DATA650 Final Project Report

Muyan Cheng, Steven Chen, Ke Xu, Xueyan Geng, Yuwei Shi

Abstract. *Our project investigates innovative approaches to managing data center resources through reinforcement learning (RL) algorithms. Using the Alibaba Cluster-Trace-GPU-v2023 dataset, we developed a simulation framework to model the complex interactions of computing resources, with a focus on dynamic task allocation, resource utilization, energy efficiency, and workload balancing. We employed two RL algorithms, Linear-Q and PPO, along with a random selection algorithm as a baseline. The performance of these models was evaluated across multiple metrics, including total reward, energy consumption, and temperature management. Our results demonstrate that the RL algorithms significantly outperform the baseline approach in all evaluated metrics. For model construction and environment simulation, we utilized AWS SageMaker, which greatly facilitated the development process.*

1. Group Members Contributions & Implementation Tools

The contributions of the group members are shown below:

- Muyan Cheng: Data Centers Simulation, Reinforcement Learning Model Training, AWS SageMaker Deployment
- Steven Chen: Data Centers Simulation, AWS SageMaker Deployment
- Ke Xu: Data Centers Simulation
- Xueyan Geng: Baseline Model and Reinforcement Learning Model Training
- Yuwei Shi: Data Processing and Information Collection

Implementation: We implement this project using AWS SageMaker. And We use Python as the programming language. The code of our project:DATA650-Final-Code.

2. Problem Statement

As our civilization advances, the energy demand has surged dramatically. Despite progress in exploring renewable energy sources, the majority of our energy—about 80 percent according to studies—still comes from fossil fuels. Reducing fossil fuel consumption to mitigate global warming while maintaining our current energy demands feels nearly impossible at this point. Therefore, instead of solely focusing on alternative energy sources, we should prioritize improving the efficiency of our most energy-intensive industries to limit overall consumption.

One such industry is cloud computing data center, which rank among the most energy-hungry sectors. Even in their relatively early stages, it's projected that their energy consumption could double by 2026. By improving the efficiency of data centers, we could significantly reduce their environmental impact and the pollution that harms our planet.

Motivated by this potential, our team is working to apply machine learning models to optimize and manage the operations of data centers, which are characterized by multiple stochastic processes. The problem we will be solving is efficiency in data centers, in terms of computation efficiency and energy consumption. Our goal is to enhance their efficiency and minimize energy consumption, contributing to a more sustainable future for our Earth.

3. Related Works

Recent advancements in cloud computing and data center management have highlighted the potential of reinforcement learning (RL) to address complex scheduling and resource allocation challenges. Notable studies in this domain include:

- GreenDRL: Managing Green Datacenters Using Deep Reinforcement Learning[1]
  Kuo Zhang, Peijian Wang, Ning Gu, and Thu D. Nguyen proposed GreenDRL, a system leveraging deep reinforcement learning to optimize energy efficiency in green data centers. Their work emphasizes balancing computational efficiency with environmental sustainability, offering a framework that inspires task allocation strategies in dynamic workloads.
- A View on Deep Reinforcement Learning in System Optimization[2]
  Ameer Haj-Ali, Nesreen K. Ahmed, Ted Willke, Joseph Gonzalez, Krste Asanovic, and Ion Stoica review the applications of deep reinforcement learning (DRL) in addressing real-world system optimization problems. The paper discusses key challenges, such as delayed rewards and sequential decision-making, and evaluates DRL's efficiency, robustness, and scalability. The authors propose a set of essential metrics to guide future research while emphasizing the importance of balancing exploration and exploitation in DRL-based optimization strategies.
- Proximal Policy Optimization Algorithms[3]
  John Schulman, et al., introduce Proximal Policy Optimization (PPO), a reinforcement learning algorithm that balances simplicity and performance. PPO optimizes the policy function by limiting the ratio of new to old policy probabilities, enhancing training stability. It has been widely adopted for dynamic scheduling tasks due to its robustness and efficiency.
- Multilevel Data Acquisition System of Energy Losses in Recreation Areas[4]
  Alexander Volkov, et al., propose a multilevel data acquisition system to address energy consumption and heat loss in recreational areas. The study introduces an innovative method for monitoring and optimizing energy systems relevant to data centers aiming to improve resource efficiency.

4. Approach and Solution

We want to take steps to verify the feasibility of the framework outlined in this project. Since we do not have physical access to actual data centers to produce the necessary framework, the idea is that we build that same framework utilizing AWS technologies. First we utilized SageMaker to effectively handle both the machine learning aspects, and computational aspects of our code.

First comes the creation of the data center operations itself. The process of a data center is quite complex and requires a process that can model the complex interactions. Here we end up deciding to utilize an open source library that handles these things. OpenAI has done work in the realm of creating models that handle complex systems. The core idea is that we have an environment that has metrics pertaining to the aspect of the data center that we care about: primarily temperature and energy. We have an agent that will choose between 10 different actions that we want to consider in our system. Each action is defined to interact with the environment in such a way to reduce temperature and consumption.

Now with the environment and actions established, we utilize 2 different approaches for choosing these actions in such a way that our metrics decline. The first approach is using Q-Learning, and the second approach is using Proximal Policy Optimizations. Both approaches are compared against a baseline, where we essentially just calculate metrics based on random selection of actions.
The framework necessary to support these calculations are built off of AWS where we utilize Sagemakerer's computational power in order to run and handle these operations. The main idea here is that after we have trained the RL algorithm with enough data, then we can deploy these models through Sagemaker as physical agents that will regulate data center operations. However, at its current state, our purposes are to use AWS applications to leverage power and workhorse operations.

5. Design of Simulation Environment

The base structure of our Simulation environment was inspired by the Alibaba Cluster-Trace-GPU-v2023 dataset. Especially they key features and key metrics my teammates have collected. With this information at hand, I was able to design an environment that allows us to choose the number of computing units in our system; How much computing power each unit have while monitoring each unit's computing power usage and metrics such as temperature and power efficiency. Once our system is initialized, task will be dynamically allocated to the computing unites through the task allocation process. The task allocation process in our simulation environment is designed to mimic the dynamic and complex decision-making required in real-world data centers. Each task in the system is defined by two main attributes: its computational load, representing the amount of resources it needs, and its duration, which is how long it will occupy those resources. These tasks are managed by a queue, where tasks are processed sequentially and assigned to computational units based on a selected allocation strategy.
When a task is ready to be allocated, the environment uses one of ten predefined strategies to determine how to distribute the task's load across the available computational units. These strategies range from simple methods, like prioritizing units with the lowest capacity or usage, to more advanced techniques, such as hybrid allocation, load balancing, or probabilistic methods. For example, the load balancing strategy divides the task evenly among all units, while the probabilistic method assigns tasks based on a weighted likelihood of each unit's remaining capacity and current load.
OpenAI Gym then allows us to dynamically update the state of computational units by updating energy consumption and temperature metrics based on the workload and capacity utilization of each computational unit. Each unit has a defined capacity, and its current load is continuously monitored. If a task cannot be fully assigned due to capacity limits, the environment tracks the remaining load and attempts to allocate it in subsequent steps. This dynamic task management ensures no resources are wasted and closely resembles the operational logic of real data centers.
Overall, the simulation captures the intricacies of resource management in a scalable and adaptable way, providing a platform for testing and improving task scheduling algorithms in data center environments.

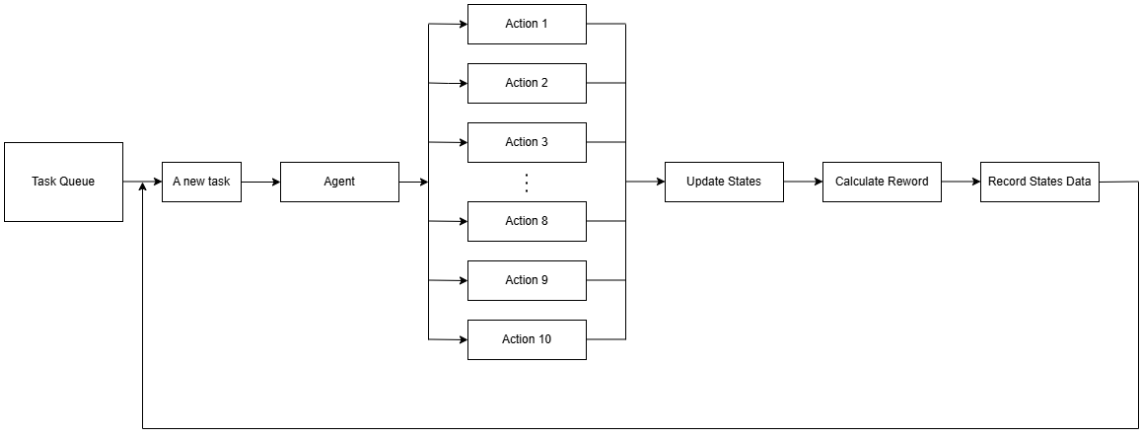

Figure 1. Simulation workflow

5.1. Task Allocation Actions

The simulation environment supports 10 distinct task allocation actions, each representing a unique strategy for distributing tasks across computational units. Below is an overview of each action:

1. **Low-Capacity Unit First:** This action prioritizes computational units with the smallest total capacity, assigning tasks to them first. It is useful in environments where smaller units are underutilized and need to be prioritized to balance overall workload.
2. **Low-Usage Unit First:** In this strategy, tasks are allocated to the units with the lowest current usage relative to their capacity. This approach helps in evenly distributing workloads and avoiding overload on highly utilized units.
3. **Queue Management:** This action breaks a task into smaller sub-tasks and distributes them among the first few units in the queue. It is effective for handling tasks with flexible load requirements, ensuring faster processing across multiple units.
4. **Load Balancing:** Tasks are divided evenly across all computational units, ensuring that every unit takes an equal share of the load. This method minimizes the risk of overloading specific units and promotes uniform resource usage.
5. **Hybrid Allocation:** This strategy combines multiple factors, such as unit capacity, current load, and energy consumption, to dynamically determine the best unit for allocation. It is a more intelligent and adaptive approach to task assignment.
6. **Batch Allocation:** Instead of handling tasks one by one, this action processes multiple tasks at once and distributes them among available units. It is suitable for systems with large queues and can improve overall efficiency by reducing task management overhead.
7. **Overload Redistribution:** This action identifies overloaded units and redistributes their excess load to underutilized units. It is particularly effective in preventing bottlenecks and maintaining system stability under high-demand conditions.
8. **Task Clustering:** Tasks are grouped based on their duration or load requirements and then assigned to units. This approach helps in optimizing processing for tasks with similar characteristics, reducing overhead and improving efficiency.

9. **Priority Scheduling:** Tasks are sorted based on a weighted priority that considers both their load and duration, ensuring high-priority tasks are allocated first. This strategy is ideal for systems with varying levels of task importance.
10. **Probabilistic Allocation:** This action uses a weighted probability to assign tasks to units, factoring in unit capacity, current load, and other dynamic metrics. It adds randomness to the allocation process, making it suitable for scenarios where deterministic methods might lead to suboptimal patterns.

Each action captures a different approach to managing resources, allowing the simulation to model real-world task scheduling challenges effectively. And the resource management is calculated through a resource score:

$$R_{total} = R_{ultilization} + R_{energy} + R_{tempreature} + R_{queue} + R_{random}$$

$$R_{utilization} = 10 \cdot exp(-|U_{avg} - U_{target}|)$$

$$R_{tempreature} = max(0, 5 - tanh((T_{avg}T_{ideal})/10))$$

$$R_{energy} = 10 \cdot \frac{Erenewable}{E_{renewable}+Enonrenewable+e}$$

$$R_{queue} = \frac{10}{1+Q_{remaining}}$$

$$R_{random} = 2 \cdot random.uniform(0, 1)$$

Figure 2. Reward funciton

By testing these actions and calculating their perspective reward score, researchers can refine their strategies and identify the best methods for optimizing resource allocation in data centers.

6. Model

In this project, we compare the performance of different algorithms through several dimensions, focusing on analyzing their performance in terms of resource consumption, reward acquisition, and so on. We have chosen the following Reinforcement Learning strategies for our study: Linear Q, and Proximal Policy Optimization (PPO) algorithms. And we use random selection model as the baseline model.

6.1. Baseline Model: Random Selection

Random Selection serves as a baseline model, where actions are chosen randomly without any learning or optimization. This method defines the minimum benchmark for performance, offering a straightforward way to evaluate whether more advanced strategies improve upon purely random decision-making. Despite its simplicity, Random Selection is useful for highlighting the added value of more sophisticated algorithms in optimizing key metrics such as energy consumption and reward accumulation.

6.2. Linear Q-function

6.2.1. Defination

Linear Q-function represents a more structured approach, using linear approximations to estimate the Q-values of state-action pairs. The Q-function is expressed as

$$Q(s, a; \theta) = \phi(s, a)^\top \theta$$

where $\phi(s, a)$ is the feature vector and θ is the weight vector learned through temporal difference updates.
This linear representation makes the algorithm computationally efficient and interpretable, as the learned weights directly indicate the importance of each feature in determining the value of a state-action pair.
Training the Linear Q-function involves minimizing the temporal difference (TD) error, which measures the discrepancy between the predicted Q-value and the target Q-value. The target Q-value incorporates the immediate reward and the discounted future reward, providing a means for the algorithm to adapt its policy over time. The weights θ are updated iteratively using gradient descent, ensuring that the Q-function converges to a stable approximation of the true state-action values.

6.2.2. Linear Q-function Workflow

1. **Interaction with the Environment**
   - The agent interacts with the environment by selecting an action a_t for the current state s_t using an ϵ-greedy policy based on the current Q-function $Q(s, a; \theta)$.
   - The environment returns a reward r_t and a new state s_{t+1}.
2. **Compute the Target Q-value**
   - Calculate the target Q-value y_t using the Bellman equation:

     $$y_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a'; \theta)$$

     where γ is the discount factor, balancing immediate and future rewards.
3. **Calculate Temporal Difference Error**
   - Compute the temporal difference (TD) error δ_t:

     $$\delta_t = y_t - Q(s_t, a_t; \theta)$$

4. **Update the Q-function**
   - Perform a gradient descent step to minimize the mean squared TD error:

     $$\theta \leftarrow \theta + \alpha \delta_t \nabla_\theta Q(s_t, a_t; \theta)$$

     where α is the learning rate.
5. **Repeat**
   - Iterate through these steps, continuously updating the Q-function $Q(s, a; \theta)$ until convergence or the desired performance is achieved.
   - Optionally, reduce ϵ over time in the ϵ-greedy policy to encourage more exploitation as learning progresses.

6.3. Proximal Policy Optimization (PPO)

PPO is a state-of-the-art reinforcement learning algorithm that improves upon policy gradient methods by ensuring stable policy updates. It uses a clipped surrogate objective to constrain large policy changes and maximize cumulative rewards.

6.3.1. Mathematical Formulation

1. **Objective Function:** PPO's objective function restricts policy updates while optimizing the agent's performance:

$$L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

   Notation:
   - $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$: the probability ratio between the new and old policies.
   - $\hat{A}_t$: the advantage function, measuring the relative value of an action a_t compared to others.
   - ϵ: the clipping parameter, controlling the range of policy updates.
   - $\text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)$: constrains $r_t(\theta)$ within $[1-\epsilon, 1+\epsilon]$.
2. **Advantage Function:** The advantage function quantifies how much better or worse an action is compared to the policy's expected performance. It is computed as:

$$\hat{A}_t = G_t - V(s_t)$$

   Notation:
   - G_t: cumulative rewards starting from time t:

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{T-t} r_T$$

     where γ is the discount factor.
   - $V(s_t)$: the value of state s_t, representing expected future rewards.
3. **Generalized Advantage Estimation (GAE):**
   To stabilize advantage computation, PPO often employs GAE:

$$\hat{A}_t = \sum_{l=0}^{T-t} (\gamma\lambda)^l \delta_{t+l}$$

   where:
   - $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$: temporal difference (TD) error.
   - λ: a smoothing parameter balancing bias and variance in advantage estimation.
4. **Value Function Loss:**PPO optimizes both the policy and the value function $V(s)$. The value function loss is defined as the mean squared error (MSE) between the predicted and actual rewards:

$$L_V(\theta) = \mathbb{E}_t \left[(V_\theta(s_t) - G_t)^2 \right]$$

5. **Policy Gradient:** The policy is updated using the gradient of the PPO objective function:

$$\nabla_\theta L^{\text{PPO}}(\theta) = \mathbb{E}_t \left[\nabla_\theta \min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

6.3.2. PPO Algorithm Workflow

1. **Interaction with the Environment**: The agent interacts with the environment based on the current policy $\pi_\theta(a \mid s)$, receiving a reward r_t and transitioning to a new state s_{t+1}.
2. **Compute Cumulative Rewards**: Use the rewards to compute G_t, the sum of future discounted rewards.
3. **Estimate the Advantage**: Calculate $\hat{A}_t$ using either the simple formula $G_t - V(s_t)$ or GAE.
4. **Optimize the Policy and Value Function**:
    - Update the policy using $L^{\text{PPO}}(\theta)$.
    - Minimize $L_V(\theta)$ to update the value function.
5. **Repeat**: Iterate over these steps, refining the policy and value function, until the desired performance is achieved.

7. Result

The results of the projects see a couple of improvements. First we see that the framework has completed the task that it was assigned to do, which is to choose policies in such a way that the most reward is gathered.

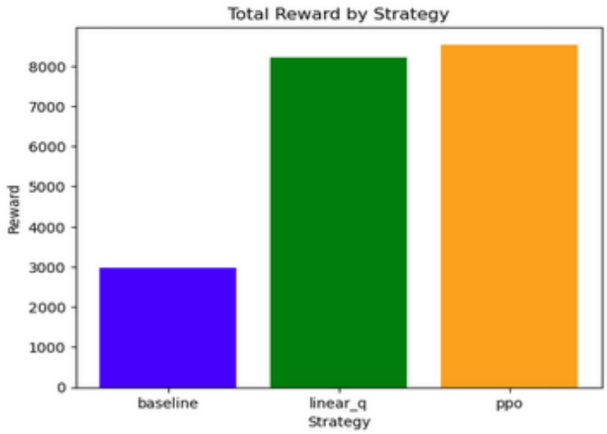

Figure 3. Total Rewards by Strategies

Figure 3 shows a comparison of the total REWARD obtained by the three algorithms. It can be seen that the rewards obtained by the two RL algorithms are much higher than the random selection algorithm. In this case, the reward obtained by the PPO algorithm is a little higher than the reward obtained by linear-Q.
Then, we show the variation of key metrics during the training process for both RL algorithms. It can be seen that the total reward gradually increases shockingly as the number of training episodes increases. Also, it can be seen that PPO has a more obvious upward trend and ends up with more rewards.

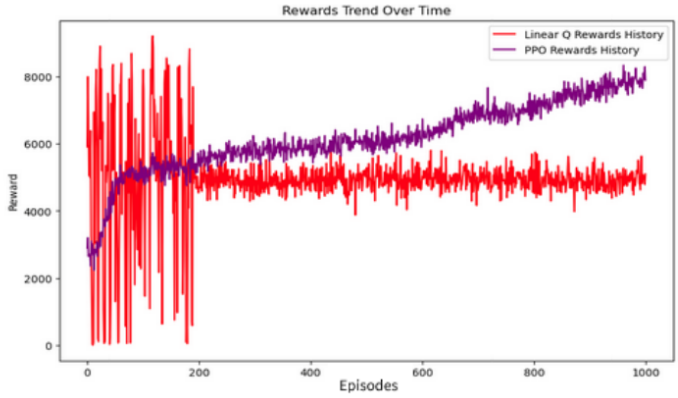

Figure 4. Reward Trend During Training

In the process of data center management, we want the total energy usage to be as low as possible. From Figure 5, it can be seen that as the number of training episodes increases, the total energy usage derived from both RL models shows a decreasing trend, with the PPO algorithm showing a more pronounced decreasing trend.

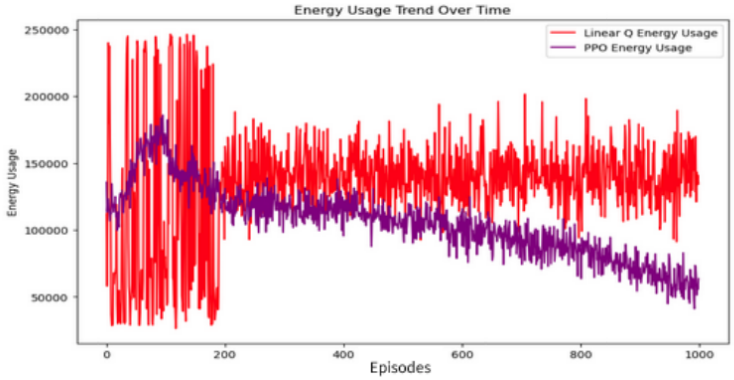

Figure 5. Energy Usage Trend During Training

In addition, in the process of data center management, we would like the average temperature of the data center to be as low as possible, because then we can save the energy used for server cooling. From Figure 6, we can see that as the number of training episodes increases, the average temperature under both RL modeling strategies shows a decreasing trend, where the decreasing trend is more obvious for the PPO algorithm.

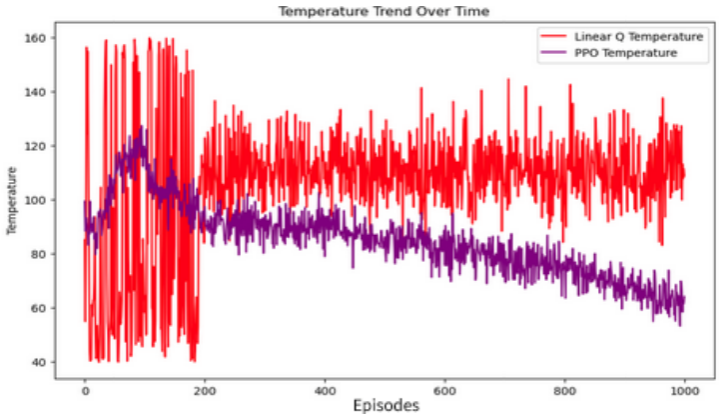

Figure 6. Average Temperature Trend During Training

Based on the above results, the RL algorithm can effectively improve the efficiency of the data center, and the PPO algorithm is better than the Linear-Q algorithm.

8. Conclusion, Novelties, and Significance

The main novelty introduced within the project is the framework to utilize RL algorithms to optimize data center operations. Normally, RL algorithms are usually seen through the lens of calculating maximums of mathematical systems. However, the framework built here, has allowed us to apply RL algorithms towards physical systems. Through the power house of AWS Sagemaker, data center simulations, and well chosen RL models we are able to train a model to reduce data center resource consumptions, verifying that the framework proposed and could in theory be utilized in a practical application. If this framework is applied towards large scale databases, it could greatly reduce the cost of running data centers.

References

[1] Kuo Zhang et al. “GreenDRL: managing green datacenters using deep reinforcement learning”. In: *Proceedings of the 13th Symposium on Cloud Computing*. SoCC ’22. San Francisco, California: Association for Computing Machinery, 2022, pp. 445–460. ISBN: 9781450394147. DOI: 10.1145/3542929.3563501. Available from: https://doi.org/10.1145/3542929.3563501.

[2] Ameer Haj-Ali et al. “A View on Deep Reinforcement Learning in System Optimization”. In: *arXiv: Learning* (2019). Available from: https://api.semanticscholar.org/CorpusID:202539882.

[3] John Schulman et al. “Proximal Policy Optimization Algorithms”. In: *ArXiv* abs/1707.06347 (2017). Available from: https://api.semanticscholar.org/CorpusID:28695052.

[4] Alexander Volkov et al. “Multilevel Data Acquisition System of Energy Losses in Recreation Areas”. In: *2019 Twelfth International Conference "Management of large-scale system development" (MLSD)*. 2019, pp. 1–4. DOI: 10.1109/MLSD.2019.8911033.