Resource Management with Deep Reinforcement Learning

Nayela Tasnim Labonno

Summer term 2020

This paper proposes a Reinforcement Learning approach to solve online resource management problems. RL methods can be convenient to solve the existing hardship of state of the art methods to perceive the problem environment and resource demands entirely and can efficiently deal with them by learning from experience.

1 Introduction

1.1 Comparison between RL approach with general heuristics

Resource management problem is one of the challenging tasks with growing user and application demands in the field of computing and networks. So far, this problem has been dealt with first, by pursuing some heuristic method to solve the problem in a basic level and then by improving the performance of the method against real life system parameters.

The existing approaches are not best suitable as the physical system parameters are hard to comprehend because of varying capacity and features of hardware, inter dependency between programs and resources.

Based on the current researches on RL combined with deep learning, it is assumed that RL based approach can be more practical to solve an online and complex resource management task. As resource management problems happen to have a repetitive pattern, it aids the RL agent with enough training data and thus to progress over time. Moreover,RL approach can come in assistance to problems which cannot be modeled initially as a whole and so an optimized solution is hard to infer. The reward signals could be varied to achieve different system goals.

1.2 Prescribed model definition:DeepRM

The proposed method, DeepRM is a cluster scheduler with multiple resources. It assigns jobs in an online environment and learns to achieve system goals from scratch and optimizes them over time based on different reward signals.

2 System design

2.1 Model

The system is modeled as a single cluster consisting of d resources where vector $r_j = (r_{j,1}, \ldots\ldots, r_{j,d})$ denotes resource demand of each job and T_j represents duration of the job. The scheduler allocates waiting jobs and once a resource is allocated it cannot be preempted.

2.2 RL formulation

2.2.1 State Representation

Figure 1 shows different states of the system for two resources CPU and memory. Leftmost two images represent currently assigned resources to different jobs with different colors. The resources are assigned to the jobs from current time step to T steps on Time axis. Resource requirements for awaiting M jobs are shown in job slot images and rest of the awaiting jobs are counted in the backlog image.

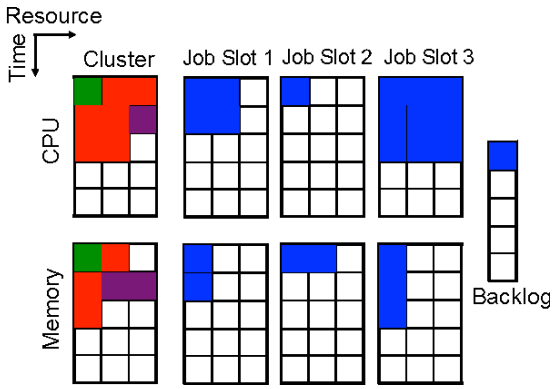


Figure 1: State representation example with two resources and three pending job slots. [**1**],Figure 2

2.2.2 Action Space

If the scheduler assigns M jobs at one time step, the action space gets too large (of size 2^M) leading to a difficulty in learning. To solve this challenge, the action space executes more than one action per time step from action space $\{\varnothing, 1, 2, \ldots., M\}$ to keep it linear. To assist multiple scheduling of jobs in a single time step, time steps are only updated if the agent chooses void or invalid action(when the job cannot be fit with available resource slots).

2.2.3 Rewards

The reward is set to be $\sum_{j \in J} \frac{-1}{T_j}$ to optimize system objective which is to reduce average job slowdown.Here, J is the total jobs in the system at the current time step.

2.3 Policy training algorithm

As the {state,action} pair gets too large because of the large job sets (N=100), instead of tabular form a neural network is used as the policy. The inputs of the policy are states and as output it generates probability distribution over every action in action space for each state.

Jobs arrive in sets at separate time steps and for each jobset there are N different episodes according to figure 2.

The policy is trained in iterations. At each iteration, the discounted cumulative reward v_t is calculated and the policy is trained using the following equation: $\theta \leftarrow \theta + \alpha \sum_t \bigtriangledown_\theta \log \pi_\theta(s_t, a_t) v_t$

```
for each iteration:
    Δθ ← 0
    for each jobset:
        run episode i = 1,...,N:
            {s_1^i, a_1^i, r_1^i, ..., s_{L_i}^i, a_{L_i}^i, r_{L_i}^i} ~ π_θ
        compute returns: v_t^i = Σ_{s=t}^{L_i} γ^{s-t} r_s^i
        for t = 1 to L:
            compute baseline: b_t = (1/N) Σ_{i=1}^{N} v_t^i
            for i = 1 to N:
                Δθ ← Δθ + α∇_θ log π_θ(s_t^i, a_t^i)(v_t^i - b_t^i)
            end
        end
    end
    θ ← θ + Δθ  % batch parameter update
end
```

Figure 2: Policy training algorithm. [**1**],Figure 3

To reduce the high variance generated from gradient estimate a baseline value is subtracted from v_t, where the baseline value is the average of v_t for same job sets and time step and over all episodes.

2.4 System Optimization Criterions

The key system objective here is *average job slowdown* which is denoted as $S_j = C_j/T_j$, where C_j is the job completion time. Another reward function $-|J|$ is used to realize average job completion time.

3 Evaluation

3.1 Methodology

Jobs arrive according to a Bernoulli process keeping an average workload of 10% to 190% and there are two resources to demand. Each job is assigned a dominant resource randomly the demand of which is uniformly chosen between 0.25r and 0.5r and for the other job between 0.05r and 0.1r.

In this setup total 100 job sets are considered each containing M=10 jobs.Over 1000 iterations, N=20 Monte Carlo simulations are executed for each job set.

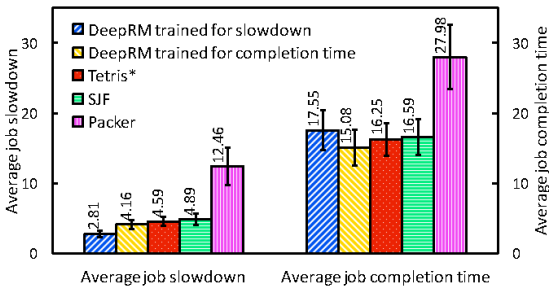
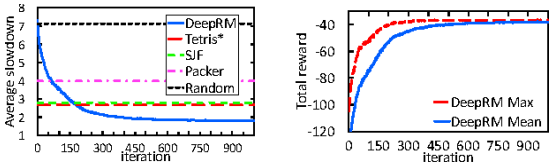


Figure 3: Performance comparison for different objectives. [**1**],Figure 5



(a) average slowdown (b) total reward

Figure 4: Average slowdown and total reward performance learning curve. [**1**],Figure 6.

3.2 Comparing scheduling efficiency

DeepRM is evaluated for average job slowdown and average job completion time and compared with a Shortest Job First agent, a Packer agent allocating jobs by job demand and resource requirement adjustment and Tetris which schedules comparably larger jobs whose resource demands are satiable.

Figure 3 shows the performance comparison for a workload of 130% where SJF performs better than Packer and Tetris outperforms them both for each objectives. It also shows that DeepRM can be optimized for different objectives by different reward function and performs equally well or better than other heuristics.

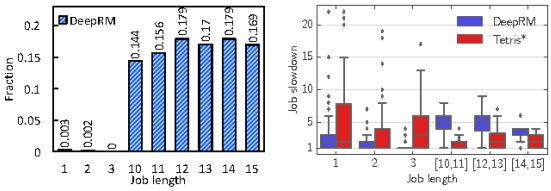
3.3 Convergence behavior of DeepRM

Average job slowdown over iterations is shown in figure 4(a). It shows how DeepRM is improving it's performance gradually and after around 200 iterations it outperforms Tetris.

Figure 4(b) shows maximum and average reward and the upward trend of the curves means better performance with improved policy. The gap between average and maximum reward until convergence signifies how some of the action path was performing well than the average one,hence the policy was being updated for better reward value over the iterations.

3.3.1 How DeepRM gains better performance

The reason behind performance gain of DeepRM lies in the fact that it holds back the large jobs from allocating despite having available resources only to keep enough space for the upcoming small jobs. Figure 5(b) depicts the scenario. This strategy comes out to be an optimal one as in this specific workload small jobs appear 4× than large jobs.



(a) length of withheld jobs (b) slowdown vs. job length

Figure 5: Performance gain of DeepRM by holding large jobs back. [**1**],Figure 7.

4 Discussion

4.1 Limitation

The discussed system design does not take into account many practical aspects like considering one single cluster of resources and overlooking resource fragmentation which is very common in real life systems. In practical systems, most often tasks are internally dependent on each other and the resource demands of applications may not be known at once.

The varying data locality of different jobs are not taken into consideration as well. Moreover, RL approach depends on large number of iterations to

learn and come up with a strategy to improve the reward gain, which makes it only appropriate for highly repetitive problems.

4.2 Future research direction

The agent can be trained to pursue data local allocations by appropriate reward function selection.The computation of baseline in figure 3 needs a bounded time horizon which is a drawback and can be solved by calculating average return value with a value network instead [**2**].

5 Summary

To sum up, Reinforcement Learning approach can be a viable alternative to current heuristics for solving complex and dynamic resource allocation problems. The exploration and exploitation features of RL approaches assist them to comprehend the problem environment and come up with optimum solution over time.

References

[1] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource Management with Deep Reinforcement Learning," *Proceedings of the 15th ACM Workshop on Hot Topics in Networks,* pp. 50–56, 2016.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning an introduction.* The MIT Press, 2018.