JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 38, XXX-XXX (2022)
DOI: 10.6688/JISE.20220X_38(X).00XX

LBRS：A Reinforcement Learning approach to Mixed Flow Scheduling in Data Center Networks*

XING-YAN ZHANG[1,2*]
1 College of Computer, Hubei University of Education, Wuhan 430205, China
2 Hubei Co-Innovation Center of Basic Education Information Technology Services, Hubei University of Education, Wuhan 430205, China
E-mail: {Zhangxy}@hue.edu.cn

Abstract: Deploying enterprise-level applications on cloud servers helps saving delivery and management costs in data centers. These applications generate lots of flows between servers within the data center. Efficient flow transmission between the servers can improve the utilization of data centers and has become a research hotspot in both academia and industry. However, the different requirement from these complex composed flows is the main obstacle for efficient flow scheduling for DCN.

In this study, a Link-state-based Bandwidth-Reservation Scheme (LBRS) is proposed to reduce the performance loss caused by the performance requirement of these mixed flows. Firstly, the bandwidth reservation mechanism is set to limit the occupancy of link bandwidth by elephant flows, thereby ensuring the transmission performance of mice flows. Secondly, the recursive path selection algorithm is designed to redirect paths for elephant flow, when too many elephant flows occupy overload link bandwidth. Finally, reinforcement learning is leveraged to train optimal configurations for better performance. Experimental results have shown that LBRS offers better performance. Compared to previous flow scheduling, such as queue management and buffer management, LBRS can reduce the latency of mice flows while increasing network throughput. In addition, intelligent technology can increase the efficiency of flow scheduling and network management, and provide reference and convenience for intelligent network management.

Keywords: **link**-bandwidth reservation, recursive path selection, model-based reinforcement learning

1. INTRODUCTION

In recent, cloud computing has exhibited high availability and high scalability. Deploying enterprise-level applications on cloud servers is helps saving delivery and management costs on data centers [1]. These applications generate lots of flows between servers within the data center. Efficient flow transmission between the servers can improve the utilization of data centers and has become a research hotspot in both academia and industry.

The flow in data center network (DCN) can be divided into mice flow and elephant flow based on the flow transmission durations [2]. Some applications (e.g. instant messaging applications, and online social networks) generate a large number of flows, and these flows need to be processed before the deadline [3]. The main performance goal is minimizing flow completion time (FCT). Some other applications (e.g. Hadoop MapReduce) generate elephant flows, which require high throughput for a long-time flow transmission. Furthermore, flow generated from online video applications [5] requires bandwidth and deadline performance. The different requirement from the complex composed flow is the main obstacle for efficient flow scheduling for DCN.

In general, adjusting the flow priority can meet the transmission latency requirements of the deadline-aware applications. When the network traffic approaches the overall throughput, it may face some problems. Firstly, not all priorities of flow can be predicted [7, 8], resulting in the flow of low-priority applications constantly under-waiting for scheduling and the performance being ignored. Additionally, the flows of low-priority applications which wait for transmission [9] occupy excessive switch cache, leading to a decline in overall network performance. Another method is the management of the switch buffer. By detecting elephant flows in the switch, the elephant flow can be quickly forwarded out through multi-path methods [4] for increasing network throughput. However, it is easy to ignore the deadline requirements of the mice flow [10]. The two-tier scheduling scheme can optimize the performance of both mice flow and elephant flow, but the prediction of flow priority can't be avoided [11]. In conclusion, the main obstacle to mixed flow scheduling is the bandwidth competition from the complex requirement flow.

In this study, a Link-state-based Bandwidth-Reservation Strategy (LBRS) is proposed to improve the FCT of mixed flow and overall throughput. The parameter, which is named the maximum number of elephants flow for each link, is set to ensure enough bandwidth for mice flow transmission. When the number of elephant flows on a link exceeds the parameter, the idle neighbor switch is selected for an alternative path, as to redirect the path for the elephant flow. Moreover, more information in flow scheduling decisions is helpful for better performance. Intelligence technology, such as the reinforcement learning [12, 13, 14], can improve network performance for data center flow scheduling. The analyses and experiment results indicate that the model-based RL algorithms can improve scalability and adaptability. The main contributions of this study are as follows.

- The link-bandwidth reservation scheme of flow scheduling can satisfy the performance requirements of both mice flow and elephant flow. We limited the count of elephant flow on links, and set the trigger for flow split and multipath redirection. The configuration parameter is combined with the ECMP protocol, which improves the flow scheduling efficiency.
- The reinforcement learning is used to training the optimal parameters for network configuration achieved intelligent parameter setting. The model-based is helpful for parameter training and performance convergence.
- The performance and simulation of LBRS show that it can reduce the average FCT and improve the overall throughput. The intelligent technology can increase the efficiency of flow scheduling and network management, and provide reference and convenience for intelligent network management.

The organizational structure of this article is as follows: the second section describes the related works, and the LBRS model is shown in the third section, the next section introduces the specific design and implementation method, the fifth section presents performance and analysis, and finally, we conclude the work and future works.

2. RELATED WORKS

The main resolution on flow scheduling in DCN can be divided into queue management and buffer management. The main idea of queue management is to adjust the priority of mixed flow. The P4-MLFQ [6] schedules incoming flow into numerous priority queues according to the active time of flows, resulting in short queuing delays for short flows. P4-MLFQ improves the overall throughput in average and FCT for 99th percentile short flows, which makes it a suitable scheduler for high-speed DCNs. EAshman [7] proposed a flow scheduling framework only to schedule the elephant flows to the lightly congested links. The method of probability-based path selection algorithm can improve network throughput. This method only focuses on the throughput improvement from the elephant flows and ignores the impact on mice. The proposed DCI-NACC [8] optimizes the priority queue to improve overall throughput, network latency and FCT. It can improve the performance of the DCNs based on lossless transmission, which ignores the effect of elephant flow transmission.

The method of buffer management utilizes the properties of switch buffers to adjust routing strategies and flow forwarding. PFLBS [9] defines a port-based source-routing addressing scheme, and designs a buffer trigger mechanism to split the elephant flow to keep the traffic load balanced. The main goal is increasing the throughput of elephant flow. The R-AQM [10] mechanism can prevent TCP from suffering incast collapse, which can reduce retransmission timeouts and forward queuing delays.

There are some flow scheduling strategies that rely on more parameters. The ReQ-tank [11] implements a two-tier scheduling scheme to reduce the FCT. On the flow scheduling layer, priority queues are segmented into two categories, and the flows within high-priority queues follow a strict priority scheduling, while those in low-priority queues adhere to differential weighted Round-robin scheduling; On the packet scheduling layer, ReQ-tank modifies the priority of initially high-priority re-transmitted packets to facilitate fine-grained data packet scheduling.

The common goal of these strategies is to improve the flow scheduling performance. Queue management and buffer management can both reduce the latency of mice flows and increase the total throughput of the network, but it’s difficult to achieve both targets.

Recently, artificial intelligence technology has been wildly used to improve the network performance in DCNs. The proposed DRL-R [12] redesigns the routing algorithm in software-defined data-center networks, which adds the cache information to the flow scheduler. The DRL-Plink [13] designs some private links to divide the link bandwidth and establishes some corresponding private links to prevent bandwidth competition for different flows. These proposed using deep reinforcement learning, which needs less time and fewer computing sources for model training. The proposal of LBRS can try to use reinforcement learning get the almost performance without less training cost. The reinforcement learning [14] can used to optimize multi-object flow scheduling. It proved that reinforcement learning or something like artificial intelligence is the trend to optimize flow scheduling, especially for large data centers.

3. DESIGN OF THE LBRS SCHEME

In this section, the mathematical model of LBRS was presented, and then, the recursive path selection strategy is specified. Finally, the rationality and performance of the model were analyzed.

3.1 LBRS model

To formalize the LBRS scheme, the specific data was formalized.

$flow_i$: the *i*th flow needs to translate

$path_{j,k}$: the path set for the *flow* j is the order of *j*th concurrent path， k is the *k*th link in the single path from *j*th concurrent path

σ: in theory, the single link is capacity for limited elephant flows, the count is marked as σ

pn: The number of elephant flows carried on a single link. The larger the value of pn, the larger the number of elephant flows that can be accommodated, and the smaller the number of mice flows that can be reserves. On the contrary, it can support more mice flows.

So the remaining bandwidth of the one path is denoted by **PB_rsv**, reserved for the mice flow transmission.

When elephant flow is detected between two hosts, the Equal-cost multi-path routing (ECMP) protocol generates a set of ***j*** number of alternative paths for the *i*th flow, $path_{i,j}$={ $path_{i,0}$, $path_{i,m}$, ..., $path_{i,j-1}$}. These paths are j number of parallel transmission paths, each consisting of k segments links, so each segment of the link can be represented as $path_{j,k}$. From the above definition, it can be seen that each segment of the link on $path_{j,k}$ can carry (between two adjacent switches, or between a host and a directly connected switch, which referring to a 1-hop network). When the remaining bandwidth can reserve enough bandwidth to accommodate elephant flow, as shown in Equation 1, this link can be used as an alternative path.

$$\mathrm{PB\_rsv}_{i,j,k}=1-\frac{pn}{\sigma} \tag{1}$$

So the maximum link bandwidth obtained from the existing k-th path is expressed as Equation 2

$$\mathrm{PB\_rsv}_{i,j}=\min(\mathrm{PB\_rsv}_{i,j,0},\mathrm{PB\_rsv}_{i,j,1},\cdots,\mathrm{PB\_rsv}_{i,j,k\text{-}1}) \tag{2}$$

For the *i*th flow, the maximum available bandwidth reserved by the network is expressed as Equation 3

$$\mathrm{PB\_rsv}_{i}=\min(\mathrm{PB\_rsv}_{i,0},\mathrm{PB\_rsv}_{i,1},\cdots,\mathrm{PB\_rsv}_{i,j}) \tag{3}$$

Only when the remaining bandwidth can meet the requirements shown in Formula 4, can this link be used as an alternative path for ECMP as one of the alternative paths.

$$\mathrm{PB\_rsv}_{i}\geq {}^{1}\!/_{\sigma} \tag{4}$$

Due to the fact that multiple paths on the link and the ECMP protocol choose the minimum link capacity as the actual transmission bandwidth during transmission, the determination of whether the path of the i-th flow can reach the optimal transmission path mainly depends on Equation 5, which can be inferred from Equation s 1 and 4

$$\mathrm{pn}\leq\sigma-1 \tag{5}$$

The remaining bandwidth of the link can be adjusted according to the distribution of flow. If the system is mainly composed of elephant flow, a certain amount of bandwidth can be reserved for the transmission of mice flow. If the mice flow is mainly, the path of mice flow cannot be predicted. If too much reserved bandwidth is left for each path, it causes elephant flow to be unable to find a suitable transmission path. Conversely, mice flow may not achieve the expected performance value. Setting appropriate bandwidth reservation parameters is important for the overall network performance.

In addition, *retrans_count* is defined to represent the number of flow retransmissions. When the number of retransmissions exceeds the value of *retrans_count*, it is handed over to the upper layer of the network for processing, and the retransmission ends. To meet the transmission efficiency of the mice flow, the retransmission time *TCP_time_out* of the flow is set, which is set to 300ms to ensure the delay requirements of the mice flow transmission.

To record the *pn* value of each link, the most intuitive way is to use a matrix to record the *pn* value on each link. Due to the large number of links and the sparsity of the matrix, we use a link forward data structure to record the *pn* value. We use an array head to store nodes and an array w to store the weights of each edge, where the weight is *pn*. The use of a link forward-data structure can effectively remove duplicate edges, and the time complexity can meet the application requirement.

3.2 Recursive Path Selection Algorithm

Once the default path found by the ECMP algorithm does not meet Equation (5), a new path needs to be found to replace the original path. Intuitively speaking, utilizing the hopping of neighboring nodes to quickly find new transmission paths. We abstract the path discovery process as the process of finding concurrent paths, implementing breadth first on each neighboring node to break out of the current default, and then using depth first algorithms to find alternative paths for sending flows.

In order to find available transmission paths, the ECMP protocol's transmission path to flow is represented as a triplet *<AggID1, CoreID, AggID2>* for calculation. We use three methods to replace the original transmission path:

(1) Replace the core switch, and the corresponding AggID1 and AggID2 will be changed accordingly. The new transmission path is:*<AggID1 ', Core_ID', Agg_ID2'>*

(2) Add a new aggregation switch *AggID1* 'to the POD where the source server is located, and the corresponding *CoreID* and AggID2 will change accordingly. The new transmission path is:*<AggID1, Edg_ID, Agg_ID1', Core_ID', Agg_ID2'>*

(3) Add a new aggregation switch *AggID2*' to the POD where the destination server is located, and the corresponding CoreID will change accordingly. The new transmission path is:*<AggID1, Core_ID', Agg_ID2', Edg_ID, Agg_ID2>*。

On these alternative paths set, the available paths through the network probe are used as alternative paths for ECMP. We can periodically monitor the operation status of the data center (5 minutes each time), detect the number of elephant flows carried by each link, the transmission delay of flows, and other data, and hand them over to intelligent decision-making for processing, thereby providing intelligent decision-making for the above key parameters and achieving the best system configuration.

3.3 analyses of LBRS

Rationality analysis: When the value of *pn* increases, it means that each link can accommodate more elephant flows, while the bandwidth that can be reserved for other flows will decrease. If *pn* approaches σ*-1*, theoretically this segment of the link may not allow other flows to pass through, regardless of how many retransmissions are made. When the link bandwidth in a network exceeds 70% of the total bandwidth, performance degradation is very severe. So, *pn* becomes a key factor in flow scheduling. On the one hand, if *pn* is set too small, as the number of network flows increases, it makes it difficult for elephant flow to use the default ECMP protocol to choose a suitable transmission path, resulting in more redirect paths and increasing the cost of network transmission [16].

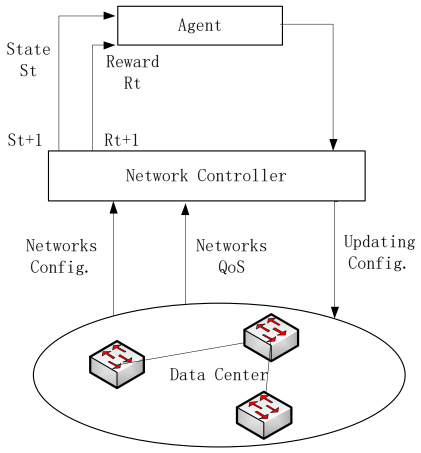
Cost analysis: flow transmission requires a certain amount of link occupancy for a certain period time. The lower the link occupancy time, the more flows can be accommodated in other links in the system. Therefore, the cost of a flow transmission can be expressed as the number of occupied links *hop* multiplied by the time *t* of the occupied links, expressed as $\text{cost} = \sum t * hop$. When the transmission time of each flow is fixed, reducing the number of new path hops in the ECMP protocol can reduce the value of the hop. If all flows can choose the shortest path chosen by ECMP, there is no doubt that the lowest transmission value can be achieved, but the requirement for *pn* to be infinitely large is not consistent with Equation (5). So the value of *pn* is a tradeoff that optimizes the transmission efficiency of both elephant flow and mice flow simultaneously.

4. REINFORCEMENT LEARNING IMPLEMENTATION

This section introduces the specific LBRS and settings of reinforcement learning, and shows the details of scheme initialization in online learning.

4.1 Intelligent Proxy Settings

This section introduces deep Markov processes to train parameter values. Reinforcement learning utilizes continuous interaction between agents and the environment, utilizing actions, rewards, and observation results to gradually update and optimize agents, providing support for intelligent decision-making. The basic idea is to achieve the goal of universal artificial intelligence by maximizing environmental rewards. In this method, we set up agents to change parameter values and optimize the parameters of the network system by monitoring network performance and adopting different rewards (or punishments) for different actions.



From a mathematical perspective, reinforcement learning is modeled as a Markov decision process (MDP). During this process, the agent interacts with the environment at each step. The intelligent agent executes an action, and the environment returns the current immediate reward and the next state. This process continues to form a sequence of states, actions, and rewards. Establish an RL algorithm for decision-making and generate a state action table, which becomes a Q-value table. The configuration policies can be generated based on the updated Q-table.

State space. For the automatic configuration task in this study, a vector state is defined as a possible configuration. The format is as follows:

$$S_i = (pn,\ retrans\_count,\ \sigma) \tag{6}$$

Action set. Three actions are defined to control parameters associated with increase, decrease, and keep. Using the vector Action_i to represent the operation on parameter i, each parameter is a vector composed of three elements, indicating that the action composed of three elements has been adopted (using 1 instead of 0). For example, $Para_i(1, 0, 0)$ represents the increase operation on parameter i, so each state has three actions, marked with 1 and 0 respectively

Immediate reward. The immediate reward received at time t, which reflects the positive benefit of network system throughput, is:

$$R_t = SLA - \text{performance}_t, \tag{7}$$

Where *SLA* represents predefined network performance, such as FCT, and $performance_t$ is the measured value at time *t*. For some SLAs, lower FCT means that agents will bring a positive reward, otherwise they will receive a negative penalty. On the contrary, if throughput is used as an immediate report, the Equation 7 is exactly the opposite.

Q-value learning. Due to the state and action space of the environment, an environment model isn't needed, and the Q-table can be updated based on measurement values, making it suitable for choosing Q-Learning algorithm. The Q-learning algorithm is not about optimizing a strategy based on its knowledge, but creating a new scheme that is represented in the form of a table. The input is the state and action, and the output is the value of each state and action. Using this incremental approach, the Q value of the action A on state S can be calculated once after each instant reward R is updated as equation 8.

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha * [R_{t+1} + \gamma * Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)] \tag{8}$$

where α is the learning rate parameter, which helps to converge to the true Q-value in the presence of noise or random rewards and state transitions, and γ is the discount rate to guarantee the accumulated reward convergence in the continuing scheduling task.

In the reinforcement learning process, the interaction with the external environment means exchanging network status with DCN. The reconstruction process can be a finite MDP, which consists of a set of states and several actions for each state. At each step of state transition, the agent could get a reward which can be calculated by Equation 9.

$$Q(S_t, A_t) = E[(G\ (S, A)\ æ\ S = st, A = at)] \tag{9}$$

The purpose of the agent is to establish a scheme $\pi: S \rightarrow A$ to maximize the accumulated rewards based on iterative trial and error interaction [17].

4.2 the parameter training

To learn the initial scheme, we need to collect training data for subsequent RL learning. The key issue of the algorithm is to select representative states for approximation. In the production environment, the parameter scale of system configuration is very large. The method proposed in this question, while considering the goal of simplifying the system, tries to use a simple method to optimize key parameters as much as possible. Therefore, three parameters (***pn, retrans_count, σ***) were selected. With the optimization of system performance, more key parameters that affect performance will be discovered and added to this method. Overall, the key parameters of the DCN include the following: *pn*, *retrans_count*, and *σ*. The range and default value of parameters are shown in Table 1, in which the value refers to experience [15].

TABLE1. KEY PARAMETERS

Parameter	Range	Default
pn	[1,10]	3
retrans_count	[3,15]	8
σ	[3,10]	7

Algorithm 1 shows the LBRS training algorithm. LBRS never stops until being stopped. At each interaction with DCN performance, the current state is marked, so the actual immediate reward can be obtained. The greedy policy is used to select the next action by the output function. The network performance is calculated during the last interval. The Q function is updated as in Algorithm 2.

Algorithm 1 The LBRS training algorithm

```
1: Configuration DCN with three parameters (pn:3, retrans_count:8, σ:7) , Measure the FCT as initial $s_t$
2: **Initialize** $\alpha$ =0.15, $\gamma$=0.9 in agent, t:0
3: **repeat**
4:      $s_t \leftarrow current\_networks\_state$ ()
5:      $reconfigure\_DCN(a_t)$
6:      $r_{t+1} \leftarrow calculate\ reward()$
7:      $a_{t+1} \leftarrow get\ next\ action(s_t, Q_{appx})$
8:      $configure \leftarrow identify\ configure()$
9:      $R_{model} \leftarrow select\ model(configure)$
10:     $update\ R_{model}(s_t, a_t, r_{t+1}, R_{model})$
11:     $update\ Q_{appx}(R_{model}, Q_{appx})$
12:     $t \leftarrow t + 1$
13: **until** LBRS is terminated
```

Algorithm 2 Q function update algorithm

```
1: Initialize Q() to the function. $\alpha$=0.15, $\gamma$=0.9
2: repeat
3:    s ← 0
4:    for n iterations do
5:        ($s_t$ , $a_t$ ,$r_t$) = generate ( )
6:          target= $r_t + \gamma * Q(s_{t+1}, a_{t+1})$
7:          error = target − $Q(s_t, a_t)$
8:          s = $\gamma$* s +$\alpha$* error * error
9:          update target network as $Q(s_t, a_t)$
10:   end for
11: until converge(s)
```

After obtaining all the training data, we ran an algorithm similar to [17] to learn the initial Q-value table. In the implementation process, we set α=0.15, γ=0.9 as the parameter values during Q function updating.

5. PERFORMANCE EVALUATION

5.1 Simulation Environment

The simulator Mininet is employed to test the performance of LBRS. We need to understand how well LBRS performs against other schemes such as P4-MLFQ [6], PFLBS [9], and DRL-Plink [13]. The Mininet version was ryu3.4.3, and the programming language was Python 3.4. The experimental machine configuration is Intel i7-14700kf, 16.0GB RAM, CentOS Linux release 7.6.1810 (Core). The experiment uses the iperf tools to generate simulated elephant flow between hosts, and uses the ping command to simulate the generation of mice flow. The topology of the network adopts the standard Fat Tree structure.

The data flow adopts two communication modes: (1) Random: randomly generating data flow between randomly numbered hosts. (2) Stag (pEdge, pPod): pEdge is the proportion of traffic between hosts within the same Edge Switch, pPod is the proportion of traffic between hosts within the same Pod that are not in the same Edge Switch, and the proportion of host traffic between different Pods is 1-pEdge pPod.

5.2 Metrics

(1) Response time for the first packet of dynamic flow

We simulated three types of flows and changed the composition of the data flow at three stages, with the basic ratio of elephant flow to mice flow, and the quantity ratio being 3:1, 7:1, and 10:1 at the 10ms, 20 ms, and 30ms, respectively. The response time of first packet is shown in Figure 2. when the proportion of the flow changes, the iterative algorithm always tends to converge in response time within a certain number of times, and the performance obtained is better than the default scheduling method. From this perspective, this method can adaptively adapt to dynamically changing flows and the performance can quickly converge than others.

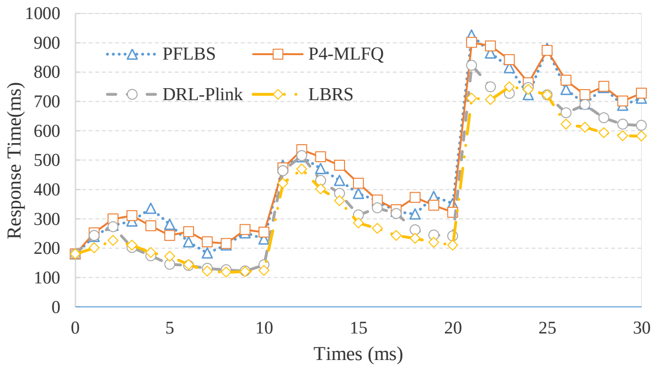


Fig. 2. Impact of Iteration Times on Response Time

(2) Average FCT

Figure 3 shows the overall average FCT for the different traffic at different link loads. The overall average FCT with LBRS is up to ~ 21.6% lower compared to P4-MLFQ in Strg(0.1,0.3) traffic, and ~ 23.6% lower compared to PFLBS in Strg(0.3,0.3). LBRS reduces the overall average FCT in these traffic by up to ~ 17.2%, ~

16.6% and ~ 9.9%, when compared to PFLBS, P4-MLFQ, and DRL-Plink, respectively. The results show LBRS can reduce the transmission time efficiency.

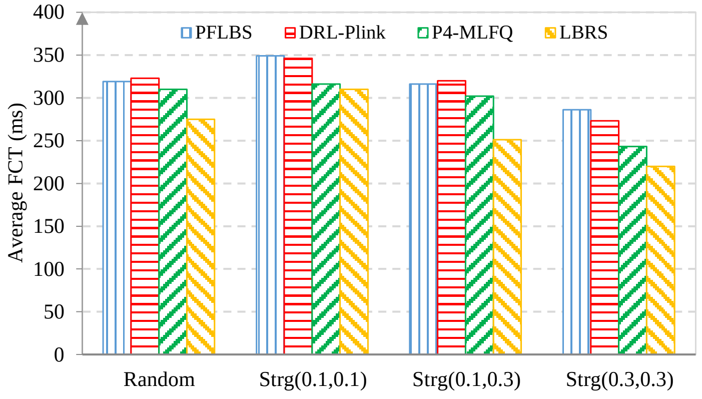


Fig. 3. Impact of Scheduling Strategies on Average FCT

(2) Total throughput

Figure 4 shows the total throughput of whole networks for the different loads. In general, LBRS offers the best performance. The total throughput with LBRS is up to ~ 27.0% higher compared to PFLBS in Strg(0.3,0.3) traffic and ~ 23.6% lower compared to PFLBS in Strg(0.1,0.3). LBRS increases the overall average FCT in these traffic by up to ~ 25.3%, ~ 21.6%, and ~ 6.7% when compared to PFLBS, P4-MLFQ, and DRL-Plink, respectively, which indicates that while restricting the forwarding of elephant flows. The result shows it can also improve the forwarding efficiency of elephant flows and enhance the network throughput.

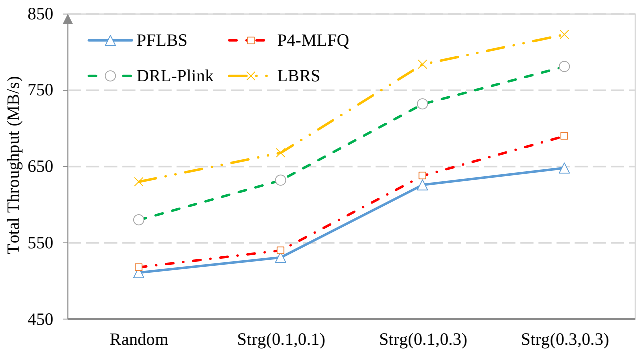


Fig. 4. Impact of Scheduling Strategies on Total Throughput

6. CONCLUSIONS

In this paper, a link-state-based bandwidth-reservation scheme is proposed mixed-flow scheduling in DCN. Theoretical analyses and experimental results show that LBRS can adapt to dynamic changes in the data flow distribution, and automatically configure appropriate parameters. In addition, intelligent technology can increase the efficiency of flow scheduling and network management, and provide reference and convenience for intelligent network management.

In this work, the fewer parameters for flow scheduling decisions may limit the performance improvement, which means ignoring the potential detail. Automatic parameter choices are not considered in this work.

In the future, fine-grained flow scheduling will become our new direction, and more parameters in intelligent decision-making may make flow scheduling more refined. Moreover, combining intelligent routing and flow scheduling could improve in better network performance.

ACKNOWLEDGMENTS

This work has been supported by the project funded by the Hubei Provincial Department of Education (No. B2018203) and the National Natural Science Foundation of China (No. 62272350); in part by the Teaching Research Project of Hubei University of Education (X2019011).

REFERENCES

1. S. M. Shetty, S. Shetty, "Analysis of load balancing in cloud data centers," *Journal of Ambient Intelligence and Humanized Computing*, Vol. 15, 2024, pp. 973–981, doi:10.1007/s12652-018-1106-7
2. A. Cornacchia, A. Bianco, P. Giaccone, and G. Sviridov, "A “Big-Spine” Abstraction: Flow Prioritization With Spatial Diversity in The Data Center Network," *2024 IEEE 25th International Conference on High Performance Switching and Routing (HPSR)*, 2024, pp. 43-48, doi: 10.1109/HPSR62440.2024.10635939.
3. M. Yenugula, S. Sahoo, and S. Goswami. "Cloud computing for sustainable development: An analysis of environmental, economic and social benefits." *Journal of future sustainability*, vol. 4, no.1, 2024, pp. 59-66. doi:10.5267/j.jfs.2024.1.005
4. Z. Li, J. Huang, S. Wang, and J. Wang, "Achieving Low Latency for Multipath Transmission in RDMA Based Data Center Network," *IEEE Transactions on Cloud Computing*, vol. 12, no. 1, 2024, pp. 337-346, doi: 10.1109/TCC.2024.3365075.
5. L. Sun et al., "BiSwift: Bandwidth Orchestrator for Multi-Stream Video Analytics on Edge," *IEEE INFOCOM 2024*, 2024, pp. 1181-1190, doi: 10.1109/INFOCOM52122.2024.10621392.
6. M. Iqbal and C. Chen, "P4-MLFQ: A P4 implementation of Multi-level Feedback Queue Scheduling Using A Coarse-Grained Timer for Data Center Networks," *2023 IEEE 12th International Conference on Cloud Networking (CloudNet)*, 2023, pp. 120-125, doi: 10.1109/CloudNet59005.2023.10490044.
7. B. Huang and S. Dong, "An Enhanced Scheduling Framework for Elephant Flows in SDN-Based Data Center Networks," *2020 IEEE Symposium on Computers and Communications (ISCC)*, 2020, pp. 1-7, doi: 10.1109/ISCC50000.2020.9219688.

8. J. Geng, "DCI-NACC: flow scheduling and congestion control based on programmable data plane in high-performance data center networks," *The International Journal of Advanced Manufacturing Technology*, Vol. 122, 2022, pp. 51-63, doi:10.1007/s00170-021-08459-4.
9. Z. Liu, et al. "An Efficient Flow Detection and Scheduling Method in Data Center Networks," *The 2nd International Conference on Computing and Data Science (CONF-CDS)*, 2021, pp. 1-7, doi:10.1145/3448734.3450819
10. X. Du, et al. "R-AQM: Reverse ACK Active Queue Management in Multitenant Data Centers, " *IEEE/ACM Transactions on Networking*, Vol. 31, no. 2, 2022, pp. 526–541. doi:10.1109/TNET.2022.3197973
11. Q. Xu, et al. "ReQ-tank: Fine-grained Distributed Ma-chine Learning Flow Scheduling Approach," *IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS),* 2023, pp. 146-152, doi: 10.1109/ICPADS60453.2023.00030.
12. W. Liu, et al. "DRL-R: Deep reinforcement learning approach for intelligent routing in software-defined data-center networks." *Journal of Network and Computer Applications,* Vol, 177: 102865, 2021, doi:10.1016/j.jnca.2020.102865.
13. W. Liu, J. Lu, J. Cai, Y. Zhu, S. Ling and Q. Chen, "DRL-PLink: Deep Reinforcement Learning With Private Link Approach for Mix-Flow Scheduling in Software-Defined Data-Center Networks," *IEEE Transactions on Network and Service Management,* Vol. 19, no. 2, 2022, pp. 1049-1064, doi: 10.1109/TNSM.2021.3128267.
14. T. Chen, et al. "Reinforcement Learning for Datacenter Congestion Control," *ACM SIGMETRICS Performance Evaluation Review*, Vol. 49, no. 2, 2022, pp. 43–46, doi:10.1145/3512798.3512815.
15. W. Liu, et al. "Fine-grained flow classification using deep learning for software defined data center networks," *Journal of Network and Computer Applications.* Vol, 168, 2020, pp. 102766, doi: 10.1016/j.jnca.2020.102766.
16. A. Satpathy, et al, "ReMatch: An efficient virtual data center re-matching strategy based on matching theory," *IEEE Transactions on Services Computing*. Vol, 16.2, 2022, pp. 1373-1386. doi:10.1109/TSC.2022.3183259
17. X. Bu, J. Rao, C. Xu, "A reinforcement learning approach to online web systems auto-configuration," *In 29th IEEE International Conference on Distributed Computing Systems,* 2009, pp. 2-11. doi: 10.1109/ICDCS.2009.76

Xing-Yan Zhang received a Ph.D. degree in Computer Science from Huazhong University of Science and Technology in 2016. He is currently a lecturer at Hubei University of Education. His research interests include data center networks and artificial intelligence.