M-DRL: Deep Reinforcement Learning Based Coflow Traffic Scheduler with MLFQ Threshold Adaption

Tianba Chen, Wei Li, Yukang Sun, Yunchun Li

M-DRL: Deep Reinforcement Learning based Coflow Traffic Scheduler with MLFQ threshold adaption

Tianba Chen, Wei Li, YuKang Sun, and Yunchun Li

Beijing Key Lab of Network Technology, School of Computer Science and Engineering
Beihang University, Beijing, China
{chentb,liw,sunyk2115,lych}@buaa.edu.cn

Abstract. The coflow scheduling in data-parallel clusters can improve application-level communication performance. The existing coflow scheduling method without prior knowledge usually uses Multi-Level Feedback Queue(MLFQ) with fixed threshold parameters, which is insensitive to coflow traffic characteristics. Manual adjustment of the threshold parameters for different application scenarios often has long optimization period and is coarse in optimization granularity. We propose M-DRL, a deep reinforcement learning based coflow traffic scheduler by dynamically setting thresholds of MLFQ to adapt to the coflow traffic characteristics, and reduces the average coflow completion time. Trace-driven simulations on the public dataset show that coflow communication stages using M-DRL complete 2.08x(6.48x) and 1.36x(1.25x) faster on average coflow completion time (95-th percentile) in comparison to per-flow fairness and Aalo, and is comparable to SEBF with prior knowledge.

Keywords: coflow, datacenter network, deep reinforcement learning

1 Introduction

With the rapid development of big data analytics application, the parallel flows abstracted as coflow[3, 13, 14, 6], appear in the datacenter network, which means that a communication stage completes only when all the flows are completed. A typical scenario is the shuffle in MapReduce and Spark. The optimization of traditional datacenter network is mostly based on packet-level and flow-level, and the datacenter network cannot perceive the communication requirement in parallel computing[3, 13, 14]. In recent years, some work such as Varys[6], Aalo[4] and CODA[15] have studied how to optimize the datacenter network based on coflow. The coflow scheduling without prior knowledge, Aalo[4] and CODA[15] overcome the shortcomings of Varys[6] requiring prior knowledge of coflow and use discretized Coflow-Aware Least-Attained Service (CLAS)[4] based on MLFQ[4, 15] to centrally schedule coflows. In these methods, MLFQ is used to divide coflows into different queues according to the coflow size. CLAS sensitive to coflow size is adopted to scheduling between different queues, First-In

First-Out (FIFO) sensitive to the arrival time of coflow is used within the same queue, and queue thresholds of MLFQ balance the overall performance of coflow scheduling. The fixed threshold of MLFQ queue is not sensitive to the change of traffic characteristics. Dynamics in complicated network environment make discretized CLAS scheduler based on MLFQ unable to achieve maximum scheduling performance. However, manual parameter optimization has a long cycle and is coarse in optimization granularity[2].

In order to deal with this problem, we introduce deep reinforcement learning (DRL) to automatically adjust MLFQ thresholds. DRL is a method for processing sequence decision-making in machine learning[7]. Some recent studies have applied DRL to network scheduling[2, 10, 12]. DRL uses neural networks with strong generalization ability to process raw and noisy inputs and optimizes the target through reward signals indirectly, suitable for dealing with network scheduling problems[2, 10, 12].

Therefore, we propose M-DRL, a deep reinforcement learning based coflow traffic scheduler to dynamically setting thresholds of MLFQ, which is driven by network traffic. Trace-driven simulations on the public dataset show that communication stages using M-DRL complete 2.08x and 1.36x faster on average coflow completion time in comparison to per-flow fairness and Aalo, and M-DRL's performance is comparable to that of SEBF using prior knowledge.

In summary, the main contribution of this paper include:

- It is the first time to model MLFQ-based coflow scheduling problem with DRL, and design state, action and reward function of DRL.
- We propose a data-driven method, M-DRL, adopting DDPG to adaptively adjust MLFQ queue thresholds to reduce the average coflow completion time.
- We conduct extensive simulations to evaluate the performance of M-DRL. The results of the experiments show that M-DRL outperforms per-flow fairness and Aalo, and is comparable to SEBF(the state-of-the-art method with prior knowledge) .

This paper is organized as follows: Section 2 describes the existing problem in the discretized CLAS of coflow scheduling. Modeling the coflow scheduling problem with DRL and algorithm design of M-DRL are described in Section 3. Section 4 evaluate our solutions. Section 5 reviews related works. Finally, the conclusion and future prospects of this paper is presented in Section 6.

2 Coflow scheduling model without prior knowledge

2.1 Background

Coflow is a collection of flows between different nodes in the cluster of datacenter network. A coflow completes only when all flows are completed. If a coflow completes faster, then corresponding job completes faster. So optimization for coflow scheduling can improve the communication performance of the application[3, 13, 14]. To simplify our analysis, the entire datacenter fabric structure is abstracted as one non-blocking switch, which performs well in practice[6, 4, 15].

In some scenarios, the information about coflow size is difficult to obtain[4, 15] and we need to assume that some characteristics of coflow are unknown, such as endpoints, the size of each flow, their arrival times. This kind of coflow scheduling when the characteristics of coflow is unknown is called coflow scheduling without prior knowledge[14]. Generally, only real-time information of coflow can be used, including sent size, width, arrival time and duration of coflow.

2.2 Coflow scheduling based on MLFQ

Coflow scheduling methods such as Aalo[4] and CODA[15], use discrete Coflow-Aware Least-Attained Service based on MLFQ[4]. MLFQ consists of K queues $(Q_1, Q_2, \ldots, Q_K)$, with queue priority decreasing from the first queue Q_1 to the last queue Q_K. There are $K-1$ thresholds $(th_1, th_2, \ldots, th_{K-1})$ between K queues and the i-th queue contains coflows of size within $[th_{i-1}, th_i)$. Note that $th_0 = 0$, and $th_K = \infty$. Actions taken during three lifecycle events determine the priority of a coflow.

1) New coflows enter the highest priority queue Q_1 when they start.
2) A coflow is demoted from Q_i to Q_{i+1}, when its size exceeds threshold th_i.
3) Coflows are removed from their current queues upon completion.

Preemptive priority scheduling is used between different queues. High priority queues are prioritized and different coflows in the same queue use FIFO method. The discretized CLAS uses MLFQ to divide coflows into different queues and assigns coflow priority in terms of coflow size and arrival time.

CLAS is appropriate for traffic with heavy-tailed distribution[6], and FIFO is appropriate for traffic with light-tailed distribution[5]. The discretized CLAS based on MLFQ combines the CLAS with FIFO to better communication performance. The arrival time of coflow has a greater impact on scheduling performance for the FIFO scheduling within the same queue. Coarse-grained setting of queue threshold will reduce the performance of FIFO.

For example[4], the number of MLFQ queues is set to 10, and the formal queue threshold is defined as $th_{i+1} = E \times th_i$, where $E = 10$ and $th_1 = 10MB$. Coflow can be assigned to different queues according to the magnitude of coflow size, where

$$th_{i+1} - th_i = E \times (th_i - th_{i-1}) \tag{1}$$

The range of coflow size in queue Q_{i+1} is E times that in queue Q_i.

The experiments shows that during the scheduling process, more than 95% of coflows utilize only three priorities in MLFQ and the rest queues are empty. It's obvious that the granularity of threshold setting is too coarse, and the discretized CLAS degenerates into FIFO to some extent. Therefore, it is necessary to provide data-driven MLFQ threshold setting for different communication scenarios.

3 Data-driven MLFQ threshold setting using DRL

Manual MLFQ threshold setting often requires traffic statistics collection, offline model analysis, parameter optimization and so on, which has a long design

cycle[2] and can only be optimized from the global distribution characteristics. Therefore, a data-driven threshold setting method can greatly reduce overhead of parameter optimization, and can improve coflow scheduling performance in a finer granularity.

We use deep reinforcement learning to model the coflow scheduling problem and design an M-DRL scheduler to make decisions based on the network status automatically. The algorithm DDPG[9] is used in M-DRL to optimize the MLFQ threshold setting.

3.1 DRL formulation

We construct the coflow scheduling problem as an MDP of continuous state and continuous action in discrete time. By training historical scheduling information, we learn the optimal scheduling policy π, which defines state $s_t \in S$, action $a_t \in A$ and reward function r_t.

In the process of coflow scheduling, M-DRL controls the setting of MLFQ threshold through coflow status in the network environment. By continuously sampling and training, agent learns the appropriate MLFQ threshold setting policy, which reduces the average coflow completion time. M-DRL regards the coflow scheduling process with one hour as an episode. When an episode ends, the initial state is reset. An episode is divided into many time steps and each time step Δt is set to ten seconds. The agent obtains the state s_t from the environment every time Δt, generates an action control a_t according to the learned policy, and gets the immediate reward r_t.

State Space The status in network scheduling is taken as state in units of coflow, including the identifier id(unique ID in cluster), coflow width $width$(the number of flows included in the coflow), sent size of coflow $sent$(current transmission volume of the coflow) and duration of coflow $duration$(spent time from the arrival time of the coflow). These attributes are realated to the size distribution of coflow. At the time t, the number of scheduled coflows is not fixed. We limit the maximum number of coflows represented in the state to N. The coflow in state is sorted increasingly according to the sent size. When the number of coflows is greater than N, the first N coflows represent state. We use zero padding when the number of coflow is less than N. Thus, the state at time t is represented as

$$s_t = [id_1, width_1, sent_1, duration_1, \ldots, id_N, width_N, sent_N, duration_N] \quad (2)$$

Where N is set to 10, and the dimension of state is $4 \times N = 40$ because every coflow has four different attributes.

Action Space Action is represented by MLFQ threshold. K queues have $K-1$ thresholds $(th_1, th_2, \ldots, th_{K-1})$, and action a_t should be represented by $K-1$ components

$$a_t = [th_1, th_2, \ldots, th_{K-1}] \quad (3)$$

In the experiment, K is 10 and the dimension of a_t is 9. Different MLFQ threshold settings can make the average coflow completion time different.

Reward Since the optimization goal of M-DRL is to reduce the average completion time of coflows in an episode, we define finished coflow set and active coflow set at time t as CS_t^f and CS_t^a, then the average coflow completion time until the time t is

$$acct_t = \frac{1}{N_t^f} \sum_{c \in CS_t^f} cct^c \tag{4}$$

Where N_t^f represents the number of coflows in finished coflow set and cct^c represents the completion time of coflow c. The average coflow duration time is defined as

$$acdt_t = \frac{1}{N_t^f + N_t^a} (\sum_{c \in CS_t^f} cct^c + \sum_{c \in CS_t^a} cdt^c) \tag{5}$$

Where N_t^a represents the number of coflows in active coflow set, and cdt^c represents the duration of coflow c.

$acct_t$ estimates average coflow completion time of the entire episode until state s_t, and $acdt_t$ estimates the effect of coflow scheduling. Therefore, the deviation $|acdt_t - acct_{t-1}|$ represents the contribution degree of current action at time t to the goal. Thus reward function can be defined as

$$r_t = -(acdt_t - acct_{t-1}) \tag{6}$$

3.2 Algorithm Design

We use DDPG[9] with Actor-Critic structure to optimize MLFQ threshold setting, and the algorithm is implemented using popular machine learning framework TensorFlow.

The algorithm uses the Actor-Critic network structure to deal with continuous actions efficiently. Actor is a policy network that generates actions in a given state s, including actor $\mu(s|\theta^\mu)$ and target actor $\mu^{'}(s|\theta^{\mu^{'}})$. Critic is the Q network including critic $Q(s,a|\theta^Q)$ and target critic $Q^{'}(s,a|\theta^{Q^{'}})$, evaluating the value of state-action pair. The role of target network $\mu^{'}(s|\theta^{\mu^{'}})$ and $Q^{'}(s,a|\theta^{Q^{'}})$ is to eliminate the deviation caused by overestimation and stabilize the policy and Q network, thereby improving the stability of training. Soft update in Equation 7 and 8 is used to update the target networks periodically

$$\theta^{\mu^{'}} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu^{'}} \tag{7}$$

$$\theta^{Q^{'}} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q^{'}} \tag{8}$$

and the coefficient τ is 0.001. Both actor and critic network use a two fully-connected hidden layers with 600 and 600 neurons[2] respectively to fit the scheduling strategy fully. The whole algorithm consists of two parts, online interaction and offline training.

Online interaction During the online interaction with the environment as shown in Figure 1, actor $\mu(s|\theta^{\mu})$ generates current action a_t according to state s_t, acting on the environment to generate the next state s_{t+1} and immediate reward r_t. This process is repeated until the end state is reached and transition $\{s_t, a_t, r_t, s_{t+1}\}$ is stored in experience replay buffer during each iteration. Here the OU noise $\mathcal{N}$ is added to the action a_t to increase the early random exploration for environment, as shown in Equation 9

$$a_t = \mu(s|\theta^{\mu}) + \mathcal{N}_t \tag{9}$$

With the advance of training, the influence of noise will gradually decrease.

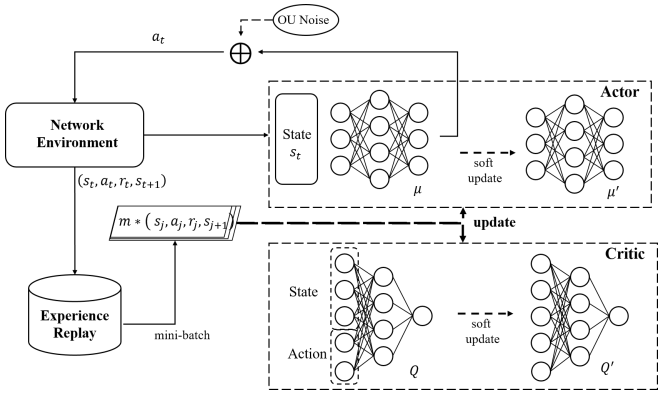


Fig. 1. Algorithm architecture based on DDPG

Offline training In the offline training, sample m transitions $\{s_j, a_j, r_j, s_{j+1}\}, j = 1, 2, \ldots, m$ from experience replay buffer where batch size of sampling is 32 and size of experience replay buffer is 10000. Then calculate the target Q value to estimate state s_j according to Equation 10

$$y_j = r_j + \gamma Q^{'}(s_{j+1}, \mu^{'}(s_{j+1}|\theta^{\mu^{'}})|\theta^{Q^{'}}) \tag{10}$$

The discount factor γ is 0.99. Next, mean squared loss function in Equation 11 is minimized to update parameters of critic network

$$\frac{1}{m}\sum_{j=1}^{m}(y_j - Q(s_j, a_j|\theta^{Q}))^2 \tag{11}$$

And the sampled gradients are calculated to update parameters of actor network in Equation 12

$$\bigtriangledown_{\theta\mu} J \approx \frac{1}{m}\sum_{j=1}^{m} \bigtriangledown_a Q(s, a|\theta^{Q})|_{s=s_j, a=\mu(s_j)} \bigtriangledown_{\theta\mu} \mu(s|\theta^{\mu})|_{s_j} \tag{12}$$

During the update, Adam optimizer is used for the gradient descent and the learning rate of actor and critic are 0.0001 and 0.001, respectively.

The setup of the coefficient, discount factor and learning rate in the algorithm refers to the paper[9]. M-DRL uses algorithm DDPG to obtain coflow status from the environment, apply MLFQ threshold actions, and repeatedly try to learn optimized MLFQ threshold setting policy, which reduces the average coflow completion time.

4 Evaluation

We conduct a series of experiments on M-DRL using production cluster trace and industrial Benchmark[6, 4]. We describe dataset for experiments, evaluation metrics and the simulator used by M-DRL. Then compare with the state-of-the-art methods of coflow scheduling to show the performance improvement of M-DRL.

4.1 Workload

Two workloads are used in our experiment. One is based on Facebook's open source trace[6, 4], which contains 526 coflows from 150 racks and 3000 machines, with more than 70,000 flows. And size of each flow is between 1MB and 10TB. In addition, in order to demonstrate the adaptation of M-DRL to different coflow distributions, we generate the coflow set with light-tailed distribution based on Facebook trace.

We categorize coflows based on their lengths(size of the longest flow in the coflow) and widths. We consider a coflow to be short if its length is less than 5MB and narrow if its width is less than 50. Otherwise, it is long and wide. Table 1 shows four categories Short&Narrow(SN), Long&Narrow(LN), Short&Wide(SW) and Long&Wide(LW).

Table 1. Coflows binned by their length(**S**hort and **L**ong) and their width(**N**arrow and **W**ide)

Workload	**Coflow Bin**	**1(SN)**	**2(LN)**	**3(SW)**	**4(LW)**
Facebook	**% of Coflows**	60%	16%	12%	12%
	% of Bytes	0.01%	0.11%	0.88%	99.0%
LightTail	**% of Coflows**	0%	76%	5%	19%
	% of Bytes	0%	44.21%	0.07%	55.72%

In the Facebook workload, 12% of coflows contributes 99% of the total bytes(LW) and 60% of coflows belongs to SN and its load is only 0.01%. It shows this workload follows heavy-tail distribution and the heavy-tail characteristic are obvious. In the LightTail workload, 95% of coflows is long, accounting for 99% of total bytes. This workload has a light-tailed distribution and 80% of

coflow size is between 1GB to 1TB. Compared with the Facebook workload, the LightTail workload has a smaller distribution range in coflow size and its coflow size is larger.

4.2 Evaluation metrics

Varys[6] and Aalo[4] use a flow-level simulator in their evaluation. For fair comparison, we use the same simulator as Varys for trace replay and implement the common interface of RL provided by OpenAI Gym[1] which is used for scheduling and training of M-DRL.

The main evaluation metric is the improvement of average coflow completion time (CCT) in the workload. Normalized CCT is used to evaluate for fairness:

$$Normalized\ Comp.\ Time = \frac{Compared\ Duration}{M - DRL's\ Duration} \tag{13}$$

If normalized CCT is larger (smaller), then M-DRL is faster (slower). Obviously, if normalized CCT is greater than 1, it means that M-DRL has better communication performance than the compared algorithm, otherwise M-DRL performance is worse.

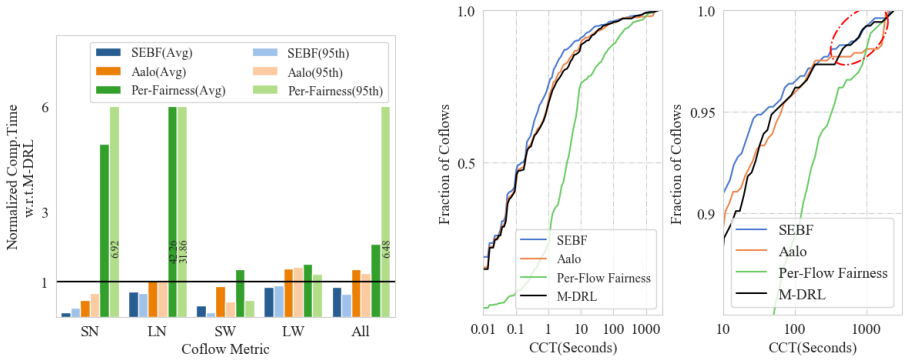
4.3 Simulation Result

To verify the improvement of M-DRL, we conduct contrast experiments on the Facebook and LightTail workload and compare M-DRL against SEBF[6] (the state-of-the-art coflow scheduling method with prior knowledge), per-flow fairness and Aalo.

Comparison of CCT Some experiments are performed on Facebook dataset to verify the effectiveness of M-DRL, per-flow fairness, Aalo and SEBF and the results are shown in Figure 2.

As shown in Figure 2(a), M-DRL reduces the average and 95th percentile completion times of the overall coflows by up to 2.08x and 6.48x, respectively, in comparison to TCP-based per-flow fairness. From the performance of per-flow fairness on SN, LN, SW, LW (as shown in Table 1), M-DRL ignores characteristics of coflow length and width and has better performance. It can also be further proved in Figure 2(b) that M-DRL is better than per-flow fairness in every CCT range. In general, per-flow fairness does not consider the characteristics of coflow, only performs flow-level scheduling and treats all flows fairly, performing the worst among the four methods.

As expected, M-DRL performs better in the overall coflows with the average completion times improved by 1.36x and 95th percentile completion times improved by 1.25x than Aalo. In detail, M-DRL outperforms Aalo on more than 99% of long coflows, LN and LW, and performs worse on SN and SW(Figure 2(a)). In fact, coflow size is not evenly distributed on the exponent and long coflows appear more frequently in MLFQ. And M-DRL uses MLFQ thresholds



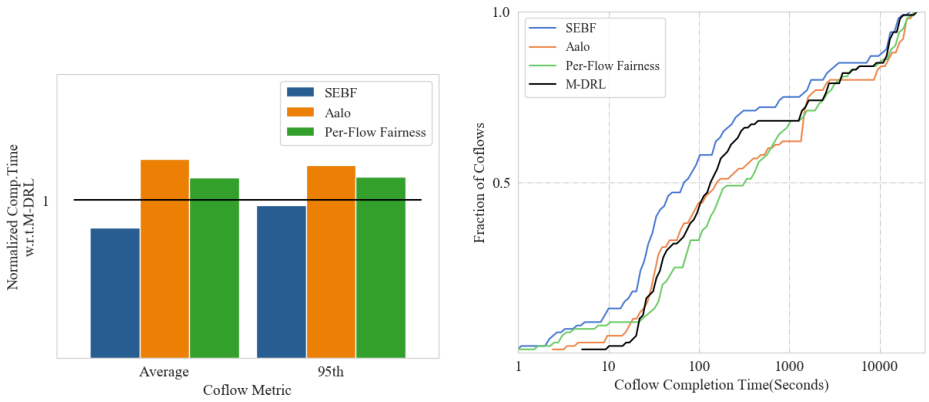
(a) CCT Comparison (b) CCT Distributions

Fig. 2. [Facebook] Improvements of M-DRL: (a) Average and 95th percentile improvements in communication completion times using M-DRL over SEBF, Aalo and per-flow fairness. The black line represents M-DRL.(b) CCT distributions for SEBF, Aalo, per-flow fairness and M-DRL mechanism. The X-axis is in log scale.

sensitive to coflow size through learning in the scheduling to make coflows more evenly distributed in the queues, which means that more queues are allocated for long coflows. Therefore, M-DRL sacrifices the performance of short coflows to schedule the long coflows better and improves the overall communication performance. In Figure 2(b), M-DRL is better than Aalo in the range of CCT 300-2500 seconds (red ellipse). This is because the allocation of more queues for long coflows in M-DRL has a greater impact on the scheduling of long coflows.

Compared to SEBF, M-DRL reduces the average and 95th percentile completion times of the overall coflows by up to 0.84x and 0.66x, respectively, as shown in Figure 2(a). Although M-DRL is not as good as SEBF on short coflows and narrow coflows, M-DRL achieves improvements of 0.86x and 0.90x respectively on the long and wide coflow, LW, which is 99% of bytes. SEBF can calculate the remaining processing time of each coflow through prior knowledge that is difficult to obtain in some scenarios. And M-DRL divides a certain range of coflows into the same queue which are scheduled by FIFO, and reduces the CCT of short coflows.

Adaptation of M-DRL We conduct experiments and retrain M-DRL on the LightTail dataset to verify its adaptation and results are shown in Figure 3. Figure 3(a) shows that M-DRL reduces the average and 95th percentile completion times of the overall coflows by up to 1.15x and 1.15x, respectively, in comparison to per-flow fairness. M-DRL is superior when CCT is greater than 20 seconds (Figure 3(b)). Per-flow fairness treats each coflows fairly and increases the CCT of shorter coflows. However, M-DRL decreases the CCT of shorter coflows though priority scheduling in M-DRL. For coflows with CCT less than 20 sec-



(a) CCT Comparison (b) CCT Distributions

Fig. 3. [LightTail] (a) Average and 95th percentile improvements in communication completion times using M-DRL over SEBF, Aalo and per-flow fairness. (b) CCT distributions for SEBF, Aalo, per-flow fairness and M-DRL mechanism. The X-axis is in log scale.

onds(Figure 3(b)), M-DRL uses MLFQ to make both short and long coflows in the same queue and performs worse by FIFO scheduling.

Compared to Aalo, M-DRL reduces the average and 95th percentile completion times of the overall coflows by up to 1.26x and 1.23x, respectively (Figure 3(a)). As can be seen in Figure 3(b), M-DRL has better performance than Aalo in the range of CCT greater than 100 seconds, which shows that the fine-grained division of MLFQ thresholds by M-DRL still has an advantage in large coflows of the dataset. Figure 3(a) shows that M-DRL has comparable performance to SEBF with average and 95th percentile improvements being 0.83x and 0.97x. Especially, M-DRL has comparable performance to SEBF in the range of CCT greater than 3000 seconds (Figure 3(b)).

In general, M-DRL uses finer granularity to prioritize coflows by learning, showing superior performance than per-flow fairness and Aalo. This demonstrates that M-DRL has a strong adaptation.

5 Related work

Chowdhury[3] proposed an abstraction of coflow which represents a collection of semantic-related flows to convey requirements of job-specific communications. Later, many coflow-based scheduling methods were proposed to improve the performance of cluster computing applications. Orchestra[5] adopts the First-In First-out strategy to implement inter-coflow scheduling. Varys[6] uses the Smallest-Effective-Bottleneck-First (SEBF) heuristic to greedily schedule coflows

assuming that prior knowledge of coflow is known. In the scenario without prior knowledge, Aalo[4] uses the discretized Coflow-Aware Least-Attained Service with Multi-Level Feedback Queue to assign priority for coflows. Further assuming that the structure of coflows is unknown, CODA[15] uses machine learning algorithm to extract coflow information and propose an error-tolerant scheduling to mitigate occasional identification errors. Moreover, CS-DP[8] solves the problem of CODA's sensitivity to parameters.

Aalo[4], CODA[15], and CS-DP[8] all use MLFQ to provide the service to coflow, but they do not take into account the impact of MLFQ threshold setting on coflow scheduling. The reasonable MLFQ threshold setting is closely related to the distribution of coflow size, but in complicated datacenter network scenarios, the distribution of coflow size is variable, so we need a coflow scheduling method that is sensitive to the coflow size distribution.

In recent years, deep reinforcement learning[7] has been applied to architecture design[11], flow control[2] and network scheduling[10, 12]. In traffic scheduling, AuTO[2] uses DDPG to make flow scheduler to adapt to the different pattern of traffic, which directly inspired us to apply DRL to optimize the coflow scheduling problem. In congestion control, MVFST-RL[12] proposes a DRL-based congestion control framework to adapt to changes in network congestion scenarios, dynamically control the transmission rate of each node to maximize the overall throughout and minimize delay and packet loss rate. In job scheduling, Decima[10] uses a graph neural network structure to process job and cluster information as the state input of algorithm PG to adapt to different cluster applications of the big data computing framework.

6 Conclusion

This paper proposes a DRL-based MLFQ threshold adaption method which is driven by coflow traffic within network environment. It realizes an adaptation to dynamic network environment by automatically setting the MLFQ threshold. Experiments show that M-DRL has comparable SEBF communication performance and performs better than Aalo and per-flow fairness.

In the future, we will toward to directly optimize the priority setting for every coflow in scheduling with deep reinforcement learning, and deploying the RL agent in an operational network to validate the effectiveness. We also plan to improve the performance of M-DRL in terms of neural network structure and better DRL algorithms for continuous action control, and compare with other non-AI scheduling to verify the performance of M-DRL.

Acknowledgements

This work is supported by the National Key Research and Development Program of China (Grant No. 2016YFB1000304) and National Natural Science Foundation of China (Grant No. 1636208).

References

1. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
2. Li Chen, Justinas Lingys, Kai Chen, and Feng 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*, pages 191–205, 2018.
3. Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36, 2012.
4. Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. *ACM SIGCOMM Computer Communication Review*, 45(4):393–406, 2015.
5. Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. Managing data transfers in computer clusters with orchestra. *ACM SIGCOMM Computer Communication Review*, 41(4):98–109, 2011.
6. Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 443–454, 2014.
7. Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *arXiv preprint arXiv:1811.12560*, 2018.
8. Chenghao Li, Huyin Zhang, and Tianying Zhou. Coflow scheduling algorithm based density peaks clustering. *Future Generation Computer Systems*, 97:805–813, 2019.
9. 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.
10. Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrishnan, Zili Meng, and Mohammad Alizadeh. Learning scheduling algorithms for data processing clusters. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 270–288. 2019.
11. Drew D Penney and Lizhong Chen. A survey of machine learning applied to computer architecture design. *arXiv preprint arXiv:1909.12373*, 2019.
12. Viswanath Sivakumar, Tim Rocktäschel, Alexander H Miller, Heinrich Küttler, Nantas Nardelli, Mike Rabbat, Joelle Pineau, and Sebastian Riedel. Mvfst-rl: An asynchronous rl framework for congestion control with delayed actions. *arXiv preprint arXiv:1910.04054*, 2019.
13. Kun Wang, Qihua Zhou, Song Guo, and Jiangtao Luo. Cluster frameworks for efficient scheduling and resource allocation in data center networks: A survey. *IEEE Communications Surveys & Tutorials*, 20(4):3560–3580, 2018.
14. Shuo Wang, Jiao Zhang, Tao Huang, Jiang Liu, Tian Pan, and Yunjie Liu. A survey of coflow scheduling schemes for data center networks. *IEEE Communications Magazine*, 56(6):179–185, 2018.
15. Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. Coda: Toward automatically identifying and scheduling coflows in the dark. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 160–173, 2016.