REINFORCEMENT LEARNING FOR RESOURCE MANAGEMENT

A Summary of Research Submitted to the Department of Engineering and Society

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia – Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science in Computer Science

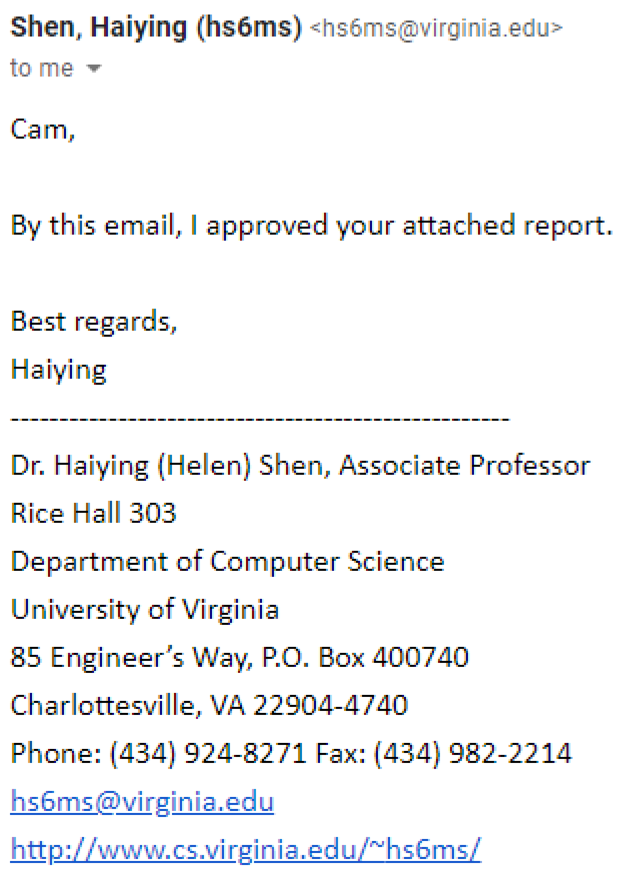
By

Cameron S. Dorsch

Fall, 2020

Signed: ___________CSD_______________ Date: 12/12/2020
Cameron Dorsch

Approved: _____Haiying Shen______________ Date: 12/14/2020
[approval on pp. 2]



Shen, Haiying (hs6ms) <hs6ms@virginia.edu>
to me

Cam,

By this email, I approved your attached report.

Best regards,
Haiying

Dr. Haiying (Helen) Shen, Associate Professor
Rice Hall 303
Department of Computer Science
University of Virginia
85 Engineer's Way, P.O. Box 400740
Charlottesville, VA 22904-4740
Phone: (434) 924-8271 Fax: (434) 982-2214
hs6ms@virginia.edu
http://www.cs.virginia.edu/~hs6ms/

The technical research of this work, completed as an Independent Capstone course, consisted of training various reinforcement learning (RL) models in a simulated resource management environment in order to test the potential run-time efficiency increase of RL models in resource management. Researchers such as Rolik et al. (2018) and Cheng, Li, & Nazarian have shown that similar RL methods could reduce the overall power consumption and runtime of systems with other modern task-scheduling algorithms by up to 320% and 144%, respectively (Cheng, Li, & Nazarian, 2018, pp. 238). This work, expanding on the work of MIT researchers Mao et al. (2016), tests the runtime improvement of asynchronous advantage actor-critic (3AC) models in the simulated environment created by Mao et al. to test Policy Gradient models.

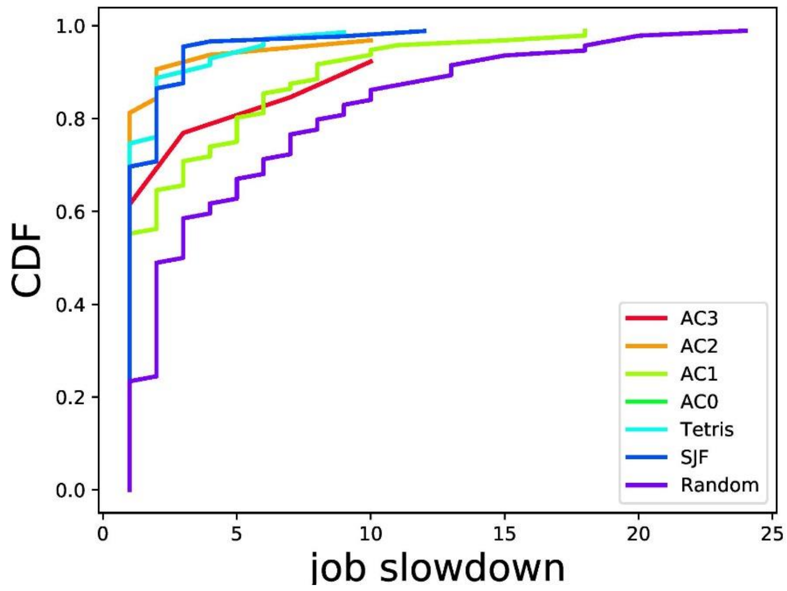
EXPANSION ON PREVIOUS WORK

The original work of Mao et al. trains a policy gradient model in which sampled rewards are used to tune the actor. Since the environment uses a negative reward mechanism, any penalties which an action incurs are totaled and returned to the agent, which then updates its network accordingly with a policy gradient. Policy gradients update network parameters with respect to the change in expected cumulative reward as by the equation $\nabla E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r_t] = E_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q^{\pi_\theta}(s,a)]$ (Mao et al, 2018, pp. 51). Here Q^{π_θ}(s,a), named a Q-value, represents the expected cumulative reward of choosing an action *a* in a state *s*. In a traditional policy gradient method, this is empirically determined by observing trajectories of the agent in the environment.

The implementation of 3AC in the environment attempted to improve on this in two main ways. First, the variance of the Q values used in the update step can be reduced by implementing a predictive agent for the average value of a state, V(s), in addition to empirically sampling trajectories. Second, the Q-value can be improved to better represent how close to optimal a reward is. With a learned predictor for V(s), the advantage function A(s,a) = Q(s,a) – V(s) can be used to calculate how good an action a_i was compared to the average action in a state *s*. In order to test the success of the new model, the reward mechanism of the environment created by Mao et al. was tuned in order to find the best possible models for runtime efficiency.

REWARD TESTING

The resource management environment used in training responds to actions with three different types of penalties: delay, hold, and miss. These punish the model for stopping a currently running task, not scheduling a new task, or missing additional tasks due to a full job queue. These mistakes will affect the runtime most greatly for short tasks, since the added time created by the bad action will be a higher proportion of the time needed to complete the task in isolation. The reward mechanism given by Mao et al. encapsulates this by dividing penalties by the job length. To improve on this mechanism, different reward variants such as squaring and cubing the previous function were tested and compared to the efficacy of the original reward function. After the squared reward function was chosen as the best variant, the weights of the three penalties were tuned to see which penalties most heavily encourage runtime efficient behavior. As shown from the results to the left, one tuning led to a model which improved over every heuristic in testing;



training with hold penalty = -3 and all other penalties = -1 resulted in a model slightly better than both an SJF strategy and Tetris scheduling.

WORKS CITED

Cheng, M., Li, J., Nazarian, S. (2018). DRL-Cloud: Deep reinforcement learning-based resource provisioning and task scheduling for cloud service providers. *2018 23rd Asia And South Pacific Design Automation Conference (ASP-DAC)*, 129-134. doi:10.1109/ASPDAC.2018.8297294

Mao, H., Alizadeh, M., Menache, I., Kandula, S. (2016). Resource Management with Deep Reinforcement Learning. *HotNets '16: Proceedings of the 15th ACM Workshop on Hot Topics in Networks,* 50-56. doi:10.1145/3005745.3005750

Rolik, O., Zharikov, E., Koval, A., Telenyk, S. (2018). Dynamic management of data center resources using reinforcement learning. *14th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET)*: 237-244. doi:10.1109/TCSET.2018.8336194