A Reinforcement Learning Based Network Scheduler For Deadline-Driven Data Transfers

Dipak Ghosal[1,2], Sambit Shukla[1], Alex Sim[2], Aditya V. Thakur[1], and Kesheng Wu[2]

[1]University of California, Davis, CA, U.S.A.
{dghosal, sshukla, avthakur}@ucdavis.edu
[2]Lawrence Berkeley National Laboratory, Berkeley, CA, U.S.A.
{dghosal, asim, kwu}@lbl.gov

***Abstract*—We consider a science network that runs applications requiring data transfers to be completed within a given deadline. The underlying network is a software defined network (SDN) that supports fine grain real-time network telemetry. Deadline-aware data transfer requests are made to a centralized network controller that schedules the flows by setting pacing rates of the deadline flows. The goal of the scheduling algorithm is to maximize the number of flows that meet the deadline while maximizing the network utilization. In this paper, we develop a Reinforcement Learning (RL) agent based network controller and compare its performance with well-known heuristics. For a network consisting of a single bottleneck link, we show that the RL-agent based network controller performs as well as Earliest Deadline First (EDF) in the underloaded case and better in the overloaded case. We also show that the RL-agent performs significantly better than an idealized TCP protocol in which the bottleneck link capacity is equally shared among the competing flows. We also study the sensitivity of the RL-agent controller for some of the parameter settings.**

***Index Terms*—Science workflows, Deadline-driven data transfers, Software-defined Networking (SDN), Reinforcement Learning, Scheduling Heuristics, Earliest Deadline First (EDF), TCP**

I. Introduction

Guaranteed delivery of large data sets within a pre-specified deadline over wide-area networks is an important requirement for many scientific workflows. For example, the Zwicky Transient Facility [21], which is capable of finding transients and variable stars an order of magnitude faster than the previous generation of synoptic surveys, generates approximately 1.3 GB of uncompressed data every 45 seconds. This data is processed through multiple pipelines at different networked [1] high-performance computing (HPC) nodes to generate alerts. These alerts must be generated within a deadline so that additional observations can be scheduled during the same observation night. Consequently, data transfer between nodes and the processing pipelines at the HPC nodes must meet their deadlines. Deadlines for data transfers can be achieved by setting up connections between nodes with guaranteed bandwidth [3]. However, such an approach can lead to poor utilization of network capacity, especially when data generation is bursty. Consequently, an additional goal is to ensure that the network utilization is maximized.

This work was supported by NSF Grant CNS-1528087 and the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

In this study, we consider a network that is equipped with SDN switches/routers [1], and a central controller that can change packet forwarding rules at the switches/routers based on the network state. Thus, rate allocation and priority for individual flows can be altered. We also assume that sources can pace traffic and change the pacing rate quickly based on commands from the central controller [19]. Furthermore, we assume that changes in the pacing rate can be achieved without incurring any Transmission Control Protocol (TCP) penalty such as packet loss and poor throughput performance. There have been a number of studies that address deadline-aware scheduling of data transfers over SDN-enabled wide area networks. These include B4 (Google) [11], SWAN and TEMPUS (Microsoft) [14], DNA/AMOEBA (AWS) [25], among others. These studies target large inter-datacenter transfers in the presence of background transfers. A common approach in these studies is to formulate an optimization problem which in most cases is computationally expensive. The goal then is to develop heuristics that achieve good performance.

The general problem is to design a network controller that, given the state of the network, determines the pacing rate of the sources of the deadline-aware flows, the routes of the deadline-aware flows, and the rate allocations of the non-deadline flows such that the deadlines are met and the network utilization is maximized. Similar problems, but without routing, have been extensively studied in scheduling malleable jobs with deadlines on parallel processing systems [5], [9]. Another somewhat similar problem is in the domain of transportation networks where the goal is to jointly optimize traffic signal control and routing to maximize throughput and minimize latency [6], [18]. The overall problem is a complex optimization problem. Machine Learning (ML) in general, and Reinforcement Learning (RL) in particular, have been applied to similar resource management problems [10], [16]. However, to the best of our knowledge there is no prior work related to the application of RL to this networking problem.

Rather than naively applying a RL tool to the general problem as stated above, we consider a simpler problem to establish the advantages of an RL-based approach by comparing with known heuristics. We consider a network

with one bottleneck link on which only deadline-driven data transfer requests are made. We consider three heuristics for sharing the bottleneck bandwidth among the competing flows: (i) Earliest Deadline First (EDF), (ii) Equal Partition, and (iii) Random. We implement an RL-agent controller and compare its performance with that of these three heuristics. As EDF is known to be optimal for the case of the single bottleneck link [13], [17], [24] and Equal Partition is an idealized version of TCP, a comparative evaluation provides a good benchmark for the design of the RL-agent controller.

The following are the main contributions of this study:

1) We present a Reinforcement Learning (RL) based approach for scheduling deadline-aware network flows.
2) We show that even using a simple reward function, the RL-based approach achieves performance comparable to the (optimal) Earliest Deadline First (EDF) heuristic.
3) We also show that our proposed RL approach performs better than an idealized TCP algorithm, which equally shares the bottleneck bandwidth.

II. Motivation

Scientific workflows are complex, often generating large amounts of data that is processed in multiple stages by high-performance computing (HPC) systems which are shared among different applications. Data generated at remote locations is transferred between the source and HPC systems using high-speed networks, that carry other background traffic. Increasingly many of these scientific workflows require processing to be completed within a deadline, which, in turn, imposes deadline for the network data transfer [7]. A recent example for deadline-aware data transfer occurred when the LIGO [2] and Virgo [8] detectors observed a gravitational wave signal associated with the merger of two neutron stars. The merger, known as a kilonova, occurred in a galaxy 130 million light-years from Earth in the southern constellation of Hydra. The data from this initial observation had to be processed in a timely manner and sent to astronomers around the world so that they could aim their instruments to the right section of the sky to image the source of the signal [4]. Such deadline transfers are also common in cloud services [14], [25] where large inter-data center transfers must be made within a deadline in the presence of interactive and delay-tolerant background traffic.

We consider a science network that runs application requiring deadline transfers [1], as shown in Figure 1. Deadline-aware data transfer requests are made to a central network controller that schedules the flows by setting pacing rates of the deadline flows. The scheduler operates in a time slotted manner where at the beginning of each time slot it determines what will be the pacing rates of the flows for the next scheduling interval. These are determined by a scheduling algorithm which could be based on some heuristic or a machine learning approach with the goal of maximizing the number of flows that meet the deadline while maximizing the network utilization. We assume that the underlying network is a software defined network (SDN) that supports fine grain real-time network telemetry, which provides information required for the scheduling algorithm. We will assume that the source can precisely pace traffic at a rate directed by the network controller.

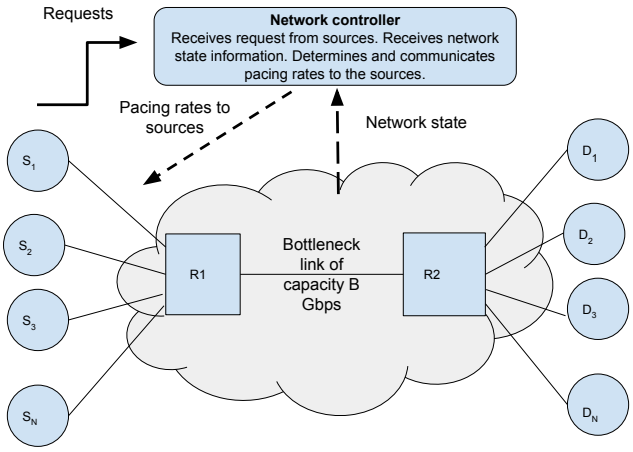

Fig. 1. A simple dumbbell network. The access links from the sources $S_1 \ldots S_N$ to router $R1$ and the links from router $R2$ to the destinations $D_1 \ldots D_N$ have large capacities. Hence, the link between $R1$ and $R2$ is the bottleneck link.

III. The Model

This study is based on a dumbbell network shown in Figure 1. We consider N sources that share one bottleneck link to N destinations. The rate of the bottleneck link is B Gbps. The links from the sources $S_1 \ldots S_N$ to the router $R1$ and from router $R2$ to the destinations $D_1 \ldots D_N$ have very high capacities and hence are never the bottleneck.

We consider an episodic (batch) model in which the scheduler receives a request from each of the sources at the beginning of each episode. The episode ends when all the requests have completed successfully (within the deadline) or not, upon which a new episode begins. 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 that is attributed if the data transfer is completed within the deadline. corresponds to a single TCP/IP packet flow, identified by a flow-id. We assume only a single flow from each source at any given time.

We consider a time-slotted system as shown in Figure 2. The basic slot is a flow update interval T_{fu}. A fixed number of flow-update intervals make up a scheduling interval T_{sc}. An episode consists of a variable number of scheduling intervals.

1) **Flow-update Interval:** This is the basic time unit. The purpose is to model progress of the flow as a consequence of interference from other deadline and background flows. If all traffic in the network are perfectly paced deadline flows, as is the case in this study, then the progress of a flow at the flow update interval is simply determined by the pacing rate that is allocated to the flows.

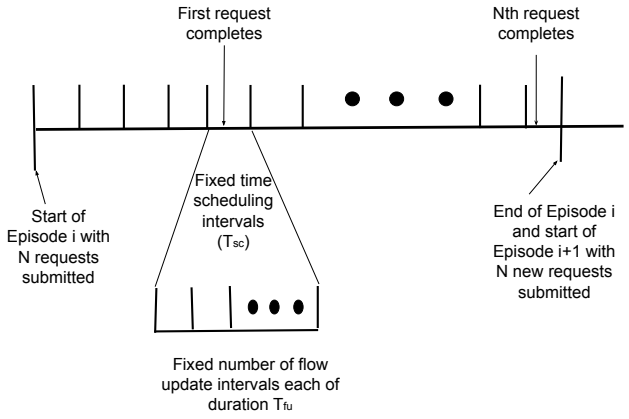

Fig. 2. The basic unit of timing is flow update interval T_{fu}. The scheduling interval T_{sc} denotes time between when scheduling decisions are taken.

2) **Scheduling Interval:** This is when scheduling decisions are taken and pacing rates are assigned to the flows.
3) **Episode:** This consists of a number of scheduling intervals that are required for all the flows to complete, whether meeting the deadline or not.

At the beginning of each scheduling interval, the network controller assigns a pacing rate to each source which it maintains for the duration of the interval. In this study, we assume that these are only integer rates. Furthermore, we assume that a flow could be paused for a given scheduling interval by assigning pacing rate of 0.

A. Heuristic Scheduling Algorithms

We consider the following heuristics for comparison:

- **Equal Partition (TCP):** In this case the bottleneck link capacity is "equally divided" among the competing flows. Since we consider only integer pacing rates, for 3 active flows, 10 Gbps bandwidth is randomly divided into one of $\{(4,3,3),(3,4,3),(3,3,4)\}$. This is an idealized version of the TCP protocol in which flows with equal round-trip times can be shown to equally share the bottleneck link capacity [15].
- **Random (RANDOM):** In this scheme, the bottleneck capacity is randomly partitioned among the active flows. This is a baseline case for comparison only.
- **Earliest Deadline First (EDF):** This scheme allocates all the capacity to the flow that has the earliest deadline. EDF has been widely studied in scheduling deadline-sensitive jobs on multi-processor systems. For a single machine system, EDF has been proved be the optimal policy [13], [24]. EDF has also been studied for packet scheduling in multihop networks with hard deadlines [17]. It was known that for single hop system, EDF has the same performance as the optimal offline algorithm when the system is underloaded. For tree-based multihop networks, EDF algorithm achieves the same performance as the optimal offline algorithm [17]. Since we consider a network with a single bottleneck link, we use EDF as the benchmark for the best achievable performance.

IV. Reinforcement Learning (RL) Agent based Network Controller

In reinforcement learning [22], the system is modeled as a Markov Decision Process (MDP) consisting of a set of states (S) and actions (A). A *state* is a snapshot of some characteristic variables that define the system. An *action* enables transition between system states. A learning agent learns by associating appropriate rewards with every state-action pair (S_i, A_j). With an appropriate reward function, the agent goes through a learning phase where it learns by associating rewards to the different actions at a given state. Below we describe these components for RL-Agent based network controller.

A. States

For each request j, we define $Rmin_j(t)$ as:

$$Rmin_j(t) = \frac{\text{remaining number of bytes}}{\text{time until deadline}} \tag{1}$$

The value $Rmin_j(t)$ denotes the minimum rate that is required at every subsequent scheduling interval for request j to meet the deadline. We let $Rmin_j(0) = \frac{f_j}{d_j}$, the initial minimum rate for request j, be denoted by $Rmin_j$. In this paper, the state of the system $S = \{n, Rmin_1(t), \ldots, Rmin_n(t), u\}$, where n is the number of active flows, $Rmin_i(t)$ is as defined above and u is the utilization of the bottleneck link. To restrict the size of the state space we consider only integer values of $Rmin_i(t), i = 1, \ldots, n$. In the current study, for all scheduling policies, the bottleneck capacity B is completely allocated and consequently the utilization u is always 1.

B. Actions

The action consist of setting the pacing rates of the sources. Thus, the action $A = \{a_1, \ldots, a_n\}$, where a_i is the pacing rate assigned to ith active flow. We assume integer rates, not exceeding bottleneck capacity B.

C. Reward Function

A good reward function is crucial for designing an efficient RL-agent. In our design, the reward function consists of two parts: (1) a scheduling interval reward, and (2) a flow completion reward. The scheduling interval reward is assigned at the end of each scheduling interval, while the flow completion reward is assigned when a flow completes within the deadline.

1) Scheduling Interval Reward: For each scheduling interval, a reward is assigned to the state-action pair. In this study, we have considered the following reward functions:

(i) Scheduling Interval Reward Function 1 **(SIRF1)**: Let (S_j, A_j) be state action pair for the scheduling interval j and let n be the number of active flows. In this reward function, we attempt to mimic the EDF policy. Let $drem_i$ denote remaining time until deadline for active flow i. Let $drem_{max}$ denote max of the remaining times over all the active flows. Let a_i denote the pacing rate

of the ith flow, then the scheduling interval reward SIR for state action pair (S_j, A_j), $SIR(S_j, A_j)$ is given by

$$SIR(S_j, A_j) = \sum_{i=1}^{n} (drem_{max} - drem_i) \times a_i \quad (2)$$

With this reward function, assigning all the capacity of the bottleneck link to the flow with earliest deadline will result in maximum reward.

(ii) Scheduling Interval Reward Function 2 **(SIRF2)**: The reward for flow i, denoted by reward_i, is given by

$$\text{reward}_i = \begin{cases} 1 & a_i(t) \geq Rmin_i(t) \\ -1 & a_i(t) < Rmin_i(t) \end{cases} \quad (3)$$

where a_i is the pacing rate assigned to flow i. For a state S_j and action A_j, the scheduling interval reward for the jth interval $SIR(S_j, A_j) = \sum_{i=1}^{n} \text{reward}_i$, where n is the number of active flows. In this reward function, an action that assigns the required $Rmin$ to more flows receives a higher overall reward.

2) Flow Completion Reward: The flow completion reward is assigned when a flow completes within the deadline. If (S_i, A_i) is the state action pair that resulted in the flow to be completed within the deadline and $\{(S_0, A_0), (S_1, A_1), \ldots (S_{i-1}, A_{i-1})\}$ is the sequence of state action pairs prior to (S_i, A_i), then the flow completion reward assigns each of these state action pairs a flow completion reward R_{FC} with an exponentially decaying discount factor γ. Specifically, the flow completion reward for state action pair (S_k, A_k), $FCR(S_k, A_k)$ is given by

$$FCR(S_k, A_k) = R_{FC} \times \gamma^{i-k} \quad (4)$$

D. Other Parameters

The following are a few other important parameters for the RL-Agent based network controller.

a) New State Action Policy: In a new state where rewards for all the actions are initialized to 0, the action can be chosen randomly (RANDOM policy) or it can be chosen following the Earliest Deadline First algorithm (EDF policy).

b) Exploration Probability (p_{exp}): For a pre-visited state, the action is chosen randomly from the set of all possible actions with a probability p_{exp}. The exploration probability is high initially and decays hyper-exponentially to a minimum value of 0.01. This decay is based on step size which is fixed length of simulation time (100000 seconds) (approx. 32000 flows). The $p_{exp}(i)$ for the i$^{\text{th}}$ step is given by

$$p_{exp}(i) = \max(p_{exp}(i-1)^i, 0.01) \quad (5)$$

c) Learning Parameter (β): $Q(S_i, A_i)$ denotes the accumulated reward for a given state action pair (S_i, A_i). When a new reward (SIR or FCR) is determined for a state action pair, the value is updated as follows:

$$Q(S_i, A_i) = \beta * Q(S_i, A_i) + \text{reward}(SIR \text{ or } FCR) \quad (6)$$

V. Results and Discussion

A. Simulation Model

We implemented a simulation model using SimPy which provides a simple yet powerful platform for discrete event simulation. The main approximation in the simulation pertains to the data transfer between the nodes. In the real system, the data transfer is performed using TCP which is a byte stream protocol. The simulation is at the flow level. Requests consist of transfer of files of a certain size with a given deadline. The flows are completely malleable and the progress of a flow is determined by the rate that is allocated to the flow. The simulation model was used to implement all the heuristics as well as the RL-agent based network controller. While there are tools available for implementing RL-agents, in this study, the RL-agent was implemented from scratch in SimPy. The current study is based on a simple Q-learning agent and as such was not difficult to implement.

As mentioned before, each episode consists of N requests, one from each of the sources. Each request is destined to a different destination. Recall that for request j, $Rmin_j(0) = \frac{f_j}{d_j}$ is the minimum rate required at the beginning of the episode for the request to meet the deadline. We define $Sum\_Rmin = \sum_{j=1}^{N} Rmin_j(0)$ to be the sum of the $Rmin$s of all the flows at the beginning of the episode. If Sum_Rmin is less than the bottleneck link capacity B, then it is always possible to meet the deadlines of the flows. This is the underloaded case. If Sum_Rmin is greater than B then it may not be possible to meet the deadlines of all the flows. This is the overloaded case. Experiments are conducted for different values of Sum_Rmin. Given a value of Sum_Rmin, the workload for each episode is generated as follows:

1) For each source i, a filesize f_i is determined by sampling from $\text{Unif}(10, 50)$.
2) Sum_Rmin is randomly partitioned among the N requests, i.e. $Rmin_1, \ldots, Rmin_N$. This study considers only integer values of Sum_Rmin.
3) The deadline of flow i, $d_i = \frac{f_i}{Rmin_i}$.

The RL-agent is trained separately for each Sum_Rmin. We assume that there is an RL-agent for each Sum_Rmin. In the real implementation, given the filesize f_i and the deadline d_i for each of the flow at beginning of the episode, we can determine Sum_Rmin based on which we can call the appropriate RL-agent controller. Table I summarizes parameters used in this study.

TABLE I
TYPICAL VALUES OF PARAMETERS USED IN THIS STUDY

Parameter	Description	Typical values
N	Number of sources (and destination)	3
B	Bottleneck bandwidth	10 Gbps
F	File size (Gb)	Unif(10, 50)
α	Discount factor	0.95
β	Learning rate	1.0
p_{exp}	Initial exploration probability	0.999

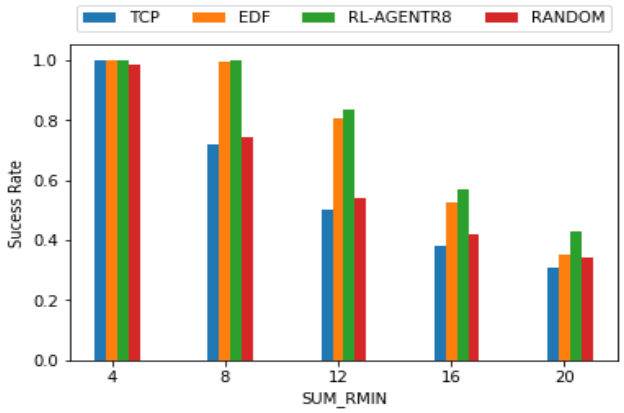

Fig. 3. The success ratio as a function of Sum_Rmin for $N = 3$ sources and $B = 10$ Gbps.

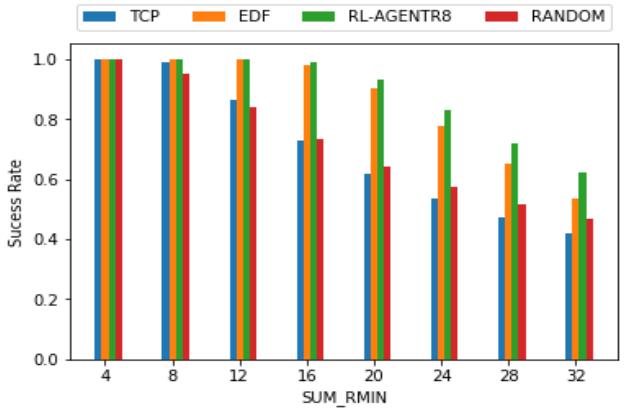

Fig. 4. The success ratio as a function of Sum_Rmin for $N = 5$ sources and $B = 20$ Gbps.

The metrics used for this study is the *success ratio*, which is defined as the fraction of the number of requests that meet the deadline. In the case of the RL-agent, the success ratio is calculated over requests once the exploration probability p has decayed to p_{min} (= 0.01). Requests prior to that are considered to be in the learning phase.

B. Comparison with Heuristics

Figure 3 and Figure 4 show the comparison of performance of our RL-agent with the three heuristics. In the first case $N = 3$ and $B = 10$ $Gbps$, while in the second case $N = 5$ and $B = 20$ $Gbps$. We observe that the RL-agent performs as well as EDF. In fact, for the overloaded case i.e., $Sum\_Rmin > B$ it performs better than EDF. The improvement in the success rate is higher for larger network. This is likely due to the sub-optimality of EDF in the overloaded case which is more pronounced with larger network. As expected, in both cases, TCP performs poorly and becomes worse as the load increases. The reason why the success rate is less than 100% for some of the underloaded cases is that the flow can finish anytime within the scheduling interval. However, new scheduling decisions

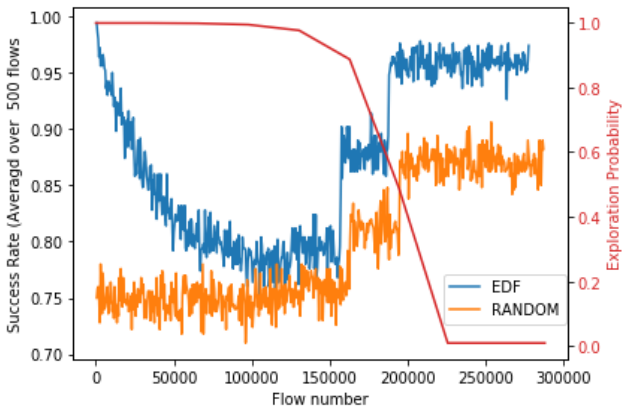

Fig. 5. The learning curve - the success ratio averaged over 500 flows starting from the beginning for the two different new state action policy. $Sum\_Rmin = 8$, $N = 3$, $B = 10$ $Gbps$.

are taken only at the end of a scheduling interval. Hence, there is some bandwidth wastage which becomes more important as Sum_Rmin gets closer to B. Comparing Figure 3 and Figure 4, it appears that for the overloaded case the difference in the success ratio between the RL-agent and EDF increases with larger network and higher B.

C. Impact of New State Action Policy

In a new state with the rewards of all possible actions initialized to 0, the choice of the action can be based on different heuristics. We compared the performance of choosing an action following the EDF policy with the RANDOM policy. Figure 5 shows the success rate averaged over 500 flows starting from the beginning of simulation. The plot also shows the variation of the exploration probability p_{exp}. First, we observe that for the EDF policy, the success ratio first decreases and then increases. This is because initially most of the actions follow the EDF policy which is the optimal policy. This benefit decreases due to the random choice among actions which is often the case when the exploration probability is high. When the exploration probability decreases resulting in more exploitation, the success ratio increases. For the RANDOM policy, the success ratio start at a lower value and then increases when the exploration probability decreases. Second, step increases in the success ratio corresponds large decreases in the exploration probability. Finally, we observe that the RANDOM policy appears to saturate at a lower value than the EDF policy. The difference can be reduced by tuning the RL agent parameters, particularly, the learning factor β.

D. Comparison of Reward Functions

We also compared the performance of the two different scheduling interval reward functions described in Section IV-C. The plot is not included due to space limitation. The results showed that performance of SIRF1 is slightly better than SIRF2. This is due to the fact that SIRF1 does a better job in mimicking EDF.

VI. Related Work

As mentioned in Section I, deadline-aware flow scheduling has been investigated in the context of inter-datacenter traffic [12], [14], [25]. CALIBERS [19] focuses on scientific workflows and the study shows that simple dynamic pacing algorithms that optimizes locally on the most bottleneck link perform as well as more complex algorithms that attempt to optimize globally. However, the study illustrated the complexity of the scheduling problem and the results were not compared with any benchmark. RL-agent based approaches have been used in resource management problems. As mentioned in [16] decisions made by these systems are often highly repetitive they provide the required training data for RL algorithms. This is particularly the case for science workflows such as the ZTF which often generate data periodically. Also, as mentioned in [16], RL agents can be used optimize complex objectives that are hard to model. Finding optimal policies for deadline driven transfers is a complex problem with multihop networks where contention can occur at multiple links [17]. Adding additional objectives such as routing and network utilization makes the problem even more complex which can be modeled using deep neural networks similar to game-playing agents [20].

VII. Conclusions

In this paper, we implemented a Reinforcement Learning (RL) agent based network controller to schedule deadline-driven data transfers. For a single bottleneck link network, we compare the performance of the RL agent with that of known heuristics such as Earliest Deadline First (EDF), which is known to be optimal, and Equal Partitioning, which is an idealized version of TCP. The results show that the RL agent achieves performance comparable to that of EDF, and performs better for the overloaded scenario. The impact of the various parameters of the RL agent suggests that it requires careful tuning to achieve good performance. To scale the RL agent to larger multihop networks, the tabular method can be replaced by an *approximation function* trained online using a deep neural network (DNN) [23]. The results of this paper provide the necessary foundation to pursue such an approach for a general network.

References

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

[2] Laser interferometer gravitational-wave observatory (ligo). https://www.ligo.org.

[3] Oscars: On-demand secure circuits and advance reservation system. http://www.es.net/oscars.

[4] B. P. Abbott, R. Abbott, T. D. Abbott, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, V. B. Adya, C. Affeldt, M. Afrough, B. Agarwal, M. Agathos, K. Agatsuma, N. Aggarwal, and et al. Multi-messenger observations of a binary neutron star merger. *The Astrophysical Journal Letters*, 848(2):L12, 2017.

[5] T. E. Carroll and D. Grosu. Incentive compatible online scheduling of malleable parallel jobs with individual deadlines. In *2010 39th International Conference on Parallel Processing*, pages 516–524, Sep. 2010.

[6] Huajun Chai, H Michael Zhang, Dipak Ghosal, and Chen-Nee Chuah. Dynamic traffic routing in a network with adaptive signal control. *Transportation Research Part C: Emerging Technologies*, 85:64–85, 2017.

[7] 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.

[8] European Gravitational Observatory (EGO). Virgo detector. http://www.virgo-gw.eu/.

[9] Dror G Feitelson, Larry Rudolph, Uwe Schwiegelshohn, Kenneth C Sevcik, and Parkson Wong. Theory and practice in parallel job scheduling. In *Workshop on Job Scheduling Strategies for Parallel Processing*, pages 1–34. Springer, 1997.

[10] Robert Glaubius, Terry Tidwell, Christopher Gill, and William D Smart. Real-time scheduling via reinforcement learning. *arXiv preprint arXiv:1203.3481*, 2012.

[11] 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. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 3–14. ACM, 2013.

[12] Srikanth Kandula, Ishai Menache, Roy Schwartz, and Spandana Raj Babbula. Calendaring for wide area networks. In *ACM SIGCOMM computer communication review*, volume 44, pages 515–526. ACM, 2014.

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

[14] Alok Kumar, Sushant Jain, Uday Naik, Anand Raghuraman, Nikhil Kasinadhuni, Enrique Cauich Zermeno, C Stephen Gunn, Jing Ai, Björn Carlin, Mihai Amarandei-Stavila, et al. Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 1–14. ACM, 2015.

[15] James Kurose and Keith Ross. Computer networks: A top down approach featuring the internet. *Peorsoim Addison Wesley*, 2016.

[16] 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.

[17] 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.

[18] Markos Papageorgiou, Christina Diakaki, Vaya Dinopoulou, Apostolos Kotsialos, and Yibing Wang. Review of road traffic control strategies. *Proceedings of the IEEE*, 91(12):2043–2067, 2003.

[19] Eric Pouyoul, Mariam Kiran, Nathan Hanford, Dipak Ghosal, Fatemah Alali, Raj Kettimuthu, and Ben Mackcrane. Calibers: A bandwidth calendaring paradigm for science workflows. In *INDIS Innovating the Network for Data Intensive Science*, 2017. https://scinet.supercomputing.org/workshop/sites/default/files/Pouyoul-Calibers.pdf.

[20] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[21] Roger M Smith, Richard G Dekany, Christopher Bebek, Eric Bellm, Khanh Bui, John Cromer, Paul Gardner, Matthew Hoff, Stephen Kaye, Shrinivas Kulkarni, Andrew Lambert, Michael Levi, and Dan Reiley. The Zwicky transient facility observing system. In *Ground-based and Airborne Instrumentation for Astronomy V*, volume 9147. International Society for Optics and Photonics, 2014.

[22] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[23] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[24] 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.

[25] 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.