Optimizing CPU Scheduling with PPO

Aditya Khedekar

2024-11-07

Project Overview

This project develops a Deep Reinforcement Learning (DRL) model that optimizes CPU scheduling, aiming to reduce turnaround time and outperform the traditional Round-Robin approach. Using Proximal Policy Optimization (PPO), the model dynamically adjusts task priorities to enhance scheduling efficiency.

Key Components

The following sections detail the main components of the project, from the PPO algorithm and custom Gym environment to the training process and model components.

Proximal Policy Optimization (PPO)

PPO is a popular DRL algorithm known for balancing sample efficiency with stability. It employs a clipped surrogate objective that prevents large policy updates, enabling stable learning. Here, PPO is central to optimizing scheduling policy, fine-tuning process priorities to minimize turnaround time.

Custom Gym Environment

A custom Gym environment (`PrioritySchedulerEnv`) was developed to provide a controlled setting for training the scheduling model. This environment manages processes based on their arrival time and instruction count, prioritizing them dynamically during runtime. Processes are assigned priorities from 0 to 10, reshuffled in a priority queue, and executed accordingly.

Probability and Action Selection with Multivariate Normal Distribution

In this project, PPO models policy distributions over actions, allowing for probability-based action selection. A multivariate normal distribution is applied to the action space, leveraging a covariance matrix to encourage diverse actions while maintaining stability.

Priority Assignment and Queueing

Processes are assigned priorities (0-10) that determine their positions in the priority queue. This priority directly influences execution order, allowing the model to adaptively reshuffle processes for optimal performance based on real-time feedback.

Reward Model

The reward model in this environment incentivizes quick completion of processes. Rewards are structured as:

- **Positive Reward:** Granted for completed processes.
- **Penalty:** Incurred based on the sum of turnaround times for completed processes.

The PPO model's objective is to maximize positive rewards while minimizing penalties, ultimately optimizing scheduling.

Code Implementation

import numpy as np import torch import gymnasium as gym from torch.distributions import MultivariateNormal from torch.optim import Adam from torch.nn import MSELoss import wandb from neural_network import FeedForwardNN

PPO Class

class PPO: def **init**(self, env: gym.Env, obs_enc_dim: int) -> None: # Initialize environment details self.env = env self.obs_dim = env.observation_space.shape[0] * env.observation_space.shape[1] self.obs_enc_dim = obs_enc_dim self.act_dim = env.action_space.n

```
        # Initialize hyperparameters and neural networks
        self._init_hyperparameters()
        wandb.init(
            project="my-ppo-project",
            config={
                "timesteps_per_batch": self.timesteps_per_batch,
                "max_timesteps_per_episode": self.max_timesteps_per_episode,
                "gamma": self.gamma,
                "n_updates_per_iteration": self.n_updates_per_iteration,
                "clip": self.clip,
                "lr": self.lr,
            },
        )
        self.actor = FeedForwardNN(self.obs_dim, self.act_dim)
        self.critic = FeedForwardNN(self.obs_dim, 1)
        self.actor_optim = Adam(self.actor.parameters(), lr=self.lr)
        self.critic_optim = Adam(self.critic.parameters(), lr=self.lr)
        self.cov_var = torch.full(size=(self.act_dim,), fill_value=0.5)
        self.cov_mat = torch.diag(self.cov_var)

    # Actor-Critic Loss Backpropagation
    def update_networks(self, actor_loss, critic_loss):
        # Actor loss backpropagation
        self.actor_optim.zero_grad()
        actor_loss.backward(retain_graph=True)
        self.actor_optim.step()

        # Critic loss backpropagation
        self.critic_optim.zero_grad()
        critic_loss.backward()
        self.critic_optim.step()
```