PROXIMAL POLICY OPTIMIZATION BASED JOB SCHEDULING ALGORITHM FOR CLOUD DATA CENTERS

DEVLOPER's GUIDE

Aman Singh, CSE, IIIT-NR

Kshitij Kumar Singh Chauhan, CSE, IIIT-NR

1. Introduction

Our project is a Data Center job scheduling simulation framework based on the Bin-Problem framework. It provides a flexible and customizable environment for studying resource management methods in data centers, particularly focusing on machine learning and deep learning-based approaches. The framework is implemented in Python and leverages the scientific computing, deep learning, and machine learning ecosystem of the Python language.

2. Framework Overview

The framework consists of two main packages: "core" and "playground."

2.1. Core Package

The "core" package abstracts and models entities involved in the data center job scheduling problem. It includes the following modules:

- **Config**: Provides configuration classes (TaskInstanceConfig, TaskConfig, JobConfig) for specifying resource requirements and durations of task instances, tasks, and jobs.
- **Job**: Models task instances, tasks, and jobs. The Job class maintains a collection of Task instances, and the Task class maintains a collection of TaskInstance instances. The state information of jobs and tasks is propagated through the entities.
- **Machine**: Models a machine in the data center.
- **Cluster**: Represents a computing cluster and maintains a list of machines.
- **Algorithm**: Defines the interface for scheduling algorithms. Users can implement custom scheduling algorithms by implementing this interface.
- **Scheduler**: Models the scheduler using the strategy pattern. Different instances of the Scheduler class can be used with different scheduling algorithms.
- **Broker**: Implements the class Broker, which replaces users in submitting jobs to the computing cluster.

- **Monitor**: Implements the class Monitor, used to monitor and record the simulation state.
- **Simulation**: Models a simulation. It requires constructing a Cluster instance, a series of JobConfig instances, and a Scheduler instance. Optionally, a Monitor instance can be used to monitor the simulation process.

2.2. Playground Package

The "playground" package provides a convenient environment for conducting experiments within the framework. It contains two sub-packages: "DAG" and "Non-DAG." These packages support simulation experiments considering dependencies between tasks and without considering dependencies between tasks, respectively.

Both sub-packages include pre-implemented heuristic job scheduling algorithms and job scheduling algorithms based on deep reinforcement learning. For example, the "Non-DAG/algorithm/DeepJS" directory contains the Data Center job scheduling algorithm based on deep reinforcement learning.

3. High-Performance Simulation

To achieve high-performance simulation, the framework utilizes several design choices:

- **TaskInstance as SimPy Process**: Conceptually, TaskInstance is designed as a SimPy Process, representing the actual resource consumer and executor of business logic in the data center. The Task and Job classes are designed as collections of TaskInstances. The framework utilizes Python's property feature to synthesize the states of tasks and jobs based on the states of their respective TaskInstance instances. This design allows for efficient and accurate status retrieval, as the acquisition of task and job states is deferred until queried, minimizing active maintenance during simulation time steps.
- **SimPy Processes**: The Broker, Scheduler, and Monitor are also implemented as SimPy processes, enabling continuous submission of jobs, scheduling, and monitoring, respectively.
- **Strategy Pattern**: The strategy pattern is used to decouple the Scheduler implementation from the scheduling algorithm. The Scheduler class serves as the context, while the scheduling algorithms are implemented as separate strategy classes. This separation allows for easy addition or modification of scheduling algorithms without changing the original class or other strategies.

4. Requirements:

- ✓ *Python 3.6 or later*
- ✓ *SimPy 3.0.11*
- ✓ *TensorFlow 1.12.0*
- ✓ *NumPy 1.15.3*
- ✓ *Pandas 0.23.4*

5. Execution and Setup

- Clone codebase

- Set PYTHONPATH:
  Add the path to the project directory to the system environment variable PYTHONPATH. This step allows Python to locate the necessary modules and packages.

- Navigate to the codebase directory and execute:

  ```
  cd BigData_Deepjs/codebase
  python main_exec.py
  ```

  This command will run the simulation using the modified approach for optimizing completion time (makespan). Adjust the script name or arguments according to your specific simulation requirements.

- Monitor the simulation:
  During the simulation execution, the Monitor class will continuously record the simulation state according to the monitoring time step. You can analyze the recorded data to evaluate the performance of the scheduling algorithm and observe the progress of the simulation.

6. Novelty

In our project, we have made several modifications to the original framework. Specifically, we have used a proximal policy gradients-based approach and modified the reward function to consider average completion time and average slowdown as optimization goals. These modifications enhance the framework's ability to handle complex