TASK SCHEDULING USING RECURRENT NEURAL NETWORKS AND DEEP REINFORCEMENT LEARNING

Deep Learning Project

Supervisor: Dr. Achille Mbogol Touye
Student: CHAU Dang Minh

University of Limoges

Outline

Introduction

Motivation

- A high-performance computing (HPC) system consists of nodes. A number of nodes are allocated when a user submits a tasks and recollected after the task is executed.
- If there is no node available, the next task comes the the waiting queue.
- Task scheduling is vital in high-performance computing systems in order to reduce waiting time (the time from submission to starting execution), execution time or total time. Furthermore is there energy.

Problem Statement

- Train schedulers to decide the number of allocated nodes using different techniques: RNN and Deep RL with objective to be the waiting time.
- Compare with some common policies.

Dataset

Dataset

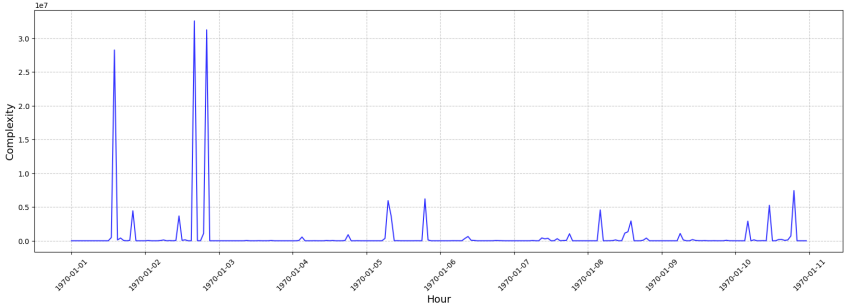
- MIT Supercloud Dataset
- One-year HPC task record
- Concerning features: `time_submit`, `time_start`, `time_end`, `nodes_alloc`

[1] https://www.kaggle.com/datasets/skylarkphantom/mit-datacenter-challenge-data

Dataset Processing

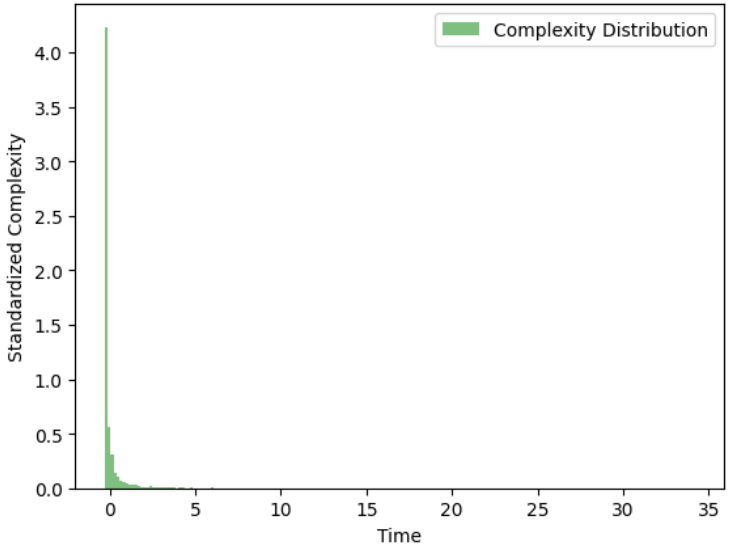
- Assumption: all nodes have the same computing capacity
- Introduce $\texttt{task\_complexity} = (\texttt{time\_end} - \texttt{time\_start}) \times \texttt{nodes\_alloc}$
- Train and test data contain `time_submit` and `task_complexity`.
- Due to computation limit, secondly and minutely predictions are intractable. Daily prediction cannot capture patterns. As an experiment, we group complexities by hour.
- Dataset is split into 90% for training and 10% for testing.

Dataset Processing



Task complexity grouped by hour

Dataset Processing



Approaches

Approaches

Recurrent Neural Networks

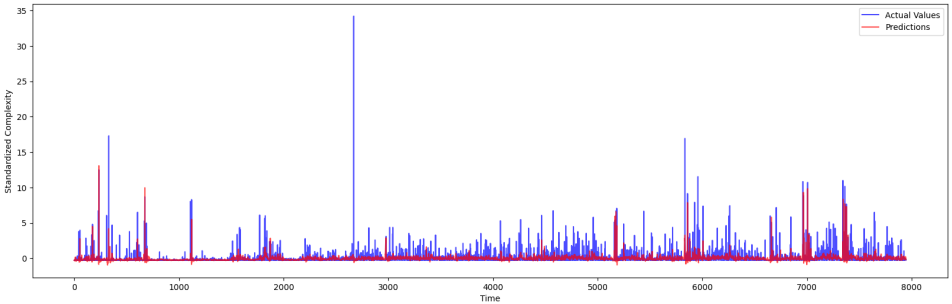
Preparation

1. Fill all time stamps with no task with a complexity-free task
2. Train a recurrent neural network (GRU or LSTM) to predict task complexity in the next time stamp
3. Store the trained task complexity distribution to decide the number of allocated nodes. Suppose that a task with complexity C is at cumulative distribution point p, then it is allocated

$$\min\{\texttt{available\_nodes}, \texttt{total\_nodes} \times p\}.$$

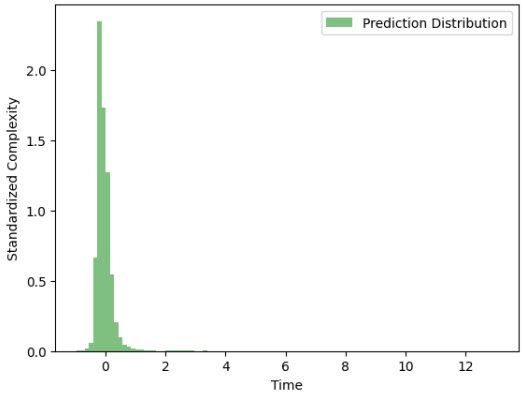
Training and Testing

We train a two-layer LSTM with a 64-dimensional fully connected layers during 300 epochs.



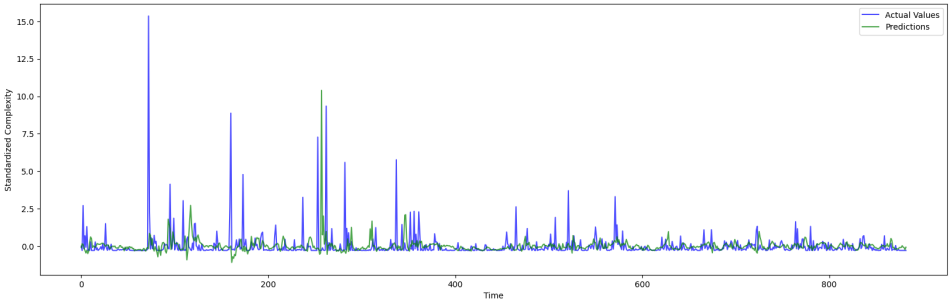
Model fit on the training data

Recurrent Neural Networks



Complexity distribution

Recurrent Neural Networks



Prediction on the test data

Approaches

Deep Reinforcement Learning

Deep Reinforcement Learning

- Environment:
  - total tasks, total nodes
  - waiting tasks, available nodes
  - executing tasks, executed tasks
- Observable states: available nodes, waiting queue (tasks along with waiting times)
- Actions: allocating $n \geq 0$ nodes based on available nodes
- Algorithms: Actor-Critic [1] or Proximal Policy Optimization (PPO) [1]

[1] Konda, Vijay, and John Tsitsiklis. "Actor-critic algorithms." Advances in neural information processing systems 12 (1999).

[1] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

Approaches

Comparison

We compare average waiting times of LSTM scheduler, PPO scheduler against common policies

- Linear scheduler: allocates $\min\{\ell, \texttt{available\_nodes}\}$ for each time stamp, where ℓ is a a constant.
- Stochastic scheduler: allocates $\min\{x, \texttt{available\_nodes}\}$ for each time stamp, where x is the realization of a random variable X with Poisson distribution.

LSTM and PPO schedulers take more time to simulate than common policies. For efficiency, PPO performs over two times better than common policies, while LSTM is slightly better.

Conclusion and Future Work

Conclusion and Future Work

- RNN and Reinforcement Learning can actually learn something in the scheduling context.
- Further experiments can be performed on minutely dataset.
- The combinatorial problem when more than one task is submitted at the same time also needs to be studied.

Thank you for listening !