Machine Learning Assignment 2

Guo An Liew s3895776
Dion Tartaglione s3239216

COSC2673/COSC2793 Semester 1 2023
Lecturer: Azadeh Alavi
Due: 16/05/2023

Contents

1. Introduction

Scheduling is a common optimization problem in operating systems which when implemented correctly, can lead to significant improvements in CPU optimization and reduced costs. This study aims to evaluate the performance of a reinforcement-learning-based scheduling algorithm as a practical solution to the scheduling problem.

2. Method

Markov Decision Process is a fundamental approach to model decision-making in optimization problems[1]. As the scheduler should not need to observe which queue it took packets from previously to decide packets to take in the current state, a Markov Decision Process learner is well suited for the packet scheduling problem.

Model free vs model based. Schedulers are closer to being model based since they make decisions based on a queue. A FIFO scheduler retrieves the packet that arrived first in any queue. We believe implementing a model-based reinforcement learner is most appropriate.

The environment of the reinforcement learner will be constructed using Open AI gymnasium, given the thorough documentation and familiarity with the platform.

The scheduling algorithm utilizes the reinforcement learning model Q-learning to enable the learning process. Q-learning generates the estimated optimal policy for the scheduler through trial and error, thus granting the scheduler the "ability to learn."

The Q-learning algorithm creates a policy to achieve the highest maxima reward. Scenario 1 possesses an optimal policy while scenario 2 can only be estimated. The reward is based on how the scheduler achieves the expected mean delays for priority queues while minimizing the mean delay for the best effort queue. The reward function is outlined in **the reward table** [7.1].

A configurable parameter is set to define the margin of delay within which the mean delay is to be met. For scenario 2, the reward is decreased by 1 to discourage frequent switching since switching queues increases the average delay of all queues.

2.1. Independent evaluation

Using *similar* data on varying conditions on schedulers, the evaluation will be based on the real-world performance of our algorithm to other schedulers based on the reward function.

2.2. Random scheduler

Why compare to random? At worst, the reinforcement learner would perform inconsistently like a random scheduler. Comparing the rewards and queue state of the random scheduler and the reinforcement learner will determine if the reinforcement learner is performing better than random.

How many samples are enough? I will only evaluate three samples to obtain a simplistic view of the performance of the random scheduler.

2.3. Earliest deadline first [2]

Why compare to earliest deadline first? The earliest-deadline-first scheduler is an optimal scheduling algorithm for efficient utilization of resources. The evaluation of the performance of the reinforcement scheduler in a simulated real-world scenario involves comparing it to EDF.

The following steps outline the execution of the EDF scheduling algorithm*:*

- The deadline is predefined for each queue initially.
- Select the action based on the smallest deadline among all the queues.
- Increase the deadline for any queue based on the reallocation criteria.
- Repeat steps 2 and 3 until termination by the algorithm.

When reallocating the deadline, if the delay of a queue is smaller than or equal to mean delay, the deadline is set farther away proportional to the difference between the delay and the mean delay. Otherwise, if the delay is larger than mean delay, the deadline is increment normally.

The performance of the reinforcement learner will be evaluated by directly comparing the rewards and queue state to those of the EDF scheduler. In scenario 2, the deadline can be adjusted such that switching to other queues is delayed.

2.4. Policy function scheduler

The maximum reward can be obtained by predefining the actions according to the mean queue delay. Although the policy function scheduler is expected to perform the best, the reinforcement scheduler may learn unexpected strategies and obtain better rewards. Evaluating the difference compares the optimization of reinforcement learners to predefined schedulers.

The following steps outline the execution of the policy function scheduling algorithm**:**

- The mean delay for queues 1 and 2 is reduced until below expected delay plus margin.
- The best effort queue is chosen.
- Step 2 is repeated until mean delay for queues 1 and 2 exceeds the expected delay plus margin, at which point the algorithm proceeds to step one.

In scenario 2, it is important to consider future rewards to avoid frequent switching. Therefore, the reinforcement learner is expected to learn more accurately than the policy function scheduler. The scheduler can minimize frequent switching by ensuring that the mean delay of the current queue is sufficiently below the expected mean delay for the respective queue before switching.

3. Results

Outlined in Appendix 7.2, the input variables used were constants for determining the average delay amounts for each of the queues. A maximum queue size of 10 with a max delay of 30 timeslots enabled a suitable and reasonable size for running experiments of 50 timeslots.

Through comparing the mean average delays as seen in Appendix 7.3, the earliest deadline kept the priority queues below the mean delay requirements.
The priority scheduler managed to bring the best effort average delay o 0, which occurs if the expected mean delays for the priority queues are too high and packet arrivals too low.

The reinforcement model performs similarly to the earliest deadline, but it did not optimize the best effort results as well.
The random scheduler has the most even distribution of average delays between all queues, which we compare to the distribution of the reinforcement model.
Because rewards are based on how small PQ1 and PQ2 can get, if the average delays are distributed in a smaller region for PQ1 and PQ2, it informs us of that the reinforcement model prioritizes optimally, as opposed to the random scheduler.

4. Discussion

In the design of the scheduler, practical issues arise due to the presence of three queues with a maximum size and delay for each packet. The packet delay is an integer value which was done to discretize the states.

Initially, a Q-Table was created to account for all possible combinations of maximum delay and queue length, resulting in a complexity of approximately 2 ^ 60 combinations for a maximum delay of 100 and max queue length of 10, which was deemed too complex.

As a solution, mean delays of the queues were used instead, which were directly incorporated into the reward function and discretized by rounding to an integer value. While some information is lost, such as the length of the queue and the delay of each packet, this simpler approach is expected to perform better according to the principle of Ockham's Razor.

The environment is initialized with small values for each delay, ranging from 0 to 5, to ensure that the learner fits the starting values region. It is unlikely that the table reaches large regions given the parameters, such as [0,0,100], a delay of 100 in the best effort queue and 0 in other queues.

In developing the Q-Learning algorithm, hyper parameter tuning was conducted on the learning rate, discount rate and epsilon values. Through our implementation there were no significant improvements to the average reward for differing hyper-parameters. A sample of the trends are examined in Appendix 7.4.

5. Conclusion

In theory, the reinforcement learner can perform scheduling well. It is very difficult to put into practice due to common machine learning problems. Among these problems, generalization was the most difficult to handle, as shown in Appendix 7.5. Other schedulers are therefore more appropriate for practical use. At the time of writing, there is a lack of models in reinforcement learning that can compare to the simple design of practical schedulers.

A more thoughtful and thorough approach to tuning, generalizing, and sampling is necessary and recommended for the reinforcement learner. In addition to this, the implementation of Scenario 2 should be put into practice as the optimal policy generated is unknown. Therefore, scenario 2 is more suited as context for machine learning problems.

6. References

1. Velay, M. (2022, August 17). Reinforcement Learning Intro: Markov Decision Process. Towards Data Science. Retrieved Monday 15 May 2023. https://towardsdatascience.com/reinforcement-learning-intro-markov-decision-process-c73a030f045d

2. Datta, S, (April 21, 2023), *Scheduling: Earliest Deadline First*. Retrieved Saturday 13 May 2023

https://www.baeldung.com/cs/scheduling-earliest-deadline-first

7. Appendices

7.1 Reward Table

Mean Delay of queue	**Priority Queue 1**	**Priority Queue 2**	**Best Effort**
Mean delay of the queue is above the expected mean delay	0	0	-
Mean delay is within a margin of the expected mean delay	-1	-1	-
Mean delay is below the expected mean delay	-1	-1	-
Priority 1 and 2 is within or below expected mean delay margin, for respective queues	-	-	1
Priority 1 or 2 is above the expected mean delay margin, for respective queues	-	-	-1

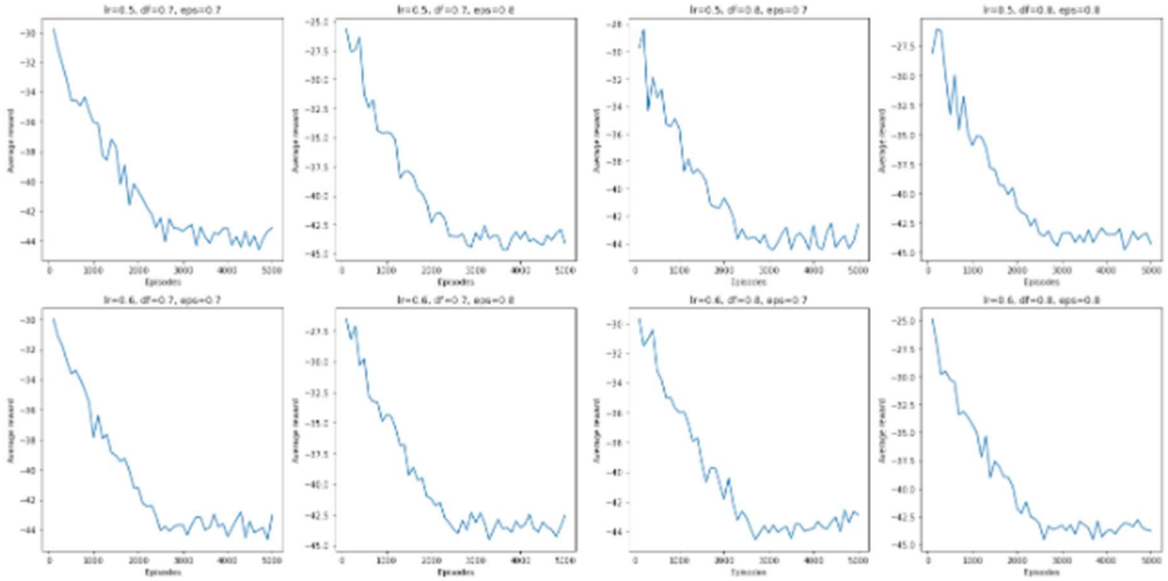
7.2 Table of Input variables

	Priority Queue 1	**Priority Queue 2**	**Best Effort**
Expected mean delay	6	4	-
Packet arrival rate	0.3	0.25	0.4
Margin of delay	1		
Maximum queue size	10		
Maximum delay	30		

7.3 Table of Average mean delay results

Scheduler	**Priority Queue 1**	**Priority Queue 2**	**Best Effort**
Random	6	0	2.33
Earliest deadline	3	2	7.29
Priority	7.75	6	0
Reinforcement	3	2	12.4

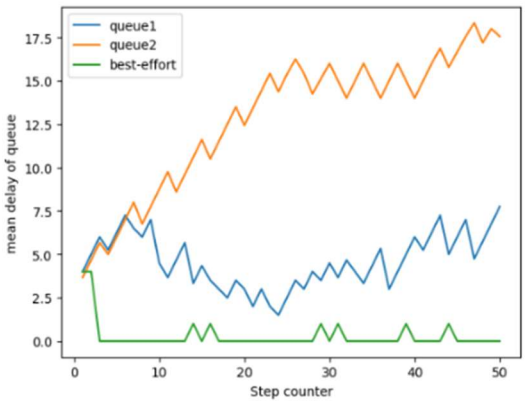
7.4 Hyper-parameter tuning for Q-Learning

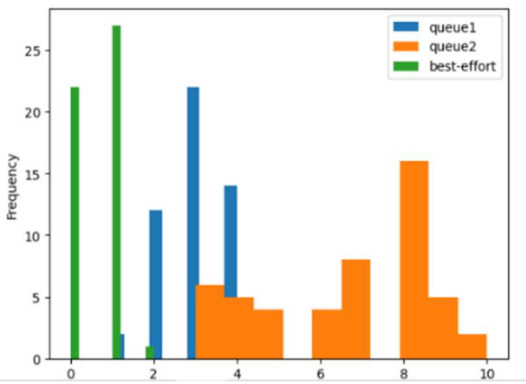


7.5 Reinforcement Scheduler Plot and Histogram

```
Bad seed: 72, score = -40
```

```
1 reinforcement_scheduler_plot(env, Q, seed = seed)
2 reinforcement_scheduler_hist(env, Q, seed = seed)
```






```
Best seed: 152, best_score = -9
```

```
1 reinforcement_scheduler_plot(env, Q, seed = best_seed)
2 reinforcement_scheduler_hist(env, Q, seed = best_seed)
```