Learning Flow Scheduling

Asif Hasnain
Department of Computer Science
Paderborn University
Paderborn, Germany
asif.hasnain@uni-paderborn.de

Holger Karl
Department of Computer Science
Paderborn University
Paderborn, Germany
holger.karl@uni-paderborn.de

***Abstract*—Datacenter applications have different resource requirements from network and developing flow scheduling heuristics for every workload is practically infeasible. In this paper, we show that deep reinforcement learning (RL) can be used to efficiently learn flow scheduling policies for different workloads without manual feature engineering. Specifically, we present LFS, which learns to optimize a high-level performance objective, e.g., *maximize* the number of flow admissions while meeting the deadlines. The LFS scheduler is trained through deep RL to learn a scheduling policy on continuous online flow arrivals. The evaluation results show that the trained LFS scheduler admits 1.05× more flows than the greedy flow scheduling heuristics under varying network load.**

***Index Terms*—Flow scheduling, Deadlines, Reinforcement learning.**

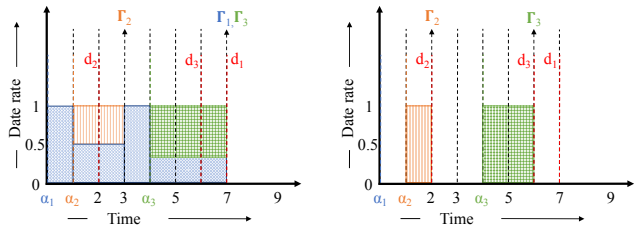
I. Introduction

The performance of datacenter applications significantly depends on different aspects of networking, e.g., flow scheduling [1]–[3]. Flow scheduling is the task of allocating resources, e.g., buffer size or data rate, to flows; we also subsume admission control, i.e., admitting or rejecting flows, under this term. In most non-trivial scenarios, flow scheduling is NP-hard [4]. Hence, flow scheduling is often performed using heuristics, which are optimized for a specific workload. These heuristics are developed by manually identifying features in network structure or flow arrival patterns, which is a time-consuming activity with long turn-around time. This problem is aggravated when workload changes render such hand-crafted heuristics no longer useful or changing performance objectives (e.g., *minimize* average flow completion times [3] or *maximize* the number of flow admissions while meeting flow deadlines [3], [5]–[7]) make them less suitable.

To address this challenge, in this paper, we attempt to *automate network resource management*, specifically, flow scheduling for deadline-sensitive flows. We present a new framework, named learning flow scheduling (LFS), for learning flow scheduling policies. Like many input-driven applications [8], [9], LFS learns flow scheduling policies through deep reinforcement learning (RL). Deep RL is well-suited for such learning because it automates learning Markov structure in flows through end-to-end training on network states. It is based on a neural network [10], which is trained to learn a scheduling policy by directly interacting with the environment. We focus here on the flow admissions aspect. That means that the policy takes an action to admit or reject new flows, arriving online in different network states. On each action, the environment produces a reward as feedback for the policy to know how well it is doing on flow admissions. The rewards are based on a workload-specific performance objective (i.e., *maximize* the number of flow admissions and meet deadlines for time-sensitive datacenter applications, e.g., web search or social networking). The flow scheduler, after a flow has been admitted, assumes no preemption and an admitted flow is assigned a constant data rate from the time a flow starts executing till its completion. The data rate assigned to flows is computed by a greedy heuristic within the environment.

One of several challenges in training a policy is learning to make decisions on stochastic flow arrivals to a network. The stochastic flow arrivals make it difficult for a policy to learn from rewards because the successive arrival of smaller flows in an input can produce higher rewards on admission than large flows for this objective. Consider, for instance, three competing flows in Table I, i.e., f_1, f_2, and f_3 (see Table II for notation). These flows have different arrival times, individual sizes and deadlines. They are competing for resources on a single link with total data rate of 1 unit. When flow f_1 (blue in Fig. 1a) arrives first at time 0, the *non-preemptive* D^3 [7] scheduler assigns, at least, minimum data rate to flow f_1 to finish within its deadline (7). However, D^3 *hogs* the link's resource by admitting flow f_1, which has a large deadline. Consequently, two flows f_2 (orange) and f_3 (green) fail to meet their deadlines because both flows require, at least, 1/1 and 2/2 units of link resource, respectively, to finish within their deadlines. But the remaining data rates at time 1 and 4, after minimum allocation to active flow f_1, are 1/2 and 2/3 units, respectively. Unlike the myopic D^3 scheme, the optimal schedule (in Fig. 1b) *rejects* large flow f_1 at time 0 because it can admit more, smaller flows, i.e., f_2 and f_3, for the desired objective (*maximum* the number of successful flows). It commits, at least, minimum data rates to two flows, i.e., $r_2 = 1$ and $r_3 = 1$, from their arrival times to finish within the

This work is supported by the German Research Foundation (DFG) within the Collaborative Research Center On-The-Fly Computing (SFB 901).



(a) D^3 (b) Optimal schedule

Fig. 1. Different flow scheduling schemes for flows in Table I, where (a) D^3 admits and meets deadline for only flow f_1 while (b) LFS admits (and meets deadline) for flows f_2 and f_3.

deadlines. Such an optimal schedule can be achieved if LFS scheduler effectively learns flow arrivals pattern in network states.

TABLE I
COMPETING FLOWS ON A LINK

Flows	Arrival time	Size	Deadline
f_1	0	4	7
f_2	1	1	1
f_3	4	2	2

As a first step to learn a flow scheduling policy, we describe the model in Subsection II-A. The LFS scheduling problem is formulated as a Markov decision process (MDP); specifically, the average reward formulation for continuous flow scheduling tasks is described in Subsection II-B. The LFS scheduler uses a policy gradient algorithm for end-to-end training on a network environment (Subsection II-C). It is evaluated on custom traces, which are generated by sampling flow information (i.e., arrival, size, and deadline) from different distributions. Our preliminary results (in Section III) show that the trained LFS scheduler outperforms greedy heuristics under varying network loads. Interestingly, the LFS scheduler learns a policy to admit more smaller flows than long flows for the specified performance objective (i.e., *maximize* the number of flow admissions while meeting deadlines). Moreover, it is flexible enough to quickly adapt to various performance objectives by using different rewards.

II. DESIGN

A. Model definition

We consider a single network link l and, like prior work [2], assume that the information about a new flow $l_f \in F$ is known at arrival time α_f (see Table II for notation). That information includes data volume v_f and deadline d_f relative to the arrival time. If an arriving flow f is admitted, its assigned data rate r_f is calculated using a greedy heuristic and it is constrained by the data rate of link C_l. The heuristic simply divides flow size v_f by the remaining time to deadline d_f. It is kept simple to learn the scheduling policy for higher flow admissions (Section II-B). We further assume that active flows $\bar{\mathbb{F}}_e$ are not preempted and they continuously receive link resource, i.e., data rate, from the time a flow starts execution till completion.

TABLE II
NOTATION FOR FLOW AND SYSTEM MODEL

t_e	Time of the scheduling event $e \in [1, \cdots, E]$
F	Set of arrived, online flows
$\mathbb{F}$	Set of admitted flows $\mathbb{F} \subseteq F$, which comprises active and completed flows
$\mathbb{F}_e$	Set of active flows $\mathbb{F}_e \subseteq \mathbb{F}$ at time t_e, not including a flow arriving at t_e
l_f	A new flow f request on link l (the event e causing that new flow will be clear from context)
α_f	Arrival time of flow f
d_f	Deadline of flow f relative to flow arrival α_f
v_f	Total data volume of flow f
$\bar{v}_f$	Remaining data volume of flow f
r_f	Assigned data rate to flow f
Γ_f	Flow completion time (FCT) of completed flow f
C_l	Data rate for link l

B. RL model

TABLE III
NOTATION FOR LEARNING MODEL

M	Number of different flow sequences as defined in Table II
F_i	The ith set of flow arrivals, where $i \in \{1, \cdots, M\}$ and F_i corresponds to F in Table II
E_i	Number of scheduling events in F_i, where E_i corresponds to E in Table II
λ	Arrival rate of flows
τ	Episode length
N	Number of different sample trajectories
s_e	Observed state at time t_e
$a_e \in \{0, 1\}$	Action taken at time t_e
R_e	Reward (or penalty) received on action a_e in state s_e
$\bar{R}_e$	Time-average reward at time t_e
$R_e - \bar{R}_e$	*Differential* reward
G_e	*Differential* return (sum of *differential* rewards from state s_e to the terminal state s_E)
$\pi_\theta(a_e \vert s_e)$	Policy network, learnt by the actor
θ	Parameters of the policy network $\pi_\theta(\cdot)$
β_θ	Learning rate (or step size) of policy network $\pi_\theta(\cdot)$
$V_{v_i}(s_e)$	State-value function for the flow arrival sequence F_i (critic)
v_i	Parameters of the state-value network $V_{v_i}(\cdot)$
β_v	Learning rate of state-value network $V_{v_i}(\cdot)$
δ_e	Error in estimation of *differential* return G_e
$\mathbb{P}(s_{e+1} \vert s_e, a_e)$	State transition probability function
$J(\theta)$	Performance objective for policy network $\pi_\theta(\cdot)$
$\nabla_\theta J(\theta)$	Policy gradient

We consider a discrete-time Markov decision process (MDP), which is defined as a sequence of state, action, and reward, i.e., (s_e, a_e, R_e), at time t_e for each of the scheduling events $e \in [1, \cdots, E]$ (see Table III for notation). The problem is solved using a policy gradient algorithm of deep RL, where a flow scheduling agent interacts with a single-link l environment. The scheduling agent can fully observe the state s_e at time t_e, i.e., information about active flows $\bar{\mathbb{F}}_e$ on link l and the new flow request f is available. A flow departure is processed within the environment and not made visible to the RL agent; if one of the active flows $\bar{\mathbb{F}}_e$ completes, it is removed from the network. On each scheduling event e, the scheduling agent takes an action a_e in state s_e, collects a reward R_e from the environment, and shifts to the next state

s_{e+1}, where the next scheduling event $e+1$ occurs at time t_{e+1}. Specifically, the state transition is assumed to satisfy the Markov property, i.e., the new state s_{e+1} depends only on the current state s_e and the action a_e taken at time step t_e.

Since the problem space is large and continuous, the scheduling agent uses a neural network [10] to learn the policy $\pi_\theta(a_e|s_e)$, where θ are parameters of a *policy network*. The policy $\pi_\theta(a_e|s_e)$ is defined as the probability of taking action a_e in state s_e with parameters θ. After each action a_e, the environment provides a *differential* reward $R_e \leftarrow R_e - \bar{R}_e$ (Section II-B) to the scheduling agent, where $\bar{R}_e$ is the time-average reward at time t_e. The scheduling agent uses the reward as signal to improve the policy $\pi_\theta(a_e|s_e)$. The reward is based on a higher-level performance objective $J(\theta)$ to *maximize* the number of admitted flows while meeting their deadlines.

The scheduling agent trains the policy network $\pi_\theta(a_e|s_e)$ through the *REINFORCE algorithm with baseline* (sometimes, also called *Monte-Carlo (MC) actor critic*) [11]. Although the algorithm is unbiased, it has high variance in policy gradient. The variance is caused by single-sample estimate and stochastic flow arrivals, which impede effective learning of a scheduling policy. The variance is usually reduced by subtracting a state-value function (critic) as baseline [12] from the actual *differential* return, where the state-value function estimates the *differential* return.

However, a single state-value function as baseline is proven ineffective [9] to estimate *differential* return in the presence of different flow arrivals because training on different flow arrivals adds noise to the reward and makes it difficult to estimate *differential* return using a single state-value function. One way [9] to effectively estimate *differential* return, with different flow arrivals, is to train a separate state-value function $V_{v_i}(s_e)$ for each flow arrivals sequence $F_i, i \in \{1, \cdots, M\}$, where v_i are parameters of a state-value network and M is the total number of flow sequences. These multiple state-value functions act as critic to *evaluate* the scheduling policy $\pi_\theta(a_e|s_e)$ and provide feedback to the policy network (which is also called an actor).

Objective. The high-level performance objective for scheduling policy is to *maximize* the number of flow admissions while meeting their deadlines. The objective is defined by an average-reward formulation because flow scheduling is a continuous task (Section II-B). Specifically, the environment gives more reward for flows with smaller flow completion time (FCT) than larger FCT to achieve higher flow admissions.

State space. It represents the fully observed state s_e at time t_e of a particular scheduling event e, i.e., a flow arrival. The state information is a flat feature vector of active flows $\bar{\mathbb{F}}_e \subseteq \mathbb{F}$ and the new flow request f on link l, where the maximum number of concurrent, active flows is configured to 50 (Subsection III-B1). The feature vector is passed as an input to the policy network for learning flow structures.

- Each active flow $f \in \bar{\mathbb{F}}_e$ has
  - its remaining data volume $\bar{v}_f$
  - its remaining time to deadline $\alpha_f + d_f - t_e$
  - its assigned data rate r_f
- For data rates of active flows on link l, it always holds that $\sum_{f \in \bar{\mathbb{F}}_e} r_f \leq C_l$.
- A new flow request f with an arrival time α_f, a deadline d_f, and a data volume v_f

Action space. It is a discrete set $a_e \in \{0, 1\}$, where the actions $a_e = 1$ and $a_e = 0$ represent the decision to either admit or reject the flow request f in state s_e, respectively. The decision is taken by the flow scheduling policy.

Actor. The actor directly learns a *softmax paramerterized policy* $\pi_\theta(a_e|s_e)$, which outputs probability distribution over all actions A in state s_e with parameters θ. The action $a_e \in A$, to admit or reject a new flow in state s_e, is then sampled from action probabilities using Gumbel-Softmax distribution. The actor updates policy parameters θ via gradient descent in the direction (i.e., gradient) suggested by the critic (Section II-B). It receives a feedback from critic, on the performance of its current scheduling policy, in the form of an estimated error δ_e (sometimes, also called an *advantage*). The actor uses the estimated error δ_e to update action probabilities such that it reaches high-valued states with more flow admissions and attempts to keep error δ_e positive (i.e., collect better-than-time-average reward). Specifically, it computes the policy gradient $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a_e|s_e)\delta_e$ to update policy parameters such that $\theta \leftarrow \theta + \beta_\theta \nabla_\theta J(\theta)$, where β_θ is the constant learning rate (or step size) for policy parameters θ. The policy gradient $\nabla_\theta J(\theta)$ on performance objective $J(\theta)$ increases probability of taking action a_e in state s_e if the action (e.g., to admit a flow) resulted in higher *differential* return than the estimated return by critic.

Critic. The critic *evaluates* the scheduling policy using multiple state-value networks as baseline. The state-value network $V_{v_i}(s_e)$ estimates the *differential* return G_e in a particular flow arrival sequence F_i, where $G_e = R_e - \bar{R}_e + R_{e+1} - \bar{R}_{e+1} + \cdots$ is the sum of *differential* rewards from state s_e to the terminal state s_E and F_i is the ith set of flow arrivals in M sequences, i.e., $i \in \{1, \cdots, M\}$. Intuitively, the critic has a separate state-value function $V_{v_i}(s_e)$ for each flow arrival sequence F_i to reduce variance of policy gradient from different flow arrival sequences. It criticizes the actor's action, based on actor's policy, by sending an error δ_e to actor. The critic computes error δ_e from the *differential* return G_e (as per actor's policy) and its estimation of value of (being in) current state $V_{v_i}(s_e)$. It is given by $\delta_e = G_e - V_{v_i}(s_e)$, where a positive error δ_e means that the actor's action was good (so should be repeated) because it led to better-than-time-average reward. On the contrary, a negative error reflects worse-than-time-average reward, which means the actor should avoid this, bad action. The state-value network $V_{v_i}(s_e)$ uses *differential* return G_e from policy to improve its predictive accuracy. It does so by reducing the magnitude of loss in estimated return close to zero, where the loss function is mean squared error (MSE).

Average reward. Since the flow scheduling problem is a continuous task — the interation between flow scheduling agent and link environment goes on forever — an average reward is better than the total reward and it maximizes $\lim_{\tau \to \infty} 1/\tau \sum_{e=0}^{\tau} R_e$ [13], where τ is a training episode length. Specifically, the network environment rewards the

scheduling agent with a *differential* reward $R_e \leftarrow R_e - \bar{R}_e$, where $\bar{R}_e$ is the moving time-average reward at time t_e from all previous scheduling events of current and previous training episodes. The scheduling agent receives a reward for every action a_e in state s_e, where the reward function is designed as follows:

- We call an action (by scheduling policy) to admit a new flow f a *true positive* (TP) decision if it is indeed possible to assign enough rate to the flow f to meet its deadline. This can be tested immediately by checking flow deadline, volume, and currently available data rate. A TP decision for a new flow f, in the confusion matrix, returns higher reward for a flow with smaller FCT than a flow with large FCT, i.e., $R_e = 1/\Gamma_f + (1/\Gamma_f * 1/|\bar{\mathbb{F}}_e|)$, where $|\bar{\mathbb{F}}_e|$ is the number of active flows at time t_e.
- A *false negative* (FN) decision means that the scheduling policy could have admitted the new flow f in current state s_e but did not. It might be a correct action after enough learning, e.g., to reject a flow with large FCT, for this performance objective $J(\theta)$. A FN decision produces a penalty $R_e = -1/\Gamma_f$, where the actor is penalized more for rejecting a flow with smaller FCT than large FCT.
- An action is considered a *true negative* (TN) decision if the scheduling policy has learnt to correctly reject the new flow f as there is not enough link resource or flow cannot meet deadline. A TN decision has zero reward $R_e = 0$.
- The action to admit a new flow f is called *false positive* (FP) decision if the flow cannot actually be assigned sufficient rate to meet its deadline. A FP decision for a new flow f is penalized with $R_e = -(\Gamma_f + |\bar{\mathbb{F}}_e|)$.

C. Training algorithm

The scheduling policy is trained using algorithm 1. Since the initial scheduling policy is assumed poor, the earlier training episodes (or epochs) are terminated stochastically at time τ to help learning on online flow arrivals (line 4). However, the episode lengths are gradually increased during training (line 15). In each episode, the training algorithm rollouts multiple trajectories $j \in [1, \cdots, N]$ on current scheduling policy $\pi_\theta(a_e|s_e)$ (line 6), computes return from *differential* rewards (line 8), and estimates error in predicting return (line 9). Based on the estimated error and return, it updates parameters of the policy and critic networks, respectively (line 10–14).

Algorithm 1 Monte-Carlo Actor Critic Algorithm for Training

Input: Policy network $\pi_\theta(\cdot)$ and state-value networks $\{V_{v_1}(\cdot), \cdots, V_{v_M}(\cdot)\}$

1: $\theta, \{v_1, \cdots, v_M\}$ {Initialize parameters of policy and state-value networks with Glorot uniform initializer}
2: Initialize learning rates $\beta_\delta, \beta_v > 0$ and the time-average reward $\bar{R}_e = 0$
3: **for** each episode **do**
4: Sample an episode length τ from a geometric distribution
5: Sample a new set of flows F for each of the flow arrival sequences $F_i, i \in \{1, \cdots, M\}$
6: Rollout multiple trajectories $j \in [1, \cdots, N]$ on current policy $\pi_\theta(a_e|s_e) \sim \{s_1, a_1, R_1, \cdots, s_E, a_E, R_E\}$ until $t_e \leq \tau$
7: $\bar{R}_e \leftarrow \bar{R}_e + 1/(N \cdot E) \sum_{j=1}^{N} \sum_{e=1}^{E} R_e$; {Update time-average reward}
8: Calculate *differential* return in each state, i.e., $G_e = \sum_{e'=e}^{E} R_{e'} - \bar{R}_{e'}$
9: $\delta_e = G_e - V_{v_i}(s_e)$ {Calculate the error δ_e in estimation}
10: **for** $j \in [1, \cdots, N]$ **do**
11: $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a_e|s_e)\delta_e$; {Compute policy gradient using the error δ_e}
12: $V_{v_i} \leftarrow V_{v_i} + \beta_v \nabla_{V_{v_i}} V_{v_i}(s_e) G_e$; {Update parameters V_{v_i} of the state-value network $V_{v_i}(\cdot)$}
13: **end for**
14: $\theta \leftarrow \theta + \beta_\theta \nabla_\theta J(\theta)$; {Update parameters θ of policy network}
15: $\tau \leftarrow \tau + \epsilon$
16: **end for**

III. Evaluation

We evaluated *LFS* scheduler through a flow-level simulator as an environment on a machine with a 16-core CPU (Intel Xeon E5-2695) and 128 GB total memory. The highlights are:

- The RL-based flow scheduler learns to optimize the specified performance objective.
- It admits 1.05× more than flows than the greedy algorithm on an unseen test dataset.
- It consistently outperforms the competing schemes under varying network load (Section III-B2).

A. Methodology

1) Workload: Like prior work [2], we assume that the flow information is known at arrival and it includes data volume and deadline — relative to the arrival time. The flow interarrival time α_f is sampled from the exponential distribution, where the flow arrival rate is set to $\lambda = 1$. The flow arrival rate λ is varied during testing to create *network load* and compare performance of the LFS flow scheduler with competing schemes (Section III-B2). The LFS flow scheduler is, however, trained on a single flow arrival rate λ. Since the datacenter traffic has long-tailed distribution [14], [15], i.e., most of the flows have short size but majority of the data is transmitted by a few large flows, the flow size v_f is sampled from the Pareto distribution, where the pareto shape and scale is set to 2.0 and 100.0, respectively. In addition, the flow deadline d_f is a uniformly drawn value between 1 and 4 seconds because most flows in datacenter traffic last less than a few seconds [15].

2) Environment: We developed a flow-level simulator as an environment for the scheduling agent. The simulator is driven by flow *arrival and departure events*. Whenever a new flow arrives α_f, the scheduling scheme (i.e., LFS or the competing algorithm) either admit or reject the flow. The data rate r_f for

the flow is computed within the environment using a greedy heuristic. The heuristic simply divides flow size v_f by the deadline d_f. It is not work-conserving, i.e., the excessive data rate on link l is not distributed, after minimum resource allocation, to the newly admitted as well as active flows. Whenever one of the active flows completes, the simulator removes the flow from the network. The incomplete, active flows continue executing at the same, constant data rate.

3) Scheduling agent: The scheduling agent consists of a policy network and multiple state-value networks. These networks are multilayer perceptron (MLP) neural networks, where each network has 2 hidden layers. The two hidden layers comprise 200 and 128 neurons each and their activation function is set to ReLU. These networks use Adam optimizer to update their parameters, where the learning rates β_θ and β_v for parameters of policy and state-value networks are set to $7{\times}10^{-3}$ and $7{\times}10^{-3}$, respectively. In addition, the entropy value for *exploring* the scheduling policy is intialized to 1. However, it is gradually decayed during training of the policy network.

4) Metrics: We primarily measure the flow admissions by various scheduling schemes. In addition, we measure rewards and flow sizes to answer the following questions:

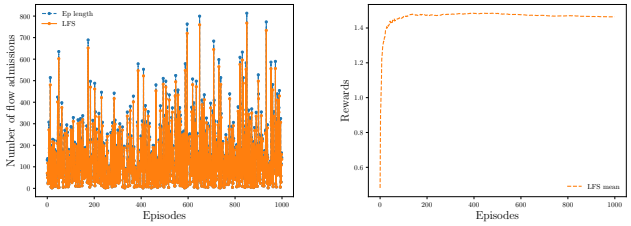
- Does the LFS scheduler learn a policy to optimize the performance objective (Section II-B)?
- What is the behaviour of trained LFS scheduler under varying *network load*?
- Do the admitted flows *meet their deadline*? We expect that all admitted flows finish within their deadline.
- Why does the LFS scheduler admit more flows?

B. Simulation results

1) Training of the LFS scheduler: The LFS scheduler is trained using algorithm 1. The algorithm runs 1000 episodes and, in each training episode, 1000 flows are generated to train the LFS scheduler. Each episode is, however, terminated at the maximum time τ, which is a sampled length from a geometric distribution. The reset probability of episode length $p = 1{\times}10^{-2}$ decays (with value $4{\times}10^{-6}$) until the minimum value $p = 5{\times}10^{-8}$ during training. In each episode, the training algorithm rollouts 6 parallel trajectories on the current policy and the *differental* reward is enabled, by default. In addition, the flow arrival rate λ is set to 1 and the maximum number of concurrent, active flows is configured to 50.

The result (in Fig. 2a) shows that the LFS scheduler learns a reasonable policy to admit flows. The Fig. 2b plots the average reward received during training of the LFS scheduler. As expected, the average reward increases with every training episode. However, after approximately 150 episodes, the average reward stops increasing, which indicates that the LFS scheduler has converged to a policy.

2) Network load: The trained LFS scheduler is compared with a greedy scheduling algorithm and its variants. The greedy variants schedule flows based on a value drawn from binomial distribution with different success probabilities, i.e., $p \in \{0.95, 0.9, 0.8, 0.7\}$. Since the arrival time α_f, data volume v_f, and deadline of flows are samples of different distributions, the test episodes are passed different seeds



(a) Total flow admissions (b) Average reward $\bar{R}$

Fig. 2. Measurement of flow admissions and average rewards during training of LFS scheduler

from training episodes to produce unseen, new samples. The arrival rate λ of flows is varied to produce a wide range of *network load*. For example, Fig. 3 shows flow admissions by various scheduling schemes at different flow arrival rates, i.e., $\lambda \in \{1, 2, 5, 10, 15, 20\}$. The result shows that the LFS scheduler admits more flows than the greedy schedulers, even under high network load. In addition, Fig. 4a shows average flow admissions over many test episodes at different network loads. As expected, we see higher flow admissions under a low network load because it is likely that a few flows are concurrently active. Each datapoint in Fig. 4a is an averaged value over 25 test episodes at a particular flow arrival rate λ. The dataset in each of the test episodes is unseen. We found that, for different network loads, the LFS scheduler admitted, on average, $1.05\times$ more flows than the best greedy scheduler G.

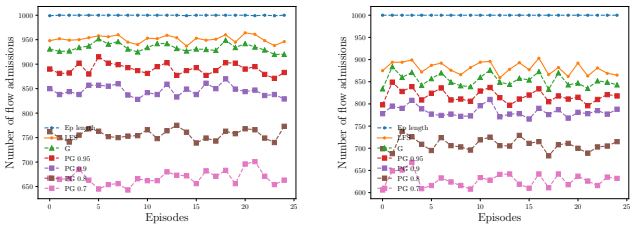
3) Number of flows that met their deadline: Since all schedulers are non-preemptive schedulers, admitted flows continue to receive link resources, i.e., data rates, from the time they start executing till their completion. During testing, we found that all flows met their deadline as per our expectation.

4) Performance Gain: We evaluate the gains of the LFS scheduler on a particular network load, in which the flow arrival rate is set to $\lambda = 5$. In the test, we record the size of both admitted and rejected flows and plot the cummulative distribution function (CDF). The CDF of flow sizes showed that the trained LFS scheduler mostly admitted smaller flows in the test. This is evident in Fig. 4b, which implies that the policy priortized smaller flows over large flows for designated performance objective and, if necessary, rejected large flows to admit not-yet-arrived smaller flows.

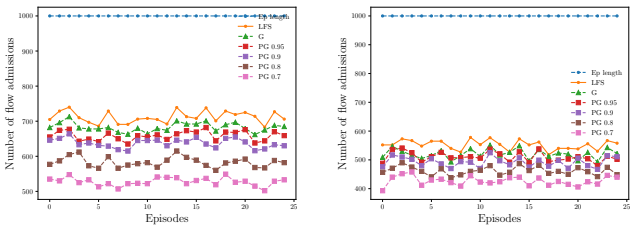
IV. Related Work

Preemptive flow scheduling The LFS scheduler differs from deadline-aware flow schedulers like PDQ [3], which approximates earliest deadline first algorithm, because it is a non-preemptive scheduler. In a recent proposal [5], an RL-based flow scheduler is trained to compute data rates of flows with deadlines but it uses a Q-learning lookup table, which is not a scalable approach. AuTO [6] automates traffic optimization in datacenter networks using deep RL but it also considers preemptive scheduling using strict priority queueing.

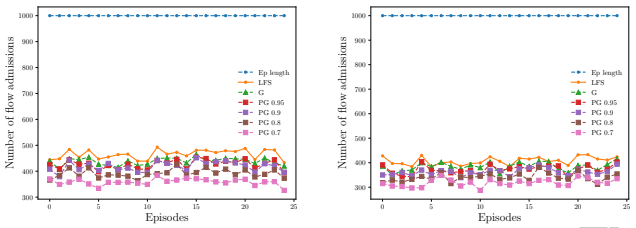
Job scheduling Many recent proposals [8], [9] use deep RL for job scheduling in computing but they are not directly



(a) Flow arrival rate $\lambda = 1$ (b) Flow arrival rate $\lambda = 2$

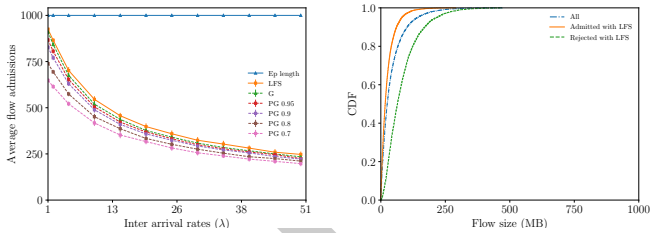


(c) Flow arrival rate $\lambda = 5$ (d) Flow arrival rate $\lambda = 10$



(e) Flow arrival rate $\lambda = 15$ (f) Flow arrival rate $\lambda = 20$

Fig. 3. (a)-(f) Some examples of flow admissions by various scheduling schemes under different network loads



(a) Average flow admissions (b) CDF of flow sizes

Fig. 4. (a) Average flow admissions under different network loads (b) Size of admitted and rejected flows by the LFS scheduler

applicable to flow scheduling in datacenter network. However, like in [8] for jobs, graph neural networks [16] can be considered to reduce feature space of active, concurrent flows in network state.

V. Conclusion and Future Work

We have demonstrated that the LFS scheduler automatically learns flow structures using deep RL and outperforms the greedy flow scheduling heuristics under varying network load. In addition, it is practically feasible to quickly adapt to different performance objectives by simply retraining the scheduling policy on the redesigned reward function. For example, with reward $R_e = -|\bar{\mathbb{F}}_e|$, LFS can learn a scheduling policy to *minimize the average flow completion time* or it can *maximize the network utilization* with reward $R_e = \sum_{f \in \bar{\mathbb{F}}_e} r_f$ at each timestep.

However, LFS scheduler is the first step to automate network resource management and we believe that the idea will encourage others to employ deep RL in different aspects of networking, e.g., coflow scheduling, routing, and congestion control.

References

[1] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with karuna," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM 16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 174–187. [Online]. Available: https://doi.org/10.1145/2934872.2934888

[2] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 435–446. [Online]. Available: http://doi.acm.org/10.1145/2486001.2486031

[3] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 127–138. [Online]. Available: http://doi.acm.org/10.1145/2342356.2342389

[4] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[5] D. Ghosal, S. Shukla, A. Sim, A. V. Thakur, and K. Wu, "A reinforcement learning based network scheduler for deadline-driven data transfers," in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.

[6] L. Chen, J. Lingys, K. Chen, and F. Liu, "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM 18. New York, NY, USA: Association for Computing Machinery, 2018, pp. 191–205. [Online]. Available: https://doi.org/10.1145/3230543.3230551

[7] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proceedings of the ACM SIGCOMM 2011 Conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/2018436.2018443

[8] H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM 19. New York, NY, USA: Association for Computing Machinery, 2019, pp. 270–288. [Online]. Available: https://doi.org/10.1145/3341302.3342080

[9] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=Hyg1G2AqtQ

[10] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*. USA: PWS Publishing Co., 1997.

[11] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, no. 34, pp. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[12] L. Weaver and N. Tao, "The optimal reward baseline for gradient-based reinforcement learning," in *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, ser. UAI01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 538–545.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[14] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC 10. New York, NY, USA: Association for Computing Machinery, 2010, pp. 267–280. [Online]. Available: https://doi.org/10.1145/1879141.1879175

[15] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social networks (datacenter) network," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM 15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 123–137. [Online]. Available: https://doi.org/10.1145/2785956.2787472

[16] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. [Online]. Available: https://arxiv.org/pdf/1806.01261.pdf