Resource Management with Deep Reinforcement Learning

Nayela Tasnim Labonno

Summer term 2020

Resource management problems, being dynamic and complex are difficult to comprehend and solve with the existing heuristics. Mao et al. [**1**] proposes a Reinforcement Learning (RL) approach to solve online resource management problems due to recent advances in RL field. RL methods can learn optimization strategies from experiences without any prior knowledge, which makes them convenient to solve scheduling problems with large workloads.

1 Introduction

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.

1.1 Comparison between RL approach with general heuristics

The existing approaches are not best suitable to solve resource management tasks. The physical system parameters are hard to comprehend because of varying capacity and features of hardware. Also, the inter dependency between programs and resources makes the modeling task more challenging.

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. In order to achieve different objectives, the reward signals can be designed and varied according to them.

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 Design

The problem discussed in section1 is modeled and formulated as a RL task here and the reward function are crafted in a way to assist RL agent in achieving system objective .

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 (ideal) of the job. The scheduler allocates resources to the waiting jobs and once a resource is allocated, it cannot be preempted. To keep the design simple, the resources will be allocated continuously from the starting of the job until the execution is finished.

2.2 RL formulation

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. Jobs are represented with different colors. The resources are assigned to the jobs from current time step to T steps on time axis. For example, the purple job in the first cluster image is allocated one unit of CPU and two units of memory for one time step.

Resource requirements for awaiting M jobs are shown in three of the job slot images. For example, the job in slot 3 will require three units of CPU and one unit of memory for the next three time steps. The reason to maintain only a fixed amount of M awaiting jobs is to make it applicable as a neural network input. The rest of the awaiting jobs are counted in the backlog image.

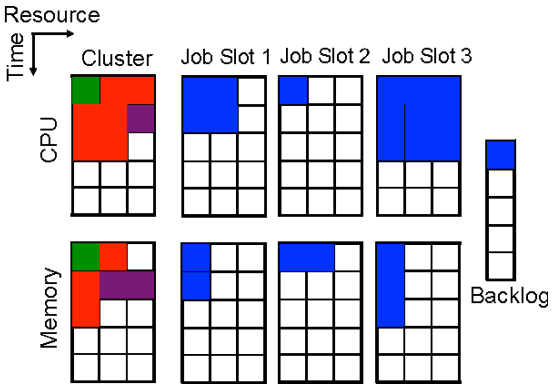


Figure 1: State representation example with two resources and three pending job slots. [**1**],Figure 2

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, more than one action per timestep are allowed to be executed 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).

System objectives and rewards. The key system objective here is *average job slowdown*. The slowdown is denoted as $S_j = C_j/T_j$, where C_j is the job completion time. In order to correlate with the objective, the reward function is set to $\sum_{j \in J} \frac{-1}{T_j}$, by taking discount factor $\gamma = 1$. Here, J is the total number of scheduled and waiting jobs in the system at the current time step. Therefore, an attempt to maximize the reward value by RL agent leads to minimization of average job slowdown.

Other objectives can also be achieved by customizing the reward function. In their work, Mao et al. [**1**] used $-|J|$ as the reward function to minimize average job completion time. Thus less number of unfinished jobs in the system will result in higher reward and help it to optimize.

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 to represent the policy. The inputs of the policy network are states and as output it generates probability distribution over actions for each state.

The policy training occurs in an episodic manner. In every iteration, a group of jobs arrive which are called jobsets. For each jobset, N episodes are run to schedule the jobs using current policy and the return information (state, action, reward) for each time step are stored. Later, these return values are used to compute the discounted cumulative reward, v_t. Thus the policy network is trained according to the pseudo-code shown in Figure 2, which is a variant of the REINFORCE algorithm [**3**].

```
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

Using a policy gradient approach leads to a problem of high variance and slow convergence. To reduce the variance, a baseline value, b_t is subtracted from the cumulative reward. The baseline value is the average of cumulative reward, v_t for same jobsets and timestep and over all episodes.

3 Evaluation

In this section, DeepRM is evaluated for average job slowdown and average job completion time and compared against three standard heuristics, such as a Shortest Job First agent; a Packer agent allocating jobs by job demand and resource requirement adjustment; and Tetris [**4**] which schedules comparably larger jobs whose resource demands are satiable.

3.1 Methodology

Jobs arrive according to a Bernoulli process keeping an average workload of 10% to 190% . Two resources are taken into account with capacity {1r, 1r}. Each job is randomly assigned one dominant resource, the demand of which is uniformly chosen between 0.25r and 0.5r and for the other resource, 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.

3.2 Comparing scheduling efficiency

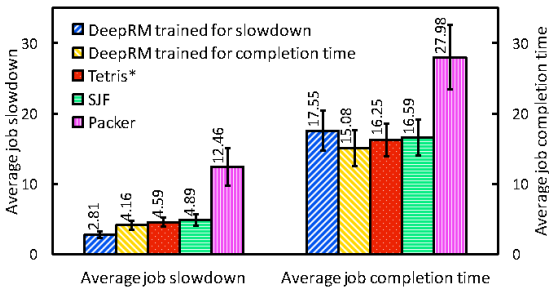


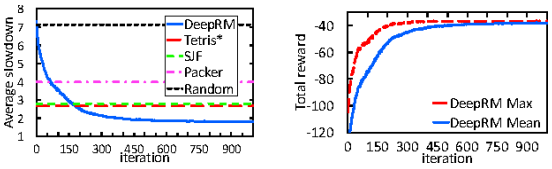
Figure 3: Performance comparison for different objectives. [**1**],Figure 5

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

3.3 Convergence behavior of DeepRM

Figure 4(a) depicts the average job slowdown by DeepRM over 1000 iterations with 70% load. It shows how DeepRM is improving its performance gradually and after around 200 iterations it outperforms Tetris.

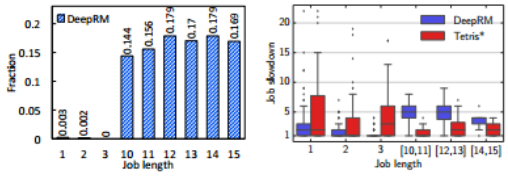
Figure 4(b) shows maximum and average reward and the upward trend of the curves means better performance with improved policy. The gap between average and maximum reward until convergence signifies how some of the action path was performing better than the average one, hence the policy was being updated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



(a) average slowdown (b) total reward

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

Performance gain of DeepRM. The reason behind performance gain of DeepRM lies in the fact that after several iterations, it learns a strategy to withhold larger jobs from allocating resources to make sure that there is 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× more 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

The drawbacks of the proposed system are discussed and probable scopes of improvement are pointed out here.

4.1 Limitation

The approach described in [**1**], does not take into account many practical aspects. It considers one single cluster of resources and overlooks 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 considered as well.

4.2 Future research direction

The agent can be trained to pursue data local allocations by formulating an appropriate reward function in future.

For the computation of baseline in Figure 3, DeepRM uses a bounded time horizon. But, this approach is infeasible in practical resource allocation problems as they end up having infinite time horizons. To resolve this drawback and to reduce the depth of the network, a value network [**2**] in addition to the policy network can be a probable solution according to [**1**].

5 Conclusion

With their experiments, Mao et al. [**1**] shows that, Reinforcement Learning approach can be a viable alternative to current heuristics for solving large scale resource allocation problems. A RL agent is able to sense the problem environment and by exploiting a proper reward function, it can devise strategies all by itself. Moreover, it is also possible to update the policy over iterations and thus optimizing the problem environment.

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, 1998.

[3] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al, "Policy gradient methods for reinforcement learning with function approximation," *Proceedings of the 12th International Conference on Neural Information Processing Systems,* pp. 1057–1063, 1999.

[4] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella. "Multi-resource packing for cluster schedulers," *Proceedings of the 2014 ACM Conference on SIGCOMM,* pp. 455–466, 2014.