A Deep Deterministic Policy Gradient Based Network Scheduler For Deadline-Driven Data Transfers

Gaurav R. Ghosal[1], Dipak Ghosal[2,3], Alex Sim[3], Aditya V. Thakur[2], and Kesheng Wu[3]

[1]University of California, Berkeley, CA, U.S.A.
[2]University of California, Davis, CA, U.S.A.
[3]Lawrence Berkeley National Laboratory, Berkeley, CA, U.S.A.

***Abstract*—We consider data sources connected to a software defined network (SDN) with heterogeneous link access rates. Deadline-driven data transfer requests are made to a centralized network controller that schedules pacing rates of sources and meeting the request deadline has a pre-assigned value. The goal of the scheduler is to maximize the aggregate value. We design a scheduler (RL-Agent) based on Deep Deterministic Policy Gradient (DDPG). We compare our approach with three heuristics: (i) P_FAIR, which shares the bottleneck capacity in proportion to the access rates, (ii) V_D_Ratio, which prioritizes flows with high value-to-demand ratio, and (iii) V_B_EDF, which prioritizes flows with high value-to-deadline ratio. For equally valued requests and homogeneous access rates, P_FAIR is the same as an idealized TCP algorithm, while V_B_EDF and V_D_Ratio reduce to the Earliest Deadline First (EDF) and the Shortest Job First (SJF) algorithms, respectively. In this scenario, we show that RL-Agent performs significantly better than P_FAIR and V_D_Ratio and matches and in over-loaded scenarios out-performs V_B_EDF. When access rates are heterogeneous, we show that the RL-Agent performs as well as V_B_EDF even though the RL-Agent has no knowledge of the heterogeneity to start with. For the value maximization problems, we show that the RL-Agent out-performs the heuristics for both homogeneous and heterogeneous access networks. For the general case of heterogeneity with different values, the RL-Agent performs the best despite having no prior knowledge of the heterogeneity and the values, whereas the heuristics have full knowledge of the heterogeneity and V_D_Ratio and V_B_EDF have partial knowledge of the values through the ratios of value to demand and value to deadline, respectively.**

***Index Terms*—Deadline-driven data transfers, Software-defined Networking (SDN), Reinforcement Learning, DDPG, Scheduling heuristics, EDF, TCP, Value maximization**

I. Introduction

Large distributed science is moving to the SuperFacility model [1] in which distributed instrument facilities, HPC systems, storage, and researchers are viewed as one integrated facility. This is achieved by interconnecting these resources using a very high-speed network, such as the ESnet [2], that must provide performance guarantees on the data transfers required by complex science workflows deployed on the SuperFacility. Furthermore, due to exponential growth in the amount of data that is generated by the instruments, it is also important that the interconnecting network is run at a high utilization [3].

Next generation science workflows will require complex processing pipelines that may be processed at different HPC facilities [4]. More and more these workflows must be completed within a deadline. For example, the Large Synoptic Survey Telescope (LSST) [5] will take more than 800 panoramic images each night with its 3.2 billion-pixel camera, recording the entire visible sky twice each week. The new data will be compared with previous images to detect changes in brightness and position of objects as big as distant galaxy clusters and as small as near-by asteroids. The processing of each 30 second image must be completed within 7.5 seconds so that necessary alerts can be generated for down-stream processing. The deadlines in processing the flows impose deadlines in the data transfer from the instrument facilities (that are typically in remote locations) to the HPC facilities. Furthermore, the data transfer deadlines are becoming increasingly tight. For example, in the Linac Coherent Light Source (LCLS) [6], the location of next image is a function of the current image.

Deadline driven data transfer is not unique to large scientific workflows and is becoming important for cloud service providers with geo-distributed data centers. In order to keep the distributed locations synchronized, large data sets must be periodically transferred within a deadline [7], [8] while requiring that the network is operating at a very high utilization [3]. This requires fine grain control of the transmission rate to eliminate contention and resulting packet losses.

To support deadline-driven data transfers, ESnet sets up on-demand circuits [9] supporting packet priority allowing the circuit to be used by other traffic when the deadline flow is inactive. For cloud service providers, the deadline transfers are scheduled over private/dedicated WANs that interconnect the geo-distributed data centers. These private dedicated WANs are less noisy and more predictable compared to the general internet. Nevertheless, the deadline transfers must be scheduled in the presence of other background and interactive flows [7], [8].

In this paper, we consider an SDN-enabled pri-

ISBN 978-3-903176-28-7 2020 IFIP

vate/dedicated WAN with a centralized controller. There is also in-network telemetry (INT) that can provide real-time fine-grained information about network state including router buffer lengths and packet drops [10]. Deadline-aware data transfer requests are made to a central network controller, which schedules the flows by setting pacing rates at the sources of the deadline flows.

This work builds upon and makes significant extensions to the Q-table based network scheduler presented in [11], which demonstrated the feasibility of using a Reinforcement Learning based approach to schedule deadline-driven flows. However, the Q-table based approach could not be applied to larger and/or heterogeneous networks. In this paper, we design and implement a network scheduler for deadline-driven flows based on Deep Deterministic Policy Gradient (DDPG). It uses an Actor-Critic model and is able to not only handle larger networks with heterogeneous links but also a generalization of the scheduling problem that assigns a utility value to each flow if the deadline is met. The scheduler task is to schedule the flows so as to maximize the aggregated utility value. We design a scheduler (RL-Agent) based on DDPG and compare its performance with three heuristics: (i) P_FAIR, which shares the bottleneck capacity in proportion to the access rates, (ii) V_D_Ratio, which prioritizes flows with high value-to-demand ratio, and (iii) V_B_EDF, which prioritizes flows with high value-to-deadline ratio. The main contributions of this paper are:

1) We propose a DDPG based network controller which a) scales with respect to the size of the network (number of sources and the bottleneck link capacity), b) can be applied to heterogeneous networks, and c) can be applied to a general value maximization problem.
2) For equally valued requests and homogeneous access rates, we show that the RL-Agent performs as well as and in some cases out-performs V_B_EDF (which is the same as the Earliest Deadline First (EDF) algorithm) and significantly better than P_FAIR (same as an idealized TCP algorithm).
3) For equally valued requests and heterogeneous access rates, we show that the RL-Agent performs as well as V_B_EDF even though it has no knowledge of the heterogeneity to start with.
4) Finally, for the value maximization problem, we show that the RL-Agent out-performs the heuristics for both homogeneous and heterogeneous access networks. This is despite the fact that the RL-Agent has no prior knowledge of the heterogeneity and the values, whereas the heuristics have full knowledge of the heterogeneity and V_D_Ratio and V_B_EDF have partial knowledge of the values.

II. System Model

In this study we consider a heterogeneous dumbbell network consisting of N sources and N destination nodes, as illustrated in Figure 1. The capacity of the link between router R1 and R2 is B Gbps. The access links between the sources S_i and R1 have rates $r_i \in (0, B)$. However, the rates of the last-hop links from R2 to the destination nodes D_i are greater than B and, hence, are never the bottleneck.

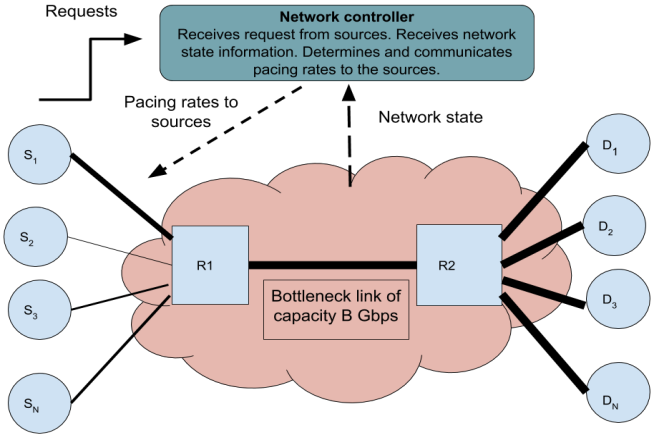

Fig. 1: A dumbbell network with heterogeneous access link rates. The thickness of the link represent different rates. Note that the link rates from Router R2 to destination nodes are same as the link between routers R1 and R2 and hence are never the bottleneck.

Requests for file transfers are made to the network controller. Each request j is a five-tuple $(s_j, d_j, f_j, d_j, v_j)$ where s_j denotes the source node, d_j the destination node, f_j is the filesize, d_j is the deadline, and v_j is a value awarded if the data transfer is completed within the deadline. Each request corresponds to a single TCP/IP flow. We assume only a single request from each source at any given time.

The basic unit of time is a scheduling interval; the network controller assigns a pacing rate to each active source at the start of each scheduling interval. We consider an episodic (batch) model in which the scheduler receives a request from each of the sources at the beginning of an episode [11]. Each request is to a different destination. The episode ends when all the requests have completed successfully (within the deadline) or not, after which a new episode begins. An episode consists of an integer number of scheduling intervals denoted as $1, \ldots, T$.

At the beginning of each scheduling interval, for each request j, we define $\text{Rmin}_j(t)$ as [11]:

$$\text{Rmin}_j(t) = \frac{\text{remaining file size}}{\text{time until deadline}} \tag{1}$$

The value $\text{Rmin}_j(t)$ denotes the minimum rate that is required at every subsequent scheduling interval for request j to meet its deadline. We let $\text{Rmin}_j(1) = \frac{f_j}{d_j}$, the initial minimum rate for request j, be denoted by Rmin_j.

We use $\text{Sum\_Rmin} = \sum_{j=1}^{n} \text{Rmin}_j$ to denote the sum of the initial Rmin of each request. $\text{Sum\_Rmin} < B$, $\text{Sum\_Rmin} = B$, and $\text{Sum\_Rmin} > B$ correspond to the **under-loaded**, **fully-loaded**, and **over-loaded** scenarios, respectively. Note that if $\text{Sum\_Rmin} \leq B$, it is should be possible to meet all the deadlines. However, as flows may end anytime during a

scheduling interval and pacing rate assignments are done only at the beginning of the scheduling intervals, there is some capacity waste which can result in lower than 100% success rate for the fully-loaded and the under-loaded cases. In the over-loaded case, the probability that not all the flows can be completed by their deadline increases as the difference Sum_Rmin $- B$ increases. For the subsequent discussion we introduce the term *demand of a request* j as $D_j = \lceil \frac{f_j}{B} \rceil$ [3]. This demand represents the number of intervals of the bottleneck link capacity that is required to transfer the file.

A. Justification of the Heterogeneous Dumbbell Network

The heterogeneous dumbbell network serves as a good abstraction of a large complex network such as the ESnet [2] due to the following reasons. First, many distributed instrument facilities are located at remote locations and it is the access links that connect these facilities to the core network that typically have low capacity. For example, the LSST cameras, which are located on top of a mountain in the Andes [5], have low capacity links connecting to the ESnet. Second, most of the ESnet links have very high capacity. There are very few links that are bottleneck links. For example, in ESnet the trans-continental links between the US and Europe is typically the only congested links. Finally, the destination nodes representing HPC systems or Data Transfer (DTN) nodes [12] are typically connected using very high speed networks and are not bottlenecks.

B. Heuristic Scheduling Algorithms

We consider the following heuristics for comparison:

- **Proportional Fair Share Allocation (P_FAIR):** The P_FAIR heuristic shares the bottleneck link capacity fairly among the active sources in proportion to their access link rates. If there are k active sources, then the pacing rate a_i to source i is:

$$a_i = \min\left(r_i, \frac{r_i}{\sum_{j=1}^{k} r_j} \times B\right) \tag{2}$$

  If all the access rates are equal, then this would correspond to an equal partition. The P_FAIR heuristic simulates an idealized version of the TCP protocol and does not take into account the value of each flow.
- **Value-Demand Ratio Based Allocation (V_D_Ratio):** The V_D_Ratio heuristic is an adaptation of the Greedy RTL algorithm outlined in [3]. In this algorithm, the parameters $p_i = \frac{v_i}{D_i}$ for each active flow i are sorted in non-increasing order. Capacity is allocated in turn starting with the flow with the largest p value. Let R be the remaining capacity (which is initially set to B). If with allocation of $\min(R, r_i)$ flow i cannot meet its deadline, then that flow is not scheduled. Otherwise, the flow is assigned $\min(R, r_i)$ and the algorithm considers the next flow in the sorted list with the updated R. Note that if all the values are equal and all the access link rates are equal, then this heuristic is the same as the Shortest Job First (SJF): the flow with the smallest demand will have the largest p value.
- **Value-Deadline Ratio Based Allocation (V_B_EDF):** The V_B_EDF heuristic is similar to the V_D_Ratio heuristic, however, the p values are calculated differently. Specifically, V_B_EDF heuristic calculates the parameter $p_i = \frac{v_i}{d_i}$ for each active flow i, and sorts this list in non-increasing order. The bottleneck capacity is allocated in a water filling approach. Starting with the flow with the highest p value, the corresponding flow is allocated up to its access rate limit (if it can meet the deadline with its access rate) and then moving down the priority list with the remaining capacity until all the flows have been considered or all the capacity is allocated. Note that if all the values are equal and all the access link rates are equal, then this heuristic will be the same as the well known Earliest Deadline First (EDF): the flow with the earliest deadline will have the largest p value.

C. Notes about the Heuristics

The V_D_Ratio heuristic is adapted from the Greedy RTL algorithm outlined in Jain et al. [13]. The problem considered by Jain et al. was to schedule deadline driven jobs with different maximum parallelism (analogous to the heterogeneous access link rates) in large computing cluster of a fixed processing capacity B so as to maximize the total value. The jobs are malleable implying that processing time scales with amount of capacity that is allocated (up to the maximum parallelism). The algorithm is greedy because it allocates high priority in allocating capacity to short jobs that have high values. It takes the deadline into account only indirectly by ignoring jobs that cannot meet the deadline. The key result is that the GreedyRTL algorithm is $\mathcal{O}(\frac{B}{B-k} \cdot \frac{s}{s-1})$ approximation of the optimal, where k is the maximum parallelism and s is a slackness factor that denotes the degree of flexibility the scheduler has in scheduling a job. Consequently, as long as s is large and $k \ll B$, this heuristic is near-optimal.

To the best of our knowledge, there is no prior research on V_B_EDF heuristic for the value maximization problem for deadline-driven jobs. This heuristic directly accounts for the deadline by prioritizing jobs that have high value and early deadline. As such it is the same as EDF when all the values are equal. EDF has been widely studied in scheduling deadline-sensitive jobs on multi-processor systems [14], [15] and for packet scheduling in multihop networks with hard deadlines [16]. For many of the above applications under certain constraints EDF achieves the same performance as the optimal offline algorithm [16]. For deadline driven data transfers in a heterogeneous network with multiple paths [8] and for the over-loaded cases [11], EDF is not optimal. Nevertheless, as shown in Section V, it out-performs the V_D_Ratio heuristic.

III. RL-Agent Based Scheduler

The overall architecture of the RL-agent is shown in Figure 2. In this section, we discuss the various components of

the architecture and how the scheduling task is represented for reinforcement learning.

A. Reinforcement Learning Formulation

In the following paragraphs we define the state and action spaces and the reward function.

a) State: The *state* $\boldsymbol{s}_t$ of the system at the beginning of the scheduling interval t, is defined as

$$\boldsymbol{s}_t = [\text{Rmin}_1(t), \text{Rmin}_2(t), \ldots, \text{Rmin}_N(t)], \tag{3}$$

where $\text{Rmin}_i(t)$ is defined in Eq. 1. When a flow i completes or its deadline has expired, the ith component in the state vector is set to 0. For each flow i, the state gives the minimum pacing rate that the agent needs to allocate at every scheduling interval until the deadline for the flow i to finish. This choice of the state gives the agent direct information about the urgency with which each flow must be allocated capacity in the bottleneck link.

b) Action: Action $\boldsymbol{a}_t$ for the interval t is defined as

$$\boldsymbol{a}_t = [a_1(t), a_2(t), \ldots, a_N(t)], \tag{4}$$

where a_i is the pacing rate allocated to the i-th flow. This definition of action is similar to the work in [11]. However, the key difference is that we consider $\boldsymbol{a}_t \in \mathbb{R}^N$, where N is the number of flows in the network. As such, this work considers the problem in a continuous action space. The usage of a continuous action space formulation here, as opposed to the discrete formulation in [11] holds several key advantages. In the Q-table approach, each state-action pair had to be individually enumerated and visited by the agent in order to generate an estimate of the value. With an increasing network size, the training of the agent could become infeasible. The continuous formulation allows for similar state-action pairs to be generalized over, and thus efficient learning over a larger state-action space [17].

An action is valid if it satisfies the following constraints:

$$a_i(t) \geq 0, \;\; i = 1 \ldots N \tag{5}$$

$$\sum_{i=1}^{N} a_i \leq B. \tag{6}$$

The second constraint implies that at every scheduling interval the scheduler uses all of the capacity of the bottleneck link. In order to generate such actions from our reinforcement learning agent, we apply the *softmax* function to the output layer of the actor network and scale the result to the bottleneck capacity B.

c) Reward: In this reinforcement learning problem, we ultimately wish to maximize the number of flows that the agent is able to complete before their deadlines. This suggests that a *flow completion reward* could be appropriate, where the agent receives a positive reward every time a flow is completed within its deadline and a zero reward otherwise. In this case, however, the reward will be nonzero infrequently, resulting in slow training. As a result, here we consider reward functions that give the agent information at every time step.

The reward function utilized in [11] measured the extent to which the *action* prioritized the completion of flows with the earliest deadlines. This prior reward function is defined as:

$$R(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{i=1}^{N} (d_{max} - d_i) * a_i, \tag{7}$$

where d_i denotes the time until the deadline for the ith flow, d_{max} is the maximum time until deadline for any flow and a_i is the ith component of the action vector, or the amount of bottleneck capacity allocated to the ith flow. Under this reward function, for example, any allocation to the flow with the latest deadline contributes zero to the reward because $d_i = d_{max}$ and so $(d_{max} - d_i) = 0$.

However, the reward function defined in Eq. 7 depends only on the action that is specified by the agent and not on variables from the state itself, which means that the reward function does not depend on the effect that the action had in the environment. Under heterogeneous flows, for example, not all actions specified by the agent may be possible to implement.

In this paper, we consider the following reward function:

$$R(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{i=1}^{N} (d_{max} - d_i) * \Delta f_i, \tag{8}$$

where d_{max} and d_i have the same significance as the previous reward function. Δf_i represents the difference between the file size remaining for flow i before the action is taken and the file size remaining for flow i after the action is taken. Thus, the Δf_i term measures how much progress the action taken made toward finishing flow i. If the action taken by the agent was fully implemented over the unit time interval, then $\Delta f_i = a_i$ so Eq. 7 is a special case of this reward function.

In the case that each flow has a value, v_i that represents how much preference is given to completing the flow, it is appropriate to scale the reward corresponding to a given flow by the value for that flow. This gives the most general case of the reward function used in this paper:

$$R(\boldsymbol{s}_t, \boldsymbol{a}_t) = \sum_{i=1}^{N} (d_{max} - d_i) * \Delta f_i * v_i \tag{9}$$

When flows are equally valued, the v_i term can be ignored; thus, reward functions in Eqs. 7 and 8 are special cases of this reward function.

B. The DDPG Architecture

The deep deterministic policy gradient algorithm utilizes an actor critic architecture [17]. We construct an actor network that takes as input the state vector described by Eq. 3 and outputs an action vector described by Eq. 4 that satisfies Eqs. 5 and 6. Additionally, we construct a critic network that takes in both the action and state vectors and returns an estimate of $Q(\mathbf{s_t}, \mathbf{a_t})$, the value of taking action $\mathbf{a_t}$ at state $\mathbf{s_t}$. The networks are trained using $(\mathbf{s_i}, \mathbf{a_i}, r_i, \mathbf{s_{i+1}})$ tuples sampled randomly from a store termed the *replay buffer*.

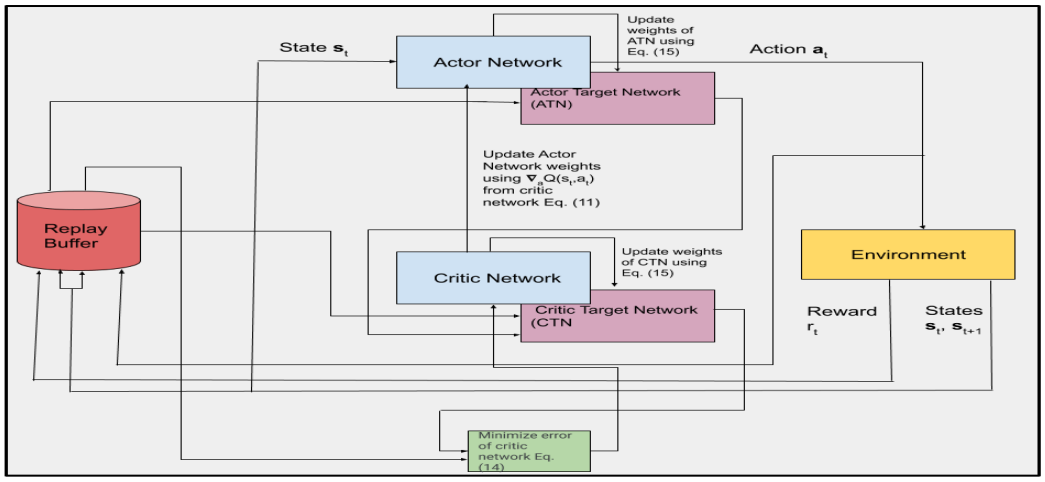

Fig. 2: The overall architecture of the RL-Agent. Equation numbers correspond to the equations in the text.

a) Actor Network: The actor network aims to learn a *policy function* $\pi_{\theta_{actor}} : S \rightarrow A$ that maximizes the *total expected discounted reward* given by:

$$\mathbb{E}[\sum_{i=t}^{T} \gamma^{i-t} * R(\boldsymbol{s}_t, \boldsymbol{a}_t)], \tag{10}$$

where S is the set of vectors described by Eq. 3 and A is the set of vectors described by Eq. 4, obeying restrictions in Eqs. 5 and 6. This expectation computes the total reward that is expected to be received by following a given policy, where rewards occurring later are discounted by a constant hyper parameter, γ, between 0 and 1. A gradient ascent process is utilized to learn the parameters θ_{actor} that maximize this expectation, by following the gradient given by

$$\mathbb{E}[\nabla_{\boldsymbol{a}_t} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) * \nabla_{\theta_{actor}} \pi_{\theta_{actor}}(\boldsymbol{s}_t)] \tag{11}$$

In this formula, the $\nabla_{\mathbf{a_t}} Q_{\theta_{critic}}(\mathbf{s_t}, \mathbf{a_t})$ is based off of a learned approximation of the $Q_{\theta_{critic}}(\mathbf{s_t}, \mathbf{a_t})$ function in the critic network. The parameters of the actor network are updated using the following equations:

$$\theta^{'}_{actor} = \theta_{actor} + \alpha * \mathbb{E}[\nabla_{\boldsymbol{a}_t} Q(\boldsymbol{s}_t, \boldsymbol{a}_t) * \nabla_{\theta_{actor}} \pi_{\theta_{actor}}(\boldsymbol{s}_t)], \tag{12}$$

where α represents a constant positive learning rate.

b) Critic Network: The critic network is trained to learn the function $Q(\mathbf{s_t}, \mathbf{a_t})$, which is the expected reward from taking action $\mathbf{a_t}$ during the state $\mathbf{s_t}$ and then following the policy $\pi_{\theta_{actor}}$. In the context of the reinforcement learning formulation presented here, the critic network aims to learn the extent to which an action taken at a given state would contribute to the *earliest deadline flow* being completed first. The critic network was trained to approximate a target $Q(\mathbf{s_t}, \mathbf{a_t})$ value given by the Recursive Bellman Equation, as follows:

$$Q^{Bellman}(\boldsymbol{s}_t, \boldsymbol{a}_t) = R(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma * Q_{\theta_{critic}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \tag{13}$$

where $Q_{\theta_{critic}}$ indicates the value that is taken from the critic network parameterized by parameters θ_{critic}. These parameters, θ_{critic} are updated to minimize the squared error between $Q_{\theta_{critic}}(\mathbf{s_t}, \mathbf{a_t})$ and $Q^{Bellman}(\mathbf{s_t}, \mathbf{a_t})$. Thus, the parameters are updated as follows:

$$\theta^{Updated}_{critic} = \theta_{critic} - \nabla_{\theta_{Critic}} \alpha * \mathbb{E}[(Q_{\theta_{critic}}(\boldsymbol{s}_t, \boldsymbol{a}_t) - Q^{Bellman}(\boldsymbol{s_t}, \boldsymbol{a_t}))^2] \tag{14}$$

c) Target Networks: As seen in the update equations for the actor and critic networks, the networks are updated directly and indirectly based off of their current values. For example, the update to the critic network depends on the value of the critic network itself. This has been identified as being susceptible to divergence and instability [17]. One innovation is to create separate target networks for both the actor and critic networks that are used to compute values for the network updates. The parameters of the target networks slowly trail those of the actual networks through the following update:

$$\theta^T = \tau * \theta + (1 - \tau) * \theta^T \tag{15}$$

where τ is a positive constant between 0 and 1. By utilizing this separate network for calculating the updates for the network parameters, we ensure the stability of the network updates and avoid divergence [17].

d) Replay Buffer: When training the critic network on samples directly taken from the agent's trajectory, there exist temporal correlations between the training samples that could induce a higher variance in the critic network's approximation of the true $Q(\mathbf{s_t}, \mathbf{a_t})$ function. This can lead to divergence of the critic network. An innovation for addressing this issue is the usage of a *replay buffer* in which $(\mathbf{s_i}, \mathbf{a_i}, r_i, \mathbf{s_{i+1}})$ tuples are stored [17]. At each training step, a mini-batch of these tuples are sampled with uniform probability from the replay buffer and utilized for the update. This breaks the temporal correlations between the samples and upholds the independent and identically distributed (i.i.d) assumption expected by the stochastic gradient ascent algorithms utilized [18].

e) Exploration Noise: An important tool employed in reinforcement learning applications is exploration noise, where the action generated by the reinforcement learning agent is

perturbed by random noise prior to being implemented. Here, the effect of exploration noise is to vary the pacing rates that are assigned to each flow in a random fashion. Exploration noise ensures that the reinforcement learning agent is able to explore different regions within the state-action space, preventing the agent from being trapped in a local optima. Like other DDPG applications, we make use of an additive Ornstein-Uhlenbeck temporally-correlated noise process [17]. Following the addition of exploration noise, the action was re-scaled to meet the restrictions in Eqs. 5 and 6.

IV. Methodology

A. Tool

A prototype RL-Agent was implemented using the TensorFlow machine learning library [19]. We generated actor and critic networks consisting of an input layer, followed by a fully-connected hidden layer with 400 units, followed by a fully connected hidden layer of 300 units. A network simulator was developed using the NumPy numerical library which, given a value of the desired Sum_Rmin of all of the requests, generated request file sizes and deadlines and simulated an episode by generating the state at each scheduling interval, taking and implementing the action, and returning the reward signal to the agent.

In this work, the RL-Agent was trained on a single network configuration. The RL-Agent was trained in an iterative manner with the environment initialized with randomly generated requests of a given Sum_Rmin workload level. The agent would step through the scheduling intervals in the environment until every flow was completed or expired and the episode terminated. Afterwards, a new episode was generated and training continued. In this work, we allowed training to continue to up to 25,000 training episodes.

B. Metrics

We have used the following metrics in this study

1) **Evaluation Run:** Periodically during the RL-agent training, an evaluation run of 100 episodes was conducted. During these 100 episodes, no exploration noise was added to the actions and no network updates were performed.
2) **Success Rate:** This is defined as the fraction of the number of requests that meet the deadline. For the RL-Agent, this was reported as an average over the 100 episode evaluation run.
3) **Total Episode Value:** This is defined as the sum of the values of all completed flows. For the RL-Agent, this is averaged over the 100 episode evaluation run.

C. Parameters

Table I gives a summary of all the hyper-parameters

In the following section unless otherwise stated the filesize for each request is sampled from Uniform distribution of range 2 to 50 Gbs (Gigabits).

TABLE I: Hyper-parameters and their corresponding values.

Parameter	Description	Typical values
$size_{rb}$	Replay Buffer Size	10^6
$size_{mb}$	Mini-batch Size	64
τ	Target Network Update Weight	0.001
γ	Bellman Equation Discount Factor	0.99
α	Learning Rate	0.001

V. Results and Discussion

A. Homogeneous Link Rates with Equally Valued Flows

We first considered the case of five sources with homogeneous access link rates of 20 Gbps and the bottleneck link of 20 Gbps capacity. Workloads with different Sum_Rmin values corresponding to under-loaded, fully-loaded, and over-loaded cases were generated. The learning curves for three cases 1) Sum_Rmin = 15 Gbps, 2) Sum_Rmin = 20 Gbps, and 3) Sum_Rmin = 25 Gbps are shown in Figure 3. The results show that the RL-Agent's performance stabilizes quickly and as expected to different values. The Success Rate performance

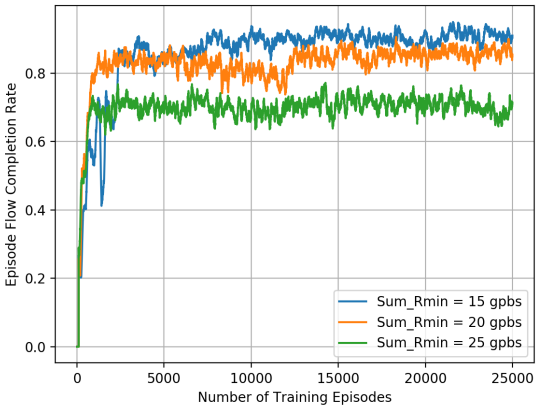

Fig. 3: Learning curve (Success Ratio as a function of time) for homogeneous equally valued flows. Parameters: $B = 20$, $N = 5$, Homogeneous access rate of 20 Gbps.

of the RL-Agent compared to the three heuristics are shown in Figure 4. The RL-Agent achieves superior performance to P_FAIR and V_D_Ratio heuristics. Moreover, it is able to achieve comparable performance to the V_B_EDF heuristic for under-loaded and fully-loaded cases and out-perform for over-loaded cases. These results replicate the work done in [11], utilizing a DDPG architecture as opposed to a Q-table. Note that the inferior performance of V_D_Ratio in this and the subsequent cases is because the requirement $k(= \max\{r_1, \ldots, r_N\}) \ll C$ is not met.

One key advantage of the continuous action formulation of the DDPG model is that the RL-Agent scales well to networks with more sources and larger bottleneck capacities. We considered a homogeneous network with 7 sources and a bottleneck capacity of 40 Gbps. This configuration was infeasible with the the simple Q-table approach in [11]. As

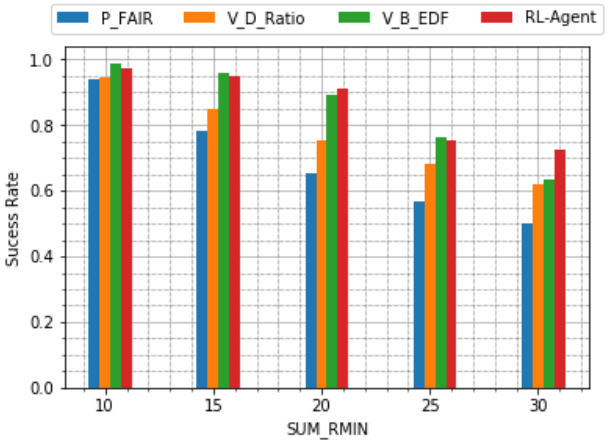

Fig. 4: Success Rate as a function of Sum_Rmin for the RL-Agent and the three heuristics. Parameters: $B = 20$ Gbps and $N = 5$, Homogeneous access rate of 20 Gbps.

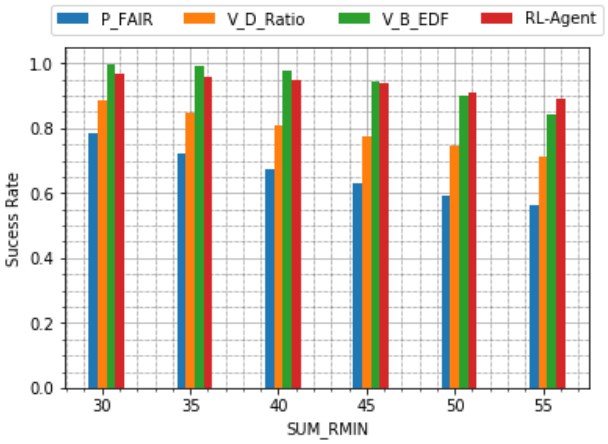

Fig. 5: Success Rate as a function of Sum_Rmin for the RL-Agent and the three heuristics. Parameters: $B = 40$ Gbps and $N = 7$, Homogeneous access rate of 40 Gbps, Filesize $f_i \sim \text{Unif}(10, 100)$ Gb.

before we considered a range of Sum_Rmin values corresponding to the under-loaded, fully-loaded and overloaded cases. The results shown in Figure 5 demonstrate that the scheduler is able to achieve a Success Rate similar to the 5 source case. The RL-Agent achieves performance close to the V_B_EDF heuristics, for the under-loaded and fully-loaded cases and out-performs it in the over-loaded cases (Sum_Rmin > 40). It is notable that only minor changes had to be made to allow the agent to accommodate the network with 7 sources. Specifically, the input and output layers of the (neural) network were modified to match the number of sources and the exploration noise variance was scaled proportionally to the bottleneck capacity. Hence the RL-Agent can be applied in even larger networks.

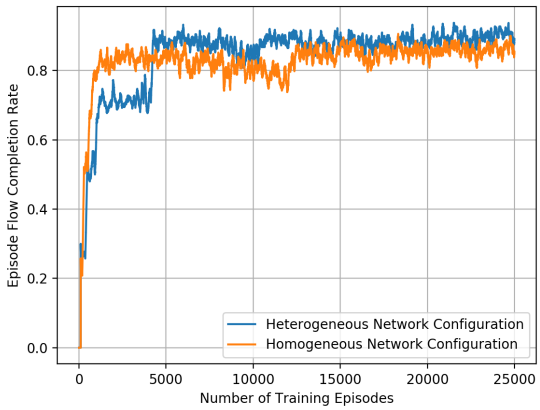

Fig. 6: Learning Curve (Success Rate as a function of training episode) for the RL-Agent for Heterogeneous network (Blue) and for Homogeneous network (Orange) for Sum_Rmin $= 20$. The Success Rate of the heterogeneous case is calculated by considering that in each episode at most 4 requests can be completed. Parameters $B = 20$, $N = 5$.

B. Heterogeneous Link Rates with Equally Valued Flows

In the case of networks with heterogeneous access rates, the pacing rate assigned to a source may be larger than its access rate. As a result, the heterogeneity must be taken into account to avoid wasting the bottleneck capacity. To investigate this, we trained RL-Agent for a heterogeneous network with 5 sources with access rates (20, 20, 20, 20, 0). This is an extreme case as requests from the source with 0 access rate can never be completed. This is particularly disadvantageous to the RL-Agent since it does not have direct information about the heterogeneity. This case would give a sharp contrast of the performance of the RL-Agent in comparison to the heuristics if it did not adapt to the heterogeneity.

We considered both the reward function in Eq. 7 [11] as well as the new generalized reward function given in Eq. 8. As discussed before, this new reward function implicitly gives the agent information about the heterogeneity. We examined the correlation between the total reward that an agent earned over an evaluation episode and the number of flows that it completed. The total episode reward when utilizing the reward function in Eq. 7 had a weak correlation (correlation coefficient $r = 0.362$) with the number of flows that the agent successfully completed. Utilizing the new reward function in Eq. 8, on the other hand, resulted in a strong positive correlation ($r = 0.857$) between episode reward and number of completed flows. The low correlation value indicates that the reward function in Eq. 7 is unsuitable for use in maximizing the number of flows completed. This finding motivated the modification of the reward function to that shown in Eq. 8. In Figure 6, the learning curve of the RL-Agent trained with reward function in Eq. 8 is shown in comparison to the learning curve for the homogeneous network for the fully-

loaded case. The Episode Flow Completion Rate for the heterogeneous case is corrected to account for the fact that it could at most complete 4 flows. The similarity of the learning trends establishes that the reward function in Eq. 8 is appropriate for training the RL-Agent for a heterogeneous network.

We continued to consider a network with 5 sources with a bottleneck capacity of 20 Gbps and access rates of (20,20,20,20,0) and compared the performance of the RL-Agent to the heuristics for different Sum_Rmin values. Results, displayed in Figure 7, indicate that the RL-Agent attains superior performance to the P_FAIR and V_D_Ratio heuristics and comparable performance to V_B_EDF. When comparing the results, it is important to note that while the heuristics had full knowledge of the heterogeneity, the RL-Agent could only learn this through the reward signal.

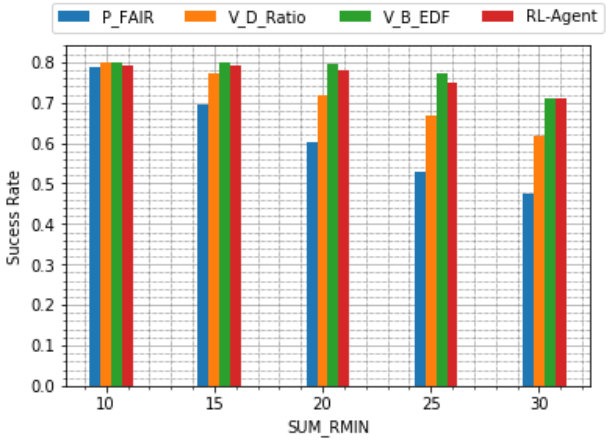

Fig. 7: Success ratio as a function of Sum_Rmin for the RL-Agent and three heuristics. Parameters: $B = 20$ Gbps and $N = 5$, Access rates = $[20, 20, 20, 20, 0]$.

C. Flows with Different Values

In the case of differently valued flows, the *total episode value* metric is used. We considered the same network of 5 sources with homogeneous access link rates of 20 Gbps and a 20 Gbps bottleneck link. Sources were each assigned a unique integer value between 1 and 5. During the training process, the value associated with each source remained constant. The performance of the RL-Agent in comparison to the heuristics is displayed in Figure 8. The RL-agent surpasses P_FAIR, V_D_Ratio, and the V_B_EDF although RL-agent had to learn about the different values through the reward.

We examined the value maximization problem utilizing the heterogeneous network with access rates [20,20,20,20,0]. This heterogeneity was chosen as an extreme case, as failure of the agent to learn about the heterogeneity would result in a definitive drop in performance. We considered two value assignments, one in which the source with 0 access rate had the highest value (5) and the other in which it had the lowest value (1). The average episode value for the RL-Agent as

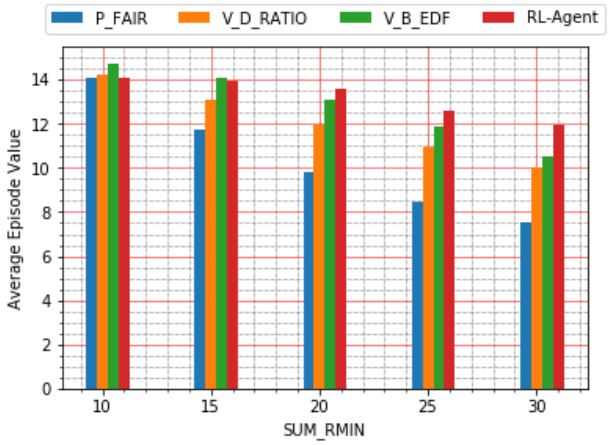

Fig. 8: The average Episode Value as a function of Sum_Rmin for the RL-Agent and the three heuristics. Parameters: $B = 20$ Gbps and $N = 5$, Homogeneous access rates of 20 Gbps.

TABLE II: Average Episode Value for heterogeneous access links rate given in R and different value assignments given in V. Parameters: $B = 20$, $N = 5$, Sum_Rmin $= 20$.

Parameter	P_FAIR	V_D_Ratio	V_B_EDF	RL-Agent
R=(20,20,20,20,0) V=(5,4,3,2,1)	10.14	11.53	12.62	12.93
R=(20,20,20,20,0) V=(1,2,3,4,5)	7.24	8.23	8.96	9.14

well as the heuristics is presented in Table II. The RL-Agent is able to out-perform all the heuristics. This demonstrates the significant robustness of the RL-Agent as it had no prior knowledge of both the heterogeneity and the values, whereas the heuristics had full knowledge of the heterogeneity. V_D_Ratio and V_B_EDF had partial knowledge of the value (through the ratios of value to demand and value to deadline, respectively) while P_FAIR did not take into account value.

VI. Related Work

Deadline-aware flow scheduling has been investigated for inter-datacenter traffic [7], [8], [20], for science workflows [21], and for data center networks [22]. The study in Tempus [8] considered a mix packing and covering problem to develop an online traffic engineering framework to efficiently schedule and route long flows to maximize the fraction of transfer delivered before deadline. The study in [7] formulated an optimization problem and proposed a framework called Ameoba for flexible bandwidth allocation to meet deadlines. The study in [21] considered a multihop network with deadline-driven data transfer requests and compared two classes heuristics - one that optimized globally and one that optimized locally at each link - and compared the success rate in meeting deadlines and network utilization. The study illustrated the complexity of the scheduling problem for a general network.

Deadline-based packet transfers over multi-hop networks have been studied [16]. Finding optimal policies for deadline-

driven transfers is a complex problem with multi-hop networks where contention can occur at multiple links [16]. Scheduling deadline-driven malleable job on a compute cluster has been investigated [3]. As mentioned in Section II, this problem is similar to the deadline-driven data transfers problem considered in this paper. However, the proposed heuristic is for a special case and hence do not perform very well when applied to the problem considered in this paper.

There has been significant recent work in RL-Agent based approaches in resource management problems such as resource allocation in wireless networks and scheduling jobs in computing systems. The study in [23] demonstrated that an RL-Agent can be trained to optimize scheduling jobs with multiple resource (processor and memory) demands. The study demonstrated that the RL-Agent can perform as well as and sometimes out-perform known optimal heuristics.

This paper builds on [11] and significantly extends it in multiple dimensions. First, using a DDPG (as opposed to a simple Q-table in [11]) addressed a critical problem of scalability. The framework now allows investigating more general network with heterogeneous links with a large number of sources and higher bottleneck link capacity as has been demonstrated in this paper. Second, the generalization of the reward function has allowed formulating the more general value maximization problem. Finally, the results have been compared with both new and known near-optimal heuristics for a heterogeneous access network.

VII. Acknowledgement

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

VIII. Conclusions & Future Work

This paper describes a DDPG based reinforcement learning agent that is capable of scheduling deadline driven flows and achieves performance that matches or outperforms well-known and near-optimal heuristics. There are multiple avenues for further work with regards to this specific scheduling problem. One possible area of investigation is identifying exploration noise processes which allow for the state-action space to be efficiently covered, as well as the optimal settings for the parameters of these processes. Although the Ornstein Uhlenbeck noise used in this paper has been shown as effective in physical control tasks [17] it remains unclear whether it is best suited to scheduling tasks, such as this one. Further investigation can also be conducted in determining the sensitivity of the agents performance to a wider range of hyper parameter settings. These more general investigations could potentially help in the use of DDPG in other scheduling tasks, particularly an extension of the problem to more complex network settings that include routing. This would require scalable representations for the states and actions and a further investigation into the flow completion reward.

References

[1] Kathy Yelick. A superfacility model for science. https://people.eecs.berkeley.edu/~yelick/talks/data/Superfacility-TechX17.pdf, 2017.

[2] ESnet. Energy sciences network (esnet). http://www.es.net/.

[3] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. 43(4):3–14, 2013.

[4] Ewa Deelman, Tom Peterka, Ilkay Altintas, Christopher D Carothers, Kerstin Kleese van Dam, Kenneth Moreland, Manish Parashar, Lavanya Ramakrishnan, Michela Taufer, and Jeffrey Vetter. The future of scientific workflows. *The International Journal of High Performance Computing Applications*, 32(1):159–175, 2018.

[5] Željko Ivezić, Steven M Kahn, J Anthony Tyson, Bob Abel, Emily Acosta, Robyn Allsman, David Alonso, Yusra AlSayyad, Scott F Anderson, John Andrew, et al. Lsst: from science drivers to reference design and anticipated data products. *The Astrophysical Journal*, 873(2):111, 2019.

[6] Christoph Bostedt, Sébastien Boutet, David M Fritz, Zhirong Huang, Hae Ja Lee, Henrik T Lemke, Aymeric Robert, William F Schlotter, Joshua J Turner, and Garth J Williams. Linac coherent light source: The first five years. *Reviews of Modern Physics*, 88(1):015007, 2016.

[7] Hong Zhang, Kai Chen, Wei Bai, Dongsu Han, Chen Tian, Hao Wang, Haibing Guan, and Ming Zhang. Guaranteeing deadlines for inter-data center transfers. *IEEE/ACM Transactions on Networking (TON)*, 25(1):579–595, 2017.

[8] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. Calendaring for wide area networks. 44(4):515–526, 2014.

[9] Chin Guok, David Robertson, Mary Thompson, Jason Lee, Brian Tierney, and William Johnston. Intra and interdomain circuit provisioning using the oscars reservation system. In *2006 3rd International Conference on Broadband Communications, Networks and Systems*, pages 1–8. IEEE, 2006.

[10] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, 2015.

[11] Dipak Ghosal, Sambit Shukla, Alex Sim, Aditya Thakur, and Kesheng Wu. A reinforcement learning based network scheduler for deadline-driven data transfers. In *IEEE Global Communications Conference*, 2019.

[12] Eli Dart, Lauren Rotman, Brian Tierney, Mary Hester, and Jason Zurawski. The science dmz: A network design pattern for data-intensive science. *Scientific Programming*, 22(2):173–185, 2014.

[13] Navendu Jain, Ishai Menache, Joseph Seffi Naor, and Jonathan Yaniv. Near-optimal scheduling mechanisms for deadline-sensitive jobs in large computing clusters. *ACM Transactions on Parallel Computing*, 2(1):3, 2015.

[14] David Karger, Cliff Stein, and Joel Wein. Scheduling algorithms. *CRC Handbook of Computer Science*, 1997.

[15] Xiaohu Wu and Patrick Loiseau. Algorithms for scheduling deadline-sensitive malleable tasks. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 530–537. IEEE, 2015.

[16] Zhoujia Mao, Can Emre Koksal, and Ness B Shroff. Optimal online scheduling with arbitrary hard deadlines in multihop communication networks. *IEEE/ACM Transactions on Networking (TON)*, 24(1):177–189, 2016.

[17] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[18] Ruishan Liu and James Zou. The effects of memory replay in reinforcement learning, 2017.

[19] Martín Abadi et. al. Dean, Tucker, Yu, and TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[20] Long Luo, Hongfang Yu, Zilong Ye, and Xiaojiang Du. Online deadline-aware bulk transfer over inter-datacenter wans. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 630–638. IEEE, 2018.

[21] Fatma Alali, Nathan Hanford, Eric Pouyoul, Raj Kettimuthu, Mariam Kiran, Ben Mack-Crane, Brian Tierney, Yatish Kumar, and Dipak

Ghosal. Calibers: A bandwidth calendaring paradigm for science workflows. *Future Generation Computer Systems*, 89:736–745, 2018.

[22] Mohammad Noormohammadpour, Cauligi S Raghavendra, Sriram Rao, and Asad M Madni. Rcd: Rapid close to deadline scheduling for datacenter networks. In *2016 World Automation Congress (WAC)*, pages 1–6. IEEE, 2016.

[23] Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, pages 50–56. ACM, 2016.