservation capability.

Workload Traces

TABLE 2: Theta and Cori workloads.

	Theta	Cori
Location	ALCF	NERSC
Scheduler	Cobalt	Slurm
System Types	Capability computing	Capacity computing
Compute Nodes	4,392 (4,392 KNL)	12,076 (2,388 Haswell; 9,688 KNL)
Trace Period	Jan. 2018 - Dec. 2019	Apr. 2018 - Jul. 2018
Number of Jobs	121,837	2,607,054
Max Job Length	1 day	7 days

Two-year job log from Theta

Capability computing focusing on solving largesized problems.

Setup:

- The system size to be 4,360 and filter out all debugging jobs in the trace
- First 2-month data for training, the next month data for validating model convergence, and the rest 21-month data for testing.

Four-month job log from Cori

Capacity computing solving a mix of small-sized and large-sized problems.

Setup:

- the first 2-week data for training, the next 1-week data for validating model convergence, and the last 15-week data for testing.

DRAS Training

TABLE 3: DRAS network configurations for Theta and Cori.

	Theta DRAS-DQL	DRAS-PG	DRAS-A2C	DRAS-PPO	Cori DRAS-DQL	DRAS-PG	DRAS-A2C	DRAS-PPO
Input	[4362, 2]	[4460, 2]	[4460, 2]	[4460, 2]	[12078, 2]	[12176, 2]	[12176, 2]	[12176, 2]
Convolutional Layer	4368	4460	4460	4460	12078	12176	12176	12176
Fully Connected Layer 1	4000				10000			
Fully Connected Layer 2	1000				4000			
Output	1	50	51	51	1	50	51	51
Trainable Parameters	21,449,004	21,890,053	21,891,054	21,891,054	161,764,004	161,960,053	161,964,054	161,964,054

Reward function: $w_1 \times \frac{\bar{t}}{t_{max}} + w_2 \times \frac{\overline{n}}{N} + w_3 \times \frac{N_{used}}{N}$

Set the weights w1 = w2 = w3 = 1/3.

Reward function: $\frac{\sum_{j \in J} -1/t_j}{c}$

The learning rate α is set to 0.001.

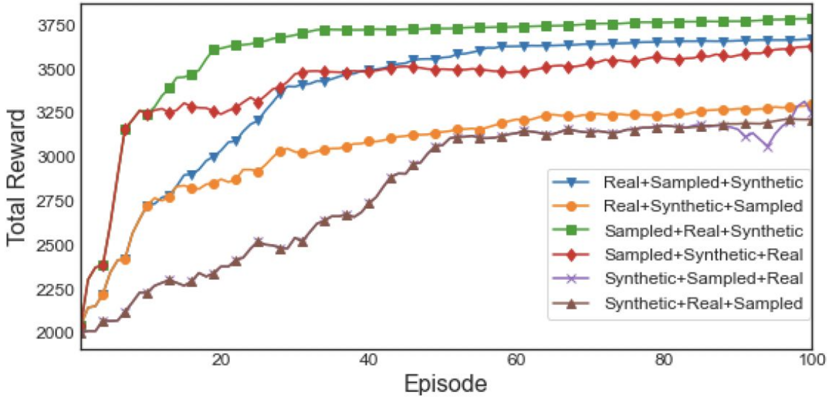
DRAS Training

Validate the trained DRAS agent with an unseen validation dataset → 2 key observations:

- Training only with real jobsets
  - cannot obtain a converged model.
  - more jobsets are needed to train our agents.
- Training order plays an important role in performance
  - Training in the order of sampled, real and synthetic jobsets achieves the best result
  - Training with real jobsets first can also obtain a converged model, the performance is not as good as the case of training with sampled jobsets first.
  - Training with synthetic jobsets first results in slow convergence.

Summary

In order to generate a converged and high-quality model, DRAS needs to first learn from simple averaged cases (sampled jobsets) and then gradually move to more complicated special cases (real and synthetic jobsets).



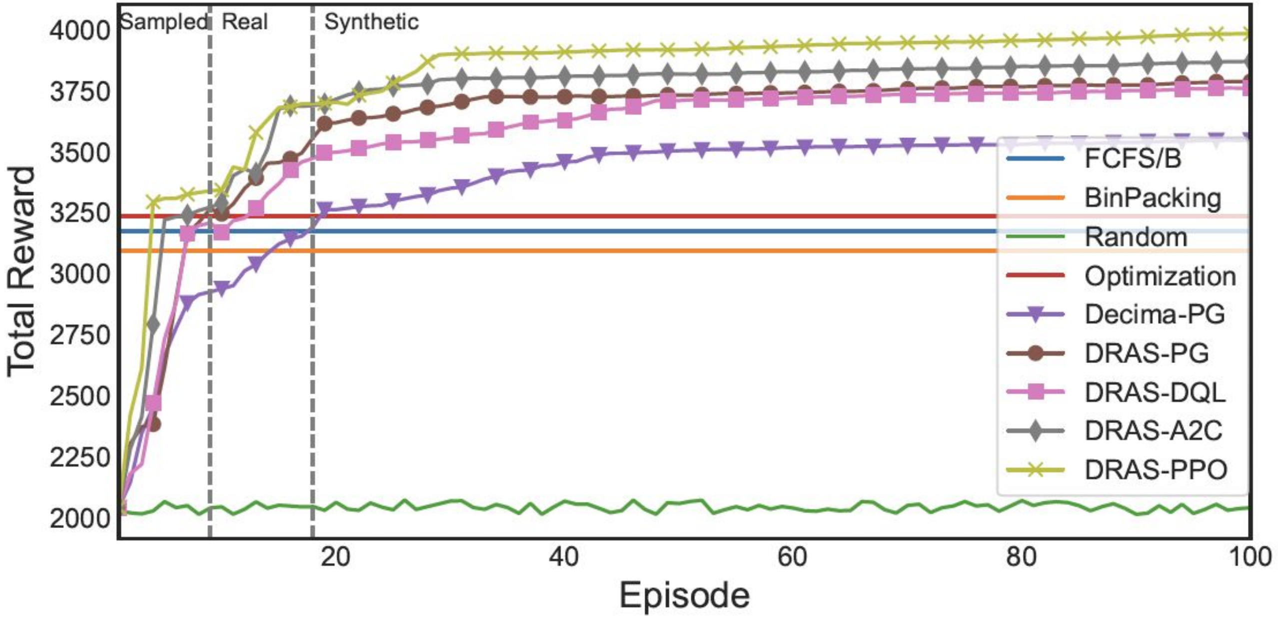


Fig. 6: The total reward collected by the different scheduling methods on Theta validation dataset.

Evaluation Metric

- Job wait time
  - Measures the interval between job submission to job start time
- Job response time
  - Measures the interval between job submission to completion.
- Job slowdown
  - Measures the ratio of the job response time to its actual runtime.
- System utilization
  - Measures the ratio of the used node-hours for useful job execution to the total elapsed node-hours.

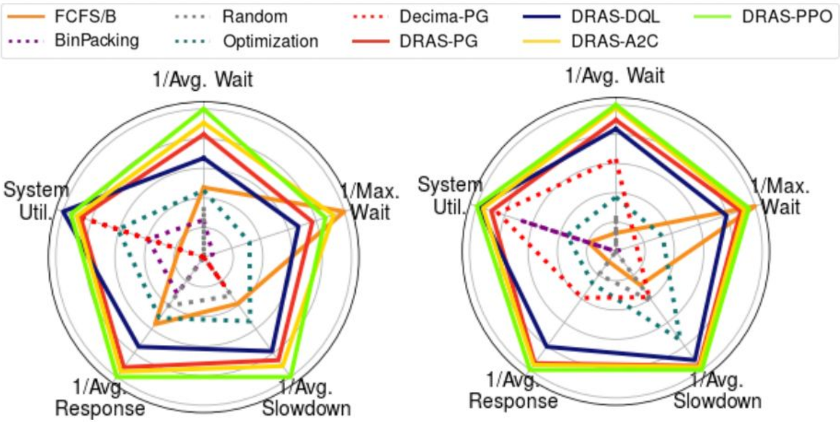
Case Study/Results

Scheduling performance

- DRAS outperforms traditional scheduling methods in both Theta and Cori
- FCFS/B has lowest maximum wait time but poor performance on the rest of metrics

TABLE 2: Theta and Cori workloads.

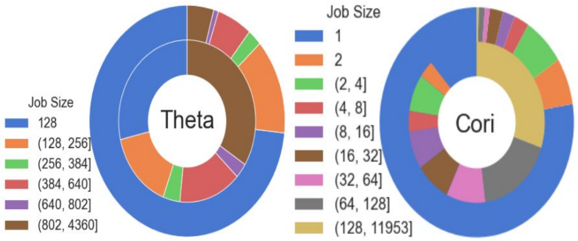
	Theta	Cori
Location	ALCF	NERSC
Scheduler	Cobalt	Slurm
System Types	Capability computing	Capacity computing
Compute Nodes	4,392 (4,392 KNL)	12,076 (2,388 Haswell; 9,688 KNL)
Trace Period	Jan. 2018 - Dec. 2019	Apr. 2018 - Jul. 2018
Number of Jobs	121,837	2,607,054
Max Job Length	1 day	7 days

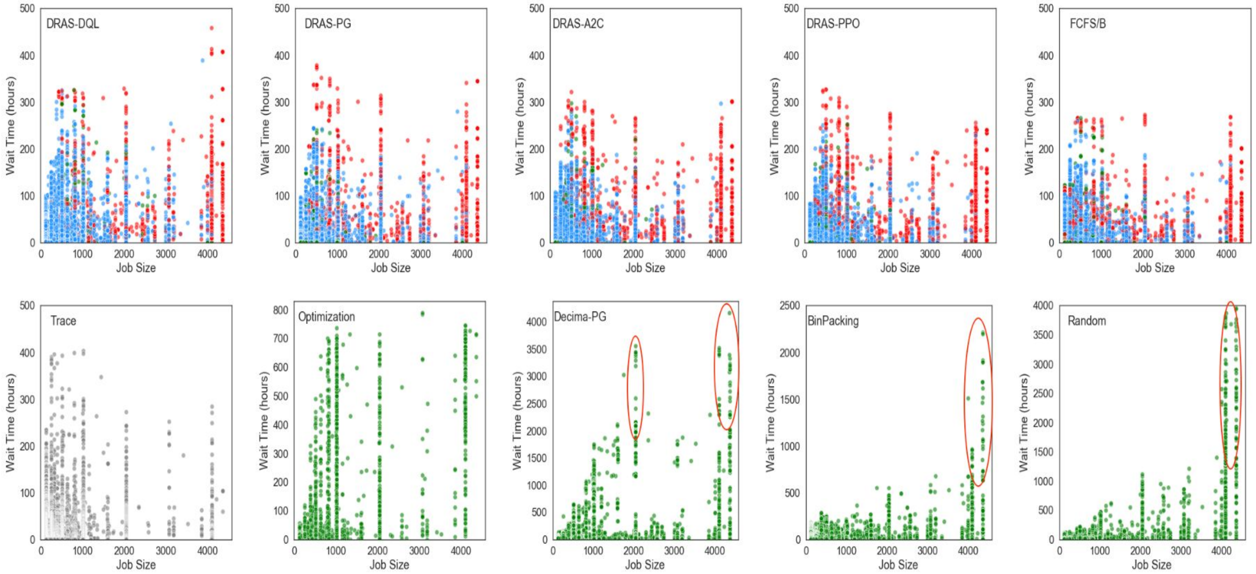


(a) Theta (b) Cori

Job Starvation Analysis

- DRAS do all jobs, while traditional scheduling methods just do small-sized jobs (green) → DRAS can prevent job starvation





Source of DRAS performance gain

- Traditional methods: just have ready jobs, consuming all time, suffer from job starvation
- DRAS: most jobs are executed in backfilling, consume least time. Reserved has least jobs but consume most time → learn to prioritize jobs and prevent job starvation through 2 level network design → maximize long-term scheduling performance
- DRAS takes advantage of incorporating backfilling

TABLE 4: Job distributions in different execution models (defined in §3.2) on Theta.

	Backfilled		Ready		Reserved	
	jobs	core hours	jobs	core hours	jobs	core hours
Optimization	0%	0%	100%	100%	0%	0%
Decima-PG	0%	0%	100%	100%	0%	0%
BinPacking	0%	0%	100%	100%	0%	0%
Random	0%	0%	100%	100%	0%	0%
FCFS/B	79.25%	30.45%	9.88%	16.99%	10.87%	52.56%
DRAS-DQL	84.83%	34.17%	6.84%	10.91%	15.17%	54.92%
DRAS-PG	83.76%	33.67%	8.63%	11.29%	7.61%	55.04%
DRAS-A2C	80.36%	38.48%	10.60%	13.95%	9.03%	47.56%
DRAS-PPO	79.73%	38.57%	10.96%	13.39%	9.30%	48.03%