Resource Management with Deep Reinforcement Learning

Hongzi Mao Mohammad Alizadeh
Massachusetts Institute of Technology

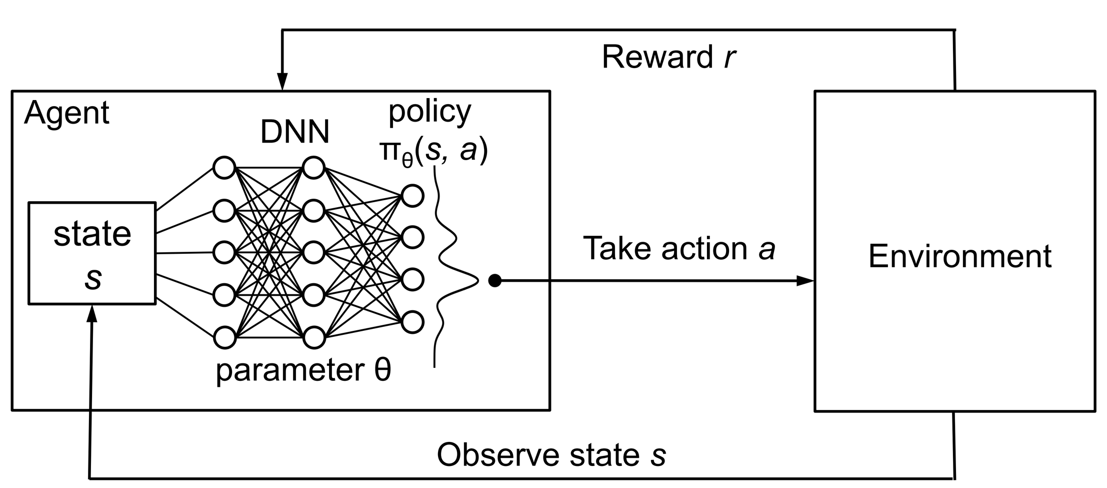
Ishai Menache Srikanth Kandula
Microsoft Research

MOTIVATION

- **Resource management** problems are ubiquitous in **computer systems** and **networks**. They often manifest as difficult **online** decision making tasks where appropriate solutions depend on understanding the **workload** and **environment**.
- Traditionally, the typical design flow is:
  - come up with clever **heuristic** for a simplified model of the problem
  - painstakingly test and **tune** the heuristics for good performance in practice.
- *Can systems **learn** to manage resources on their own?*

BACKGROUND

- In Reinforcement Learning, an *agent* interacts with an *environment*. The agent observes some *state*, and takes an *action* based on its *policy* π_θ. Through the interactions, the environment evolves its states and feedbacks the agent *reward* signals. The goal is to maximize total discounted award $\sum_{t=0}^{\infty} \gamma^t r_t$.



Components of Reinforcement Learning

- The agent learns to tune its policy parameter θ to achieve higher expected total reward, through its experience in state action function Q:

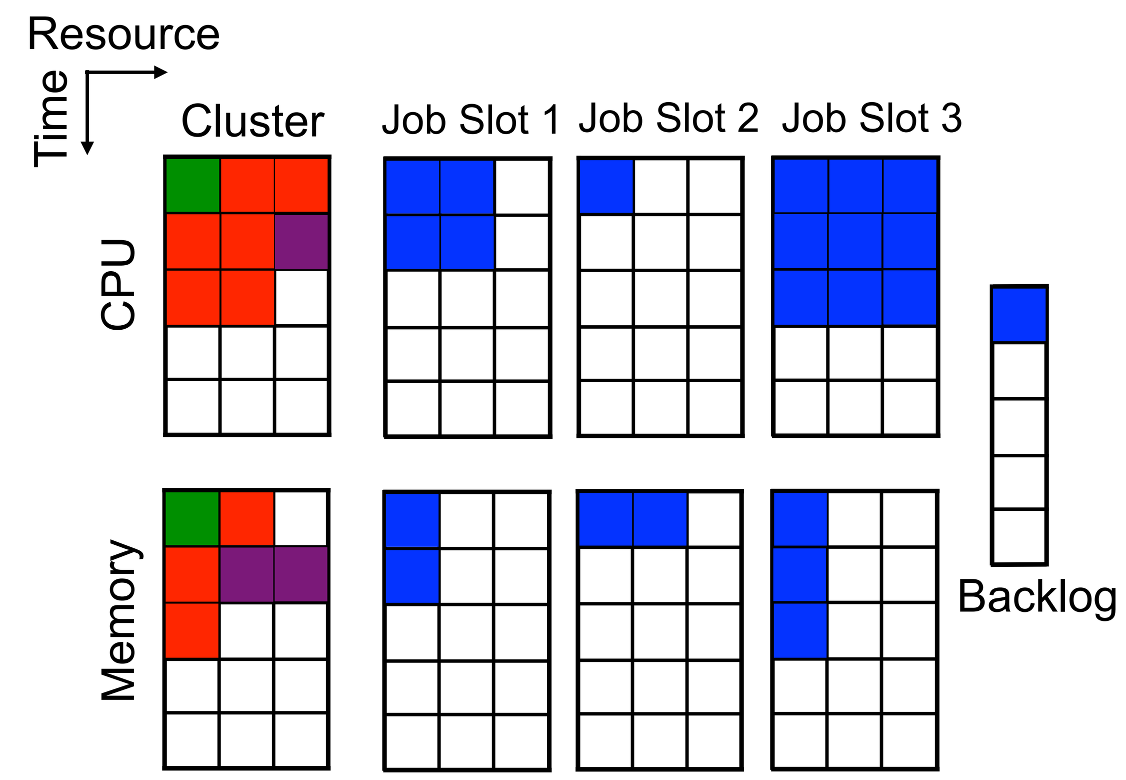
$$\nabla_\theta \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right] = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)\right]$$

- In practice, the training of parameter θ follows policy gradient, and the above Q can be obtained by samples v:

$$\theta \leftarrow \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) v_t$$

- Why is RL a good fit?
  - Computer systems generate a large amount of data for training
  - A natural framework for easy-to-identify signals and observations
  - Optimize the policy directly from experience
  - Train for objectives that are hard-to-optimize analytically
  - Adapt towards different workloads in varying conditions

DESIGN



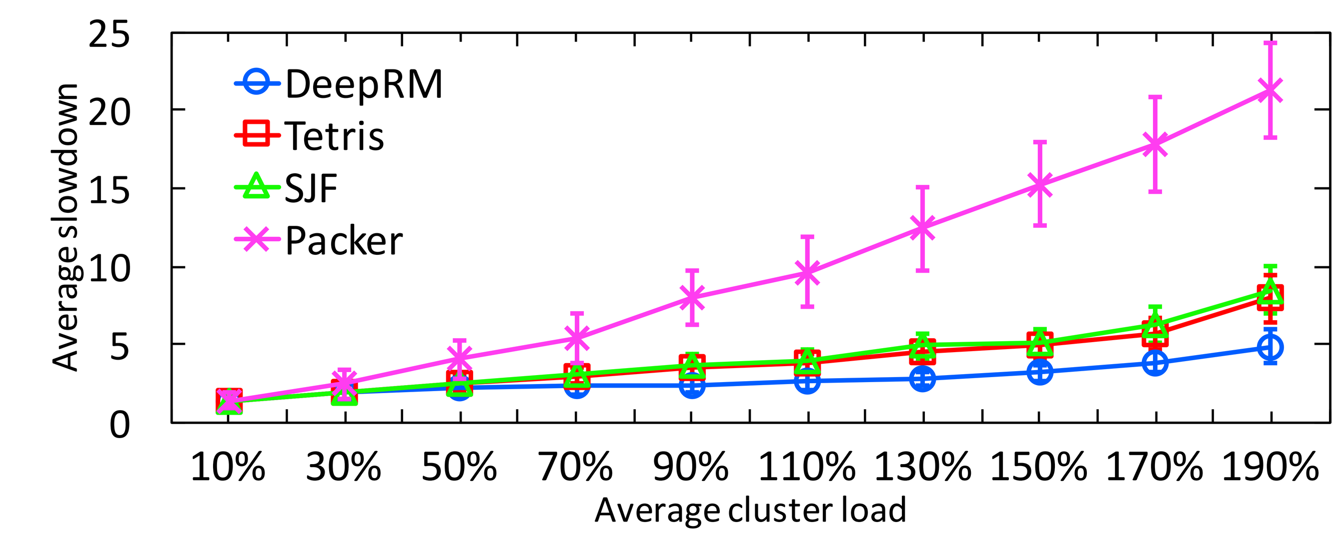
Resource management model

- **State**:
  - Cluster: resource pool of multiple types with a provisioning time
  - Jobs: blocks of resource demand and duration in time
- **Action**: Select which new job to put into the cluster, assuming no preemption and fixed allocation profile
- **Dynamics**: New job(s) arrive along the time, while allocated jobs blocks move up simulating jobs being processed in the cluster
- **Objective**: average job slowdown, given by completion_time/job_duration
- **Reward***:* -1/job_duration penalty for all jobs in the system

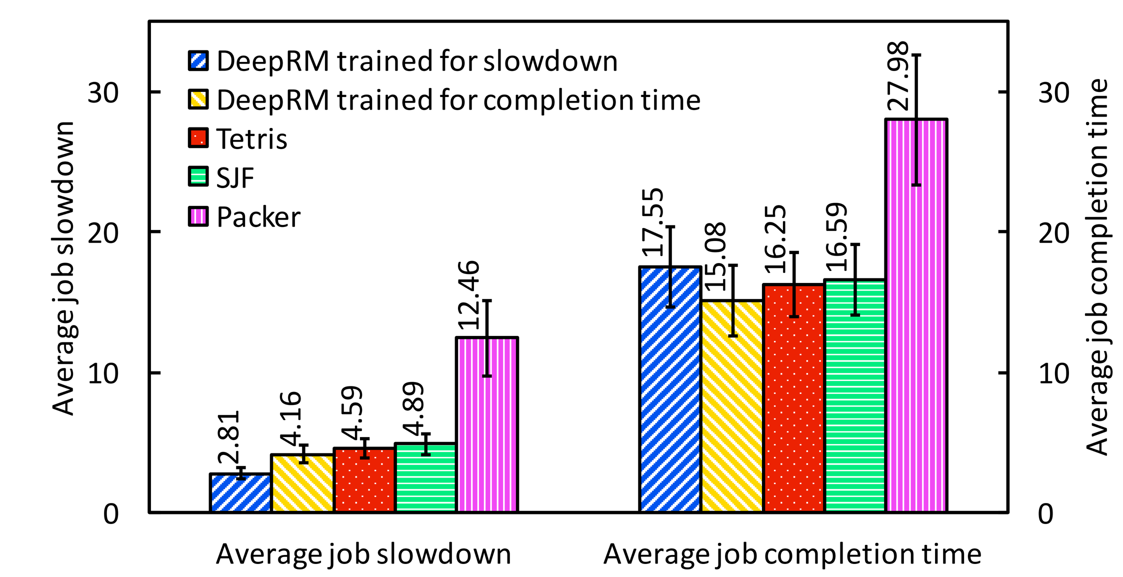
for each iteration:
  $\Delta\theta \leftarrow 0$
  for each jobset:
    run episode $i = 1, \ldots, N$:
      $\{s_1^i, a_1^i, r_1^i, \ldots, s_{L_i}^i, a_{L_i}^i, r_{L_i}^i\} \sim \pi_\theta$
    compute returns: $v_t^i = \sum_{s=t}^{L_i} \gamma^{s-t} r_s^i$
    for t = 1 to L:
      compute baseline: $b_t = \frac{1}{N}\sum_{i=1}^{N} v_t^i$
      for i = 1 to N:
        $\Delta\theta \leftarrow \Delta\theta + \alpha \nabla_\theta \log \pi_\theta(s_t^i, a_t^i)(v_t^i - b_t^i)$
      end
    end
  end
  $\theta \leftarrow \theta + \Delta\theta$ % *batch parameter update*
end

- Train the policy neural network using REINFORCE algorithm with in an *episodic* setting:
  - Sample batches of episodes, where a set of jobs arrive and get scheduled, and we evaluate the cumulative reward following each decision.
  - Update neural network parameters based on the policy gradient for the batch.
- Intuition: the algorithm compare the outcome from each decision and tune the policy to perform more likely on the decisions that lead to better return.

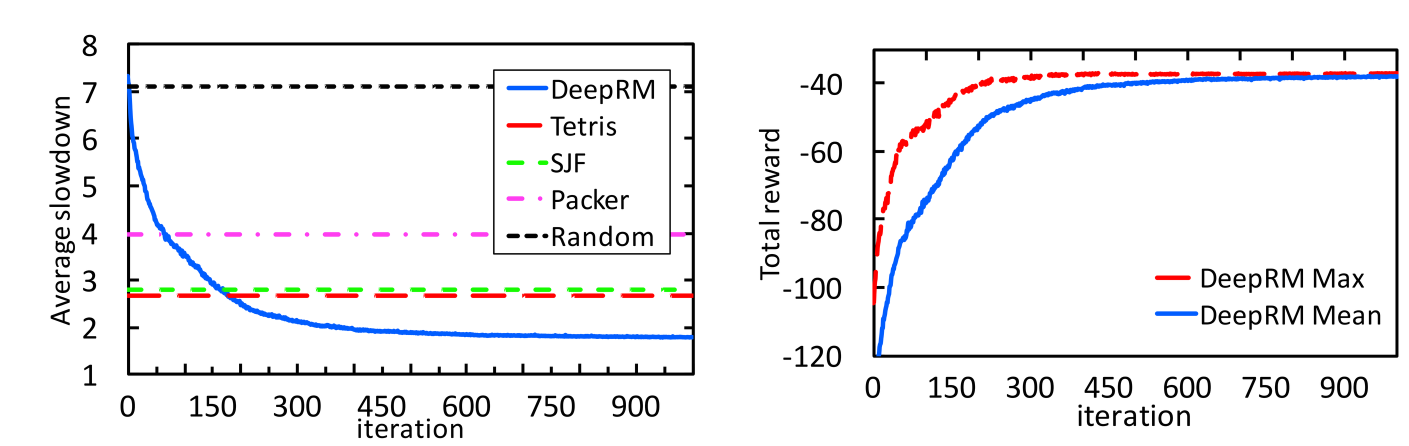
EVALUATION



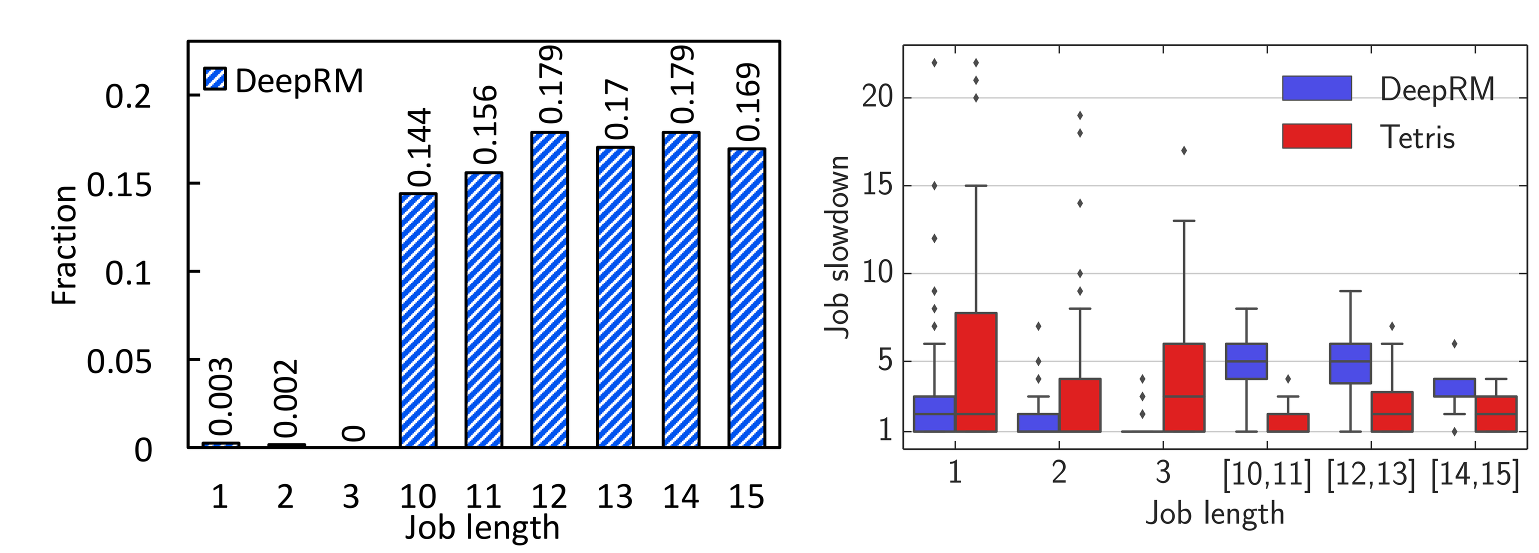
- In a multi-resource bi-model (many small jobs mixed with sporadic big jobs) distributed workload, DeepRM outperforms existing schemes in all workloads.



- By designing different reward signal, DeepRM can tune towards different objectives. E.g., -1 penalty corresponds to minimizing job completion time.



- Learning curve and training procedure of DeepRM



- Where are the gains from : being *non-work conservative*, holding big jobs to leave room for small jobs, resulting in better slowdown for small jobs. DeepRM *learns* this strategy.