Optimization of GPU Cluster Job Scheduling via Reinforcement Learning

- Based on Alibaba PAI-v2020 Cluster Trace

Problem Definition & Motivation

- The Problem:
- - Modern AI/ML workloads vary in size, duration, and GPU demand.
- - Standard FIFO scheduling causes resource fragmentation and high queue delays.

- Commercial Value:
- - 1000 GPUs @ $3/hr = ~$2.16M/month.
- - 10% efficiency gain = ~$2.6M/year savings.
- Objective: Minimize job queuing time & maximize GPU utilization using RL.

Dataset

- Source: Alibaba Cluster Trace Program (v2020)
- Scale: 6,500+ GPUs, ~1,800 machines, 2 months (Jul–Aug 2020)
- Workload: MLaaS – deep learning training & inference

- Key Features:
- - Heterogeneous GPU types (T4, V100, P100)

- Tables: pai_job_table, pai_machine_spec, pai_machine_metric, pai_group_tag_table

Key Challenges & Direction

- Observed Problems:
- Short Task Delays: Many short-duration jobs waited >50% of their runtime in queue.
- High-end GPU Scheduling Issues: Few critical jobs require specific GPUs (V100/NVLink) with topology & concurrency constraints.
- CPU Bottleneck:Data preprocessing (RecSys, GNN, RL) saturates CPU → GPU idle.
- Load Imbalance: Weak-GPU nodes overloaded; high-end GPUs underutilized.
- Resource Mismatch: 2-GPU vs 8-GPU servers have poor CPU/GPU ratio alignment.

- Proposed Solutions:
- Reserving-and-Packing Strategy
  - Reserve top GPUs (V100) for demanding jobs, pack smaller tasks onto low-end nodes.
- CPU/GPU Co-optimization
  - Consider multi-resource scheduling (CPU, GPU, Memory) to avoid bottlenecks.

Data Preprocessing

- Cleaning:
- - Remove missing plan_cpu/mem/duration
- - Keep only successful jobs

- Merging:
- - pai_job → pai_machine

- Result: Unified Job + Machine + Runtime view
- Feature Extraction: state_cols (cluster load), critical_cols (job reqs)

Simulating Heterogeneity

- Challenge: Same job runs faster on V100 than T4

- Method:

- 1. Group by group_tag (graphlearn, ctr prediction, bert etc.)
- 2. Compute avg duration across GPU types
- 3. k = AvgDuration(V100)/AvgDuration(T4)

- Application: Predicted_ T4 = Duatation_V100 / k
- Contribution: Modeling heterogeneous GPU speeds → more realistic simulator.

RL Agent Design

- Flow:
- 1. Sort jobs by start_time
- 2. Update cluster state
- 3. RL decides: assign or wait

- State: cluster load, plan_cpu/mem/gpu
- Action: select machine or wait
- Reward: r = -(current_time - job_arrival) & gpu

- Goal: minimize job waiting delay, maximize gpu utilization

Baselines & Experimental Setup

- Baselines:
- - FIFO: common but inefficient
- - SJF: favors short jobs, risks starvation

- Simulation:
- - Custom Python simulator with real traces

- Metrics: Avg Waiting Time, GPU Utilization

Next Steps

- Immediate:
- - Train RL agent vs FIFO/SJF
- - Visualize waiting time & utilization

- Optimization:
- - Tune Hyperparameter, reward scaling
- - Try DQN/PPO

- Goal: Job scheduling + Open-source simulator