Summary and Refinement Plan for Flow-Fair Scheduling Implementation

CMU

August 29, 2019

This white paper summarizes the pros and cons of our current implementation, as well as a plan to refine it to a fully distributed implementation.

1 Current Implementation Summary

In short, our current implementation of the environment for observing network conditions under our chosen protocols and the algorithm for choosing protocols is not a fully distributed framework as we planned several months ago (refer to our first white paper for this direction in ffs-01-cmu-2019.pdf). The reason of not sticking to the original plan is due to difficulties of generalizing OpenAi gym API to realize our plan. Our concern about the current implementation is its scalability. We elaborate it in the following.

First of all, the environment script that inputs the observation and feedback to the algorithm script a2c.py in each time step can only take the following input arguments: a vector of observations, a scalar reward, a dictionary of probability transition matrix, and a binary "done" (whether the episode is done). See https://github.com/openai/gym/blob/master/gym/core.py for the definition of gym environment class. We use the observation vector to represent the total available bandwidth capacity of our flows *on each link*. However, the reward cannot be a vector. Therefore, to reflect whether the currently chosen protocol works well, we can only assign one constant as a "score" of the entire vector of actions (protocols) inputted from the algorithm script. This restriction on the data structure of reward also makes sense – Suppose that the environment outputs a vector of rewards to represent the transmission rate achieved *on each link*. Then the algorithm cannot translate this vector into the intuition of whether the protocols are good or bad, because the algorithm A2C selects the actions for the entire network (or a path), and it has no intelligence to check the individual element of the reward vector in order to adjust the protocol for each link. In this sense, the algorithm views a vector of protocols (one for each link) as a super action, and a scalar of reward serves as the score of the super action. Therefore, the scalability issue results from the fact that there would be an exponential number of actions.

2 A Plan of Fully Distributed Implementation

A potential way to implement a fully distributed interaction between the environment and A2C algorithm is to associate a pair of algorithm and environment independently for each link and run a set of pairs over all links in parallel. In this way, for each link, the corresponding algorithm will check the scalar reward only for this link, and each environment also only reads the protocol chosen for the associated link.

However, it raises the challenge: there are also some dependency between different environments: they should have the same flow size if the environment is associated with the link that the current flow traverses; the flow completion time calculation function should check the flow time and update the remaining flow size etc for all links that the flow traverses. How to add this dependency as arguments to different environments? Nevertheless, it is possible to utilize the gym packages and our written environment based on the Ray framework (https://ray.readthedocs.io/en/latest/index.html). Using Ray, one can implement concurrent and multiple environments interacting with the algorithms for choosing protocols in one or more processes that run in parallel. Built on top of Ray, the package RLlib (https://ray.readthedocs.io/en/latest/rllib.html)is an open source library that can support multi-agent reinforcement learning and vectorized environment.

There are potentially two ways to implement the distributed protocol selection algorithm. (i) Vectorized gym environment (please refer to the definition of the vectorized environment at `https://github.com/ray-project/ray/blob/master/rllib/env/vector_env.py`). Our written environment (`https://github.com/ChelseaZhang90/gym-flowsched/blob/master/gym_flowsched/envs/flowsched_env.py`) can be used as the base environment to be vectorized; each environment with index i is associated with link i in the network. Then, the algorithm A2C can output a vector of protocols (actions) for all links and the i^{th} environment takes the i^{th} action from A2C. *However, so far it is unclear to us how to assign different features (e.g., a constant flow size assigned to the links that the flow traverses but zero flow size to other links) to different environments, as there seems no way to define different input arguments for different environments other than the actions.* (ii) Multi-agent reinforcement learning (RL) framework. A detailed tutorial on how the multi-agent RL is designed using RLlib can be found at `https://bair.berkeley.edu/blog/2018/12/12/rllib/`. We think this framework is more suitable for our setting, which however requires significant time to develop as it is much more different than the using basic gym package.