Topology configuration scheme for accelerating Coflow in Hyper-FleX-LION

HAO YANG[1,2] AND ZUQING ZHU[1,*]

[1] School of Information Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, P. R. China
[2] Department of Information Engineering, Southwest University of Science and Technology, Mianyang, Sichuan 621010, China
[*] Corresponding author: zqzhu@ieee.org

Compiled August 17, 2022

In order to address the challenges that data center networks (DCNs) are currently facing, people have tried to introduce optical circuit switching (OCS) in DCNs and proposed a number of architectures, for improving DCNs' performance on port capacity, energy efficiency, data transfer latency, *etc*. This led to a few all-optical interconnects (AOIs), among which Hyper-FleX-LION possesses the reconfigurability that can support various traffic patterns efficiently. In this work, we study the topology management of AOIs in Hyper-FleX-LION for efficiently scheduling Coflows in them, such that the average Coflow completion time (CCT) can be minimized. We first propose time-efficient algorithms to address the provisioning of a single Coflow and then extend them for multi-Coflow scenarios. Simulation results demonstrate that AOIs in Hyper-FleX-LION can accelerate Coflows better than the traditional AOIs based on optical cross-connects (OXCs), and achieve an acceleration of up to 5.6× under the assumed situations. Our simulations also verify that for Coflow scheduling in Hyper-FleX-LION, our proposal can reduce the average CCT of Coflows and outperform benchmarks significantly.

http://dx.doi.org/10.1364/ao.XX.XXXXXX

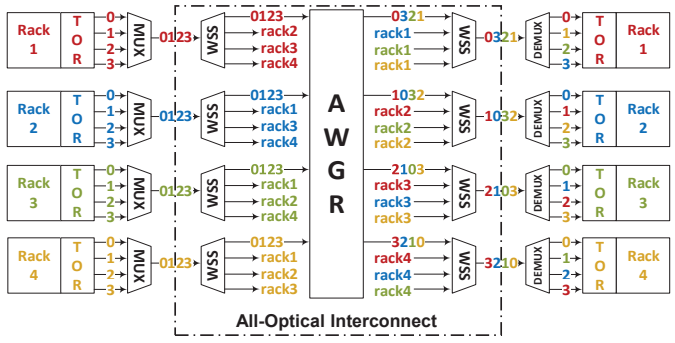
1. INTRODUCTION

Recently, the explosive development of cloud services has motivated intensive research and development on data centers (DCs) [1]. Therefore, the infrastructure of DC networks is undergoing revolutionary changes to address the mismatch between quality-of-service (QoS) demands of network services and capacity/latency/cost of the interconnects among computing and storage platforms in DCs. Specifically, the traditional interconnects that are solely based on electrical packet switching (EPS) are facing many challenges such as limited port capacity, ever-increasing energy consumption, and prolonged latency due to packet processing and queuing in EPS switches [2, 3]. Meanwhile, optical circuit switching (OCS) is a promising alternative, since it provides larger port capacity, higher energy efficiency, and shorter data transfer latency than EPS [4–8]. To this end, all-optical interconnects (AOIs) based on OCS have been introduced to DC networks (DCNs) [9–14] to relieve the EPS-induced bottlenecks there, especially for accelerating the tasks whose completion relies on the delivery of huge amounts of elephant flows (*e.g.*, large-scale distributed machine learning (DML) [15]).

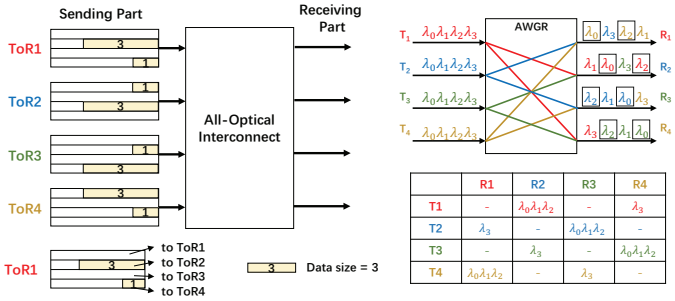
Compared with EPS, OCS improves the performance of interconnects, but it is also less adaptive due to the relatively large switching granularity at the port or wavelength level [4]. Although this drawback can be partially addressed by leveraging virtualization techniques to groom the traffic of similar network services with virtual networks [16–18], the reconfigurability of an AOI cannot be fully explored without a sophisticated algorithm to manage its topology and schedule traffic through it adaptively [19]. However, for each type of AOIs, such an algorithm is unique. Hence, the topic is still under-explored, especially for new types of AOIs.

More importantly, an emerging network service such as DML can involve multiple correlated data flows, whose overall performance determines its QoS [15]. Therefore, the topology management and traffic scheduling algorithm that treats the data flows independently might not work well in such a situation. The correlated data flows of a network service can be modeled as a *Coflow*, whose completion time (*i.e.*, Coflow completion time (CCT)) is the time when the data transfers of all the flows have been accomplished [20]. To minimize CCT, previous studies have proposed various algorithms to schedule Coflows in EPS-based DCNs [21–23], and people have also addressed the problem of topology management and Coflow scheduling in a few specific AOIs [24–27].

Lately, Yoo *et al.* have proposed and fabricated an integratable OCS device, namely, FleX-LIONS [28, 29], with which large-scale reconfigurable AOIs (*i.e.*, Hyper-FleX-LIONs) can be built to support various traffic patterns efficiently [13]. Fig. 1(a) shows an illustrative example on an AOI that is in Hyper-FleX-LION and connects four top-of-rack switches (ToRs). The AOI has an ar-



(a) Network architecture



	R1	R2	R3	R4
T1	-	$\lambda_0\lambda_1\lambda_2$	-	λ_3
T2	λ_3	-	$\lambda_0\lambda_1\lambda_2$	-
T3	-	λ_3	-	$\lambda_0\lambda_1\lambda_2$
T4	$\lambda_0\lambda_1\lambda_2$	-	λ_3	-

(b) Topology configuration for scheduling a Coflow

Fig. 1. Coflow scheduling in an AOI that is in Hyper-FleX-LION and connects four ToRs, MUX/DEMUX: wavelength multiplexer/demultiplexer, WSS: wavelength selective switch, AWGR: arrayed waveguide grating router.

rayed waveguide grating router (AWGR) working as the core, and the sending/receiving parts of each ToR are respectively plotted at its left/right sides. In general, for a Hyper-FleX-LION that connects N ToRs (*i.e.*, $N = 4$ in Fig. 1(a)), each ToR equips N transceivers (TRXs), each of which transmits in a unique wavelength-division multiplexing (WDM) channel and can receive optical signal in any of the N WDM channels. In Fig. 1(a), we label the WDM channels from the TRXs with numbers and color each number to mark the source ToR of each optical connection.

In the sending part, all the outputs of a ToR are first multiplexed by a WDM multiplexer (MUX) and then sent to a $1\times N$ wavelength selective switch (WSS). The WSS connects one of its outputs to the AWGR, while its remaining $N-1$ outputs directly go to the $N-1$ WSS' that locate in the receiving parts of other ToRs, respectively. For instance, the second output of the WSS in the sending part of *ToR* 1 connects to the second input of the WSS in the receiving part of *ToR* 2. Then, in the receiving part of each ToR, the optical signals, which are from the WSS' in the transmitting parts of other ToRs and the AWGR, are distributed to the ToR's TRXs by a WSS and a WDM demultiplexer (DEMUX).Note that, together with the WSS, the AWGR achieves effective wavelength switching with a small port count. Specifically, it switches optical signals from an input to different outputs according to their WDM channels. For instance, in Fig. 1(a), *ToR* 1 has four TRXs operating in different WDM channels, respectively, and after the WSS in the sending part, the WDM channels are all forwarded to the first input of the AWGR, which respectively switches them to the four outputs according to their wavelengths (*i.e.*, *Channel* 0 to the first output, *Channel* 1 to the second output, and so on so forth). The AWGR is a necessary component in the integrated Silicon photonic version of Hyper-FleX-LION [28]. However, it can be replaced with a full-mesh interconnection in the Hyper-FleX-LION in Fig. 1(a) (*i.e.*, an implementation of the integrated version with bulk components), because the WSS' can also realize wavelength switching.

Such a structure brings high flexibility to the traffic scheduling in AOIs. Specifically, we can adjust the WSS' in the sending/receiving parts of an AOI in Hyper-FleX-LION and utilize the wavelength switching in its AWGR to realize various AOI topologies for adaptive traffic scheduling, especially to support the inter-rack communications other than unicast (*e.g.*, multicast and incast). Moreover, although an AOI in Hyper-FleX-LION uses the OCS facilitated by the WSS' and AWGR, it does not rule out the possibility of packet switching for highly dynamic flows in DCs. This is because the ToR switches are packet-based, *i.e.*, even without changing the topology of the AOI, the ToR switches can switch flows to different TRXs at the packet level. For example, in [30], we experimentally demonstrated that an AOI in Hyper-FleX-LION achieved better acceleration of DML jobs than a hybrid optical/electrical interconnect (HOEI) that was architected with packet switches and an optical cross-connect (OXC).

Fig. 1(b) explains how to leverage the flexibility of Hyper-FleX-LION to schedule a Coflow. Here, the left subplot shows the configuration of the Coflow, where each of the destination ToRs of a ToR is plotted as a row and the block in each row represents the amount of data transfer to the destination ToR for the Coflow. For instance, at *ToR* 1, the Coflow needs to transmit 3 units and 1 unit of data to *ToRs* 2 and 4, respectively. Then, we can configure the Hyper-FleX-LION as that in the right subplot to minimize the CCT of the Coflow, where the WDM channels without boxes are switched by the AWGR and those marked with boxes go directly between WSS' in the sending/receiving parts. For example, λ_0 and λ_2 from *ToR* 1 are sent to *ToR* 2 directly without going through the AWGR. Hence, at *ToR* 1, the capacity bandwidth to *ToR* 2 is three times of that to *ToR* 4, which ensures that their data transfers can be finished at the same time to shorten the Coflow's CCT.

To the best of our knowledge, the topology management and Coflow scheduling algorithm that is specifically designed for the AOIs in Hyper-FleX-LION has not been addressed in the literature, even though Hyper-FleX-LION's ability on accelerating Coflows can be easily observed in the example in Fig. 1(b). Meanwhile, we can also see that due to the unique flexibility of Hyper-FleX-LION, the algorithms that were designed for scheduling Coflows in other types of AOIs [24–26] are not applicable. This motivates us to study how to configure the topology of an AOI in Hyper-FleX-LION and route the traffic of Coflows accordingly to minimize CCT.

Specifically, we consider the algorithm design to schedule one and multiple Coflows in an AOI in Hyper-FleX-LION, where the network model is built with realistic system parameters, *e.g.*, reconfiguration latency of Hyper-FleX-LION. We propose time-efficient dynamic topology management and Coflow scheduling algorithms to address the two scenarios, for minimizing CCT as well as maximizing the port usage in Hyper-FleX-LION. Extensive simulations suggest that the AOIs in Hyper-FleX-LION can accelerate Coflows better than the traditional AOIs that are based OXCs, and our proposed algorithms can effectively reduce CCT in both the single- and multi-Coflow scenarios.

The rest of the paper is organized as follows. We first briefly survey the related work in Section 2. Section 3 provides the

problem descriptions for the singe- and multi-Coflow scenarios. In Section 4, we discuss the algorithm design, and the numerical simulations are presented in Section 5. Finally, Section 6 summarizes the paper.

2. RELATED WORK

Considering the limitations of the model that treats the data flows of a network service independently, people defined the concept of Coflow in [20], which quickly gained intensive research interests. As the primary and most-used inter-rack architecture for DCNs, EPS-based interconnects have been extensively studied for Coflow scheduling. Varys [21] focused on sorting out the correlations among Coflows and proposed a heuristic to reduce the average CCT. The study in [23] formulated an integer linear programming (ILP) model to plan the provisioning of a known set of Coflows for minimizing the total weighted CCT, and it also designed approximation algorithms to reduce the time complexity of problem solving. Zhang *et al.* [31] tried to locate the critical data transfer in a Coflow, and developed a bandwidth algorithm to avoid bandwidth contention. The authors of [32] performed bandwidth-constrained routing for each data transfer in a Coflow to reduce the chance of bandwidth competition and thus minimize the average CCT. CODA [33] leveraged machine learning to accurately identify the traffic matrices of Coflows in a DCN, for optimizing their scheduling. However, as the aforementioned studies were all based on EPS-based interconnects, their proposals cannot be applied to solve the problem of topology management and Coflow scheduling in AOIs.

With the trend of adding OCS in DCNs, both HOEIs [9, 10] and AOIs [11–13] have been proposed and demonstrated in the past decade. Meanwhile, a few studies have considered topology management and Coflow scheduling in these architectures, with the assumption that the OCS is based on OXC(s), which can only provide one-to-one connectivity and a limited number of topology variations [34]. For scheduling a single Coflow, the study in [24] formulated an ILP model to solve the topology management problem for minimizing the CCT, and designed a polynomial-time approximation algorithm. Tan *et al.* [25] leveraged the Birkhoff-von Neumann decomposition to match the traffic matrix of each Coflow to a feasible AOI topology, for reducing not only the CCT but also the number of OXC reconfigurations. The authors of [26] tried to optimize the placement of Coflow-related jobs, such that Coflows generated by the jobs can make the best utilization of the OCS bandwidth in an HOEI. Except for the OXC-based architectures considered above, there are other AOI architectures that can offer better flexibility (*e.g.*, in [12, 19]), but Coflow scheduling has not been studied in them. As for Hyper-FleX-LION, the existing investigations only addressed the chip design and fabrication [28, 29], architectural optimization [13], and proof-of-concept applications [30], but Coflow scheduling was not considered either. In all, to the best of our knowledge, the topology management and Coflow scheduling in Hyper-FleX-LION have not been studied in the literature, which justifies the novelty and contribution of this work.

3. PROBLEM DESCRIPTION

In this section, we first explain the network model of topology management and Coflow scheduling in Hyper-FleX-LION, and then describe the optimizations that we would like to tackle with algorithm design.

A. Network Model

We consider an AOI in Hyper-FleX-LION that connects N ToRs and operates with the principle shown in Fig. 1(a). In the AOI, each ToR equips N TRXs, which transmit in N adjacent WDM channels (*i.e.*, $\{\lambda_i, i \in [1, N]\}$). We denote the bandwidth capacity of each TRX as B. Each Coflow can be defined as c_k, where k is its unique index, and the Coflow has two key parameters, which are its arrival time t_a^k and the data transfer matrix (DTM) $\hat{\mathbf{D}}_k$. The DTM $\hat{\mathbf{D}}_k$ is an $N \times N$ matrix, each of whose element $\hat{d}_{i,j}^k$ represents the size of the data that a Coflow c_k needs to transfer from *ToR* i to *ToR* j ($i, j \in [1, N]$). For the sake of convenience, we normalize each element in DTM $\hat{\mathbf{D}}_k$ with the bandwidth capacity of a TRX (*i.e.*, B), and get a normalized DTM $\mathbf{D}_k$ for each Coflow c_k. Specifically, each element $d_{i,j}^k \in \mathbf{D}_k$ is obtained as

$$d_{i,j}^k = \lceil \frac{\hat{d}_{i,j}^k}{B} \rceil, \quad \forall i, j \in [1, N]. \tag{1}$$

We assume that the time when $d_{i,j}^k$ has been transferred is $t_{i,j}^k$. Then, the CCT of c_k is $\tau^k = \max_{\forall i,j \in [1,N]} \left(t_{i,j}^k - t_a^k \right)$. Hence, to accelerate a Coflow c_k, we need to minimize τ^k. Here, as the focus of this work is algorithm design, we follow the existing studies in this area [24–26] to assume that the CCT can be accurately obtained with calculations. In real-world network systems, the completion time of a data flow can also be got accurately with a few methods, such as time-stamping [30, 35] and monitoring the relevant packet queues on ToR switches [36]. Meanwhile, as multi-hop forwarding will make traffic be processed by more ToR switches and thus increase the average CCT of Coflows, we do not consider it in this work. Specifically, we assume that for transferring the data of Coflows, each ToR switch only receives the traffic whose destination is its rack.

For simplicity, we denote a Hyper-FleX-LION that connects N ToRs as N-Hyper-FleX-LION in the following discussions. As the configuration of the N-Hyper-FleX-LION is the key for Coflow acceleration, we define the network model for it based on a binary variable $f_{i,j}^l$, which equals 1 if *ToR* i uses the l-th TRX on it to set up an optical connection to *ToR* j. Therefore, to avoid wavelength contention at the receiving part of N-Hyper-FleX-LION, its configuration should follow the following constraints.

$$\begin{aligned} &\sum_{j \in [1,N]} f_{i,j}^l \leq 1, \quad \forall i \in [1, N], \forall l \in [1, N], \\ &\sum_{i \in [1,N]} f_{i,j}^l \leq 1, \quad \forall j \in [1, N], \forall l \in [1, N], \end{aligned} \tag{2}$$

which ensures that N-Hyper-FleX-LION can switch the optical signal from each TRX on a ToR to one ToR at most, and all the optical signals switched to a ToR have to use different WDM channels.

Note that, to reconfigure N-Hyper-FleX-LION, we need to change the status of the WSS' in it, which also introduces latency, and the reconfiguration latency of a WSS is usually in the range of a few hundred milliseconds to seconds [37], and all the WSS' in N-Hyper-FleX-LION can be reconfigured in parallel. Hence, we define the latency of reconfiguring N-Hyper-FleX-LION once as ϵ seconds, and to focus the analysis in this work more on the architectural differences between AOIs and Coflow scheduling algorithms, we consider the worst case scenario and assume that the reconfiguration of N-Hyper-FleX-LION follows the *all-stop* model [38], *i.e.*, each reconfiguration increases the CCT of each active Coflow by ϵ. Meanwhile, we need to check the current network status from time to time to reconfigure the AOI

for Coflow scheduling, with an interval that is longer than the reconfiguration latency ϵ. Otherwise, AOI reconfigurations will be triggered too frequently before they can actually be finished. Moreover, we should set the interval properly to adjust the tradeoff between the performance and complexity of Coflow scheduling. To adapt to highly dynamic network environments, the interval will be chosen empirically in real-world operations.

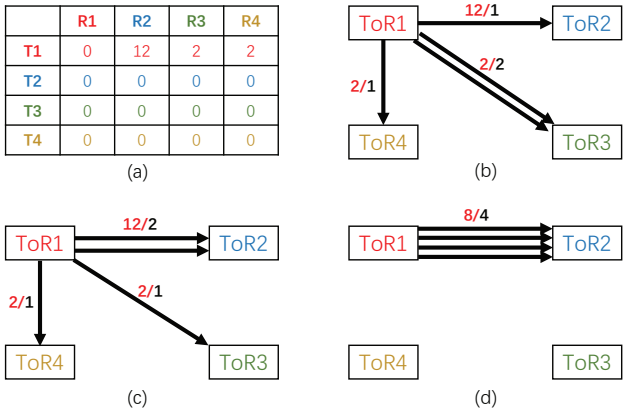


Fig. 2. Example on topology management for Coflow scheduling.

B. Single-Coflow Scenario

Note that, an AOI in Hyper-FleX-LION can adjust its topology freely with wavelength switching, and thus its switching granularity is smaller than some well-known commercial OXCs (*e.g.*, Polatis Series 6000 OXC), which can only switch at the port level [25]. Therefore, compared with the AOIs built with such OXCs, AOIs in Hyper-FleX-LION are more flexible and can support the inter-rack communications other than unicast better. Specifically, it can realize uneven connectivity among the N ToR more easily, which is essential for Coflow acceleration. The single Coflow acceleration scenario only tries to schedule the data transfers of one Coflow c to minimize its CCT τ. For instance, we assume that a Coflow c needs to be scheduled in the 4-Hyper-FleX-LION shown in Fig. 1(a). The Coflow's DTM $\mathbf{D}$ contains four non-zero elements, as $d_{1,2} = d_{2,3} = d_{3,4} = d_{4,1} = 3$. If we denote the bandwidth allocation from *ToR* i to *ToR* j as $b_{i,j}$ (having been normalized with B already), the completion time of $d_{i,j}$ is $\frac{d_{i,j}}{b_{i,j}}$. Therefore, if we configure the AOI as that in the right subplot of Fig. 1(b), the CCT of c is $\tau = \frac{3}{3} = 1$.

Fig. 2 shows a straightforward example to explain why topology management is important for Coflow scheduling. Here, we consider a Coflow that needs to transfer data from *ToR* 1 to other ToRs in an AOI in 4-Hyper-FleX-LION, with the DTM in Fig. 2(a), *i.e.*, $d_{1,2} = 12$, $d_{1,3} = 2$, and $d_{1,4} = 2$. Fig. 2(b) illustrates a feasible configuration of the AOI, where each black arrow denotes the optical connection from a TRX and the number on it shows the calculation of the completion time for the data transfer using it. For example, the completion time of $d_{1,3}$ is $\frac{2}{2} = 1$. Then, the CCT of the Coflow using the configuration in Fig. 2(b) is 12. Similarly, the CCT with the configuration in Fig. 2(c) is 6, representing a better solution.

Note that, the CCTs in Figs. 2(b) and 2(c) are obtained by only planning the AOI's configuration once, and we can further reduce the Coflow's CCT by leveraging the reconfigurability of Hyper-FleX-LION. Specifically, with the configuration in Fig. 2(c), we can find that the TRXs for $d_{1,3}$ and $d_{1,4}$ become idle after $t = 2$, when the data transfers using them have been done. Hence, we can reconfigure the AOI to use the topology in Fig. 2(d) after $t = 2$. Then, the completion time of $d_{1,2} = 12$ becomes $\frac{4}{2} + \epsilon + \frac{8}{4} = 4 + \epsilon$, where ϵ is the reconfiguration latency. Therefore, the Coflow's CCT is further reduced.

C. Multi-Coflow Scenario

In a dynamic DCN environment, multiple Coflows can be generated on-the-fly. Therefore, we need to manage the topology of Hyper-FleX-LION and schedule Coflows in it dynamically.For this scenario, we aim to minimize the average CCT of all the Coflows, following the optimization models in [25, 26].

$$\text{Minimize} \quad \frac{1}{K} \sum_{k \in [1,K]} \tau^k, \tag{3}$$

where K is the total number of Coflows.

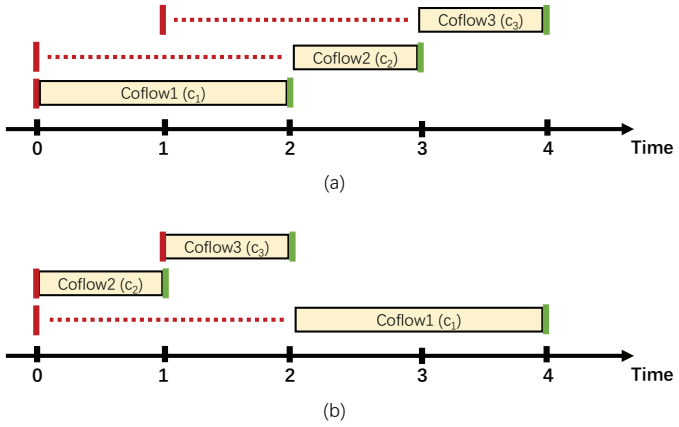


Fig. 3. Example on the effect of Coflow serving order on CCT.

When there are multiple Coflows, the order that we use to serve them can also significantly affect their CCTs, as explained in Fig. 3. Here, we need to schedule three Coflows whose arrival time and completion time are marked with red and green vertical lines, respectively. It can be seen that for the scheduling scheme in Fig. 3(a), the three Coflows need to count $2 + 2 = 4$ units of waiting time in their CCTs, while the one in Fig. 3(b) only lets c_1 wait for 2 time units. Then, the serving order in Fig. 3(b) is apparently better.

4. ALGORITHM DESIGN

In this section, we propose time-efficient dynamic topology management and Coflow scheduling algorithms to address the two scenarios defined in the previous section.

A. Algorithm for Single-Coflow Scenario

We first propose an algorithm to solve the topology management and Coflow scheduling in Hyper-FleX-LION for the single-Coflow scenario. The algorithm treats an AOI in N-Hyper-FleX-LION as a directed graph, where each node v_i corresponds to a ToR ($i \in [1, N]$) and each directed edge represents an optical connection from a TRX. Then, the topology management is just to plan the edges such that a Coflow's CCT can be minimized. The rationale behind our design is to find the *necessary CCT* $\tau_{\min}$ for the Coflow and use it to guide the planning of the edges. Specifically, $\tau_{\min}$ refers to the shortest time that is required for

completing the Coflow, and it is determined by the largest element(s) in its DTM. Therefore, for any data transfers in the DTM, if the planned edges can already make its completion time not longer than $\tau_{\min}$, we do not need to allocate more edges to it.

Algorithm 1 describes our proposed procedure for calculating the configuration of an AOI in Hyper-FleX-LION based on the remaining DTM of a Coflow $\tilde{\mathbf{D}}$, each element of which is the amount of remaining data that still needs to be transferred. *Lines* 1-2 are for the initialization, where insert a node v_i in the directed graph $G(V,E)$ to represent each ToR i, N_i^+/N_i^- denote the numbers of available input/output ports on *ToR* i, respectively, and $b_{i,j}$ is used to record the number of allocated edges from *ToR* i to *ToR* j. Then, we calculate the total amounts of data that each ToR needs to send and receive, and calculate the necessary CCT $\tau_{\min}$ based on them (*Lines* 3-5). *Lines* 6-10 allocate edges in $G(V,E)$ based on each nonzero element in $\tilde{\mathbf{D}}$. Next, we check whether each nonzero element $\tilde{d}_{i,j} \in \tilde{\mathbf{D}}$ has a feasible return path for the communication in the reverse direction, and will add a direct edge (v_j, v_i) if the route does not exist (*Lines* 11-13). This operation is needed because for each data transfer $\tilde{d}_{i,j} \in \tilde{\mathbf{D}}$, a connection in the reverse direction is still required for supporting the data transfer protocol (*e.g.*, carrying the control signals of transmission control protocol (TCP)), even though the actual data amount that is transmitted on the connection can be very small.

Lines 14-24 try to assign more edges to accelerate each nonzero data transfer $\tilde{d}_{i,j} \in \tilde{\mathbf{D}}$. Here, we use $\tau_{i,j}$ to record the current completion time of $\tilde{d}_{i,j}$ (*Line* 16). If it is already not longer than $\tau_{\min}$, we do not need to assign more edges to it (*Lines* 17-18). Otherwise, we assign more edges to it greedily until we have $\tau_{i,j} \leq \tau_{\min}$ or the available ports on *ToRs* i and j have been used up (*Lines* 19-24). Finally, *Line* 25 assigns WDM channels to the allocated edges in $G(V,E)$ under the constraints in Eq. 2, which can be solved with an existing polynomial-time coloring algorithm [39].

With *Algorithm* 1, we can obtain the configuration of an AOI for a specific remaining DTM. Then, we design *Algorithm* 2 for the topology management of the single-Coflow scenario, which leverages reconfigurations to accelerate a Coflow. *Line* 1 uses *Algorithm* 1 to obtain the initial serving scheme of the Coflow based on its DTM $\mathbf{D}$ and provision it accordingly. Then, the while-loop that covers *Lines* 2-11 constantly checks to see whether any data transfer in $\mathbf{D}$ (*i.e.*, $d_{i,j}$) has been accomplished. If yes, we determine whether and how to reconfigure the AOI with *Lines* 3-11. Specifically, the idea is to leverage a reconfiguration to improve the related port usage and reduce the completion time of a data transfer if its current completion time is still longer than $\tau_{\min}$. Therefore, *Line* 4 gets the current running time t of the Coflow , and *Lines* 5-7 calculate the remaining running time $\hat{t}$ of the data transfers from *ToR* i. If we have $t + \hat{t} > \tau_{\min}$, *Algorithm* 1 is applied to re-plan the AOI's topology according to the remaining DTM $\tilde{\mathbf{D}}$ (*Line* 9). Then, *Lines* 10-11 check the remaining running time $\tau'_{\min}$ in the new topology, and only reconfigure the AOI if the sum of $\tau'_{\min}$ and reconfiguration latency ϵ is less than the current remaining running time $\hat{t}$.

Algorithm 1. Calculating AOI Configuration

Input: N, $\tilde{\mathbf{D}}$
Output: $G = (V,E)$, $\tau_{\min}$, $\{b_{i,j},\ i,j \in [1,N]\}$
1: $V = \{v_i, i \in [1,N]\}$, $E = \emptyset$, $\{b_{i,j} = 0,\ \forall i,j \in [1,N]\}$
2: $\{N_i^+ = N_i^- = N,\ \forall i \in [1,N]\}$, $\tau_{\min} = 0$
3: **for** each ToR $i \in [1,N]$ **do**
4: $\quad B_1 = \frac{1}{N}\sum_{j=1}^{N} \tilde{d}_{i,j}$, $B_2 = \frac{1}{N}\sum_{j=1}^{N} \tilde{d}_{j,i}$
5: $\quad \tau_{\min} = \max(\tau_{\min}, B_1, B_2)$
6: **for** each ToR $i \in [1,N]$ **do**
7: $\quad$ **for** each ToR $j \in [1,N]$ **do**
8: $\quad\quad$ **if** $\tilde{d}_{i,j} > 0$ **then**
9: $\quad\quad\quad (v_i, v_j) \to E$, $N_i^+ = N_i^+ - 1$, $N_j^- = N_j^- - 1$
10: $\quad\quad\quad b_{i,j} = b_{i,j} + 1$
11: **for** each $\tilde{d}_{i,j} > 0$ in $\tilde{\mathbf{D}}$ **do**
12: $\quad$ **if** no feasible path from v_j to v_i **then**
13: $\quad\quad$ add (v_j, v_i) into E and update N_j^+, N_i^- and $b_{j,i}$
14: **for** each ToR $i \in [1,N]$ **do**
15: $\quad$ **for** each ToR $j \in [1,N]$ **do**
16: $\quad\quad \tau_{i,j} = \tilde{d}_{i,j}$
17: $\quad\quad$ **if** $\tau_{i,j} \leq \tau_{\min}$ **then**
18: $\quad\quad\quad$ **continue**
19: $\quad\quad$ **while** $(N_i^+ > 0)$ AND $(N_j^- > 0)$ **do**
20: $\quad\quad\quad b_{i,j} = b_{i,j} + 1$
21: $\quad\quad\quad \tau_{i,j} = \frac{\tau_{i,j}}{b_{i,j}}$
22: $\quad\quad\quad (v_i, v_j) \to E$, $N_i^+ = N_i^+ - 1$, $N_j^- = N_j^- - 1$;
23: $\quad\quad\quad$ **if** $\tau_{i,j} \leq \tau_{\min}$ **then**
24: $\quad\quad\quad\quad$ **break**
25: use available WDM channels to color edges in $G(V,E)$
26: **return** $G = (V,E)$, $\tau_{\min}$, $\{b_{i,j},\ i,j \in [1,N]\}$

B. Algorithm for Multi-Coflow Scenario

As for the algorithm that addresses the multi-Coflow scenario, we would like to first reduce the average CCT by optimizing the serving order of Coflows. Specifically, we get the necessary CCT $\tau_{\min,k}$ of each Coflow k and serve the Coflows in the ascending order of their necessary CCTs. Meanwhile, as $\tau_{\min,k}$ depends on the remaining DTM $\tilde{\mathbf{D}}_k$, its ranking may change as time goes (or a new Coflow comes in). Therefore, we design our algorithm based on the preemptive scheduling, which updates the serving order of Coflows based on their latest necessary CCTs and prioritizes the Coflow with the shortest $\tau_{\min,k}$ from time to time.

Algorithm 3 shows our proposed algorithm for addressing the multi-Coflow scenario, where we store all the pending Coflows in set $\mathbf{C}$ and denote the main active Coflow as c^* (whose current necessary CCT is $\tau_{\min,*}$). *Line* 2 ensures that $\mathbf{C}$ is always up-to-date. Then, we get the current remaining DTM $\tilde{\mathbf{D}}_k$ of each pending Coflow c_k, update its necessary CCT, and sort the Coflows in $\mathbf{C}$ accordingly (*Lines* 3-9). Next, if there does not exist a main active Coflow, *Lines* 10-12 select the first Coflow in $\mathbf{C}$ (*i.e.*, the one with the shortest necessary CCT) as c^*, and serve it with *Algorithm* 2. Here, the *Algorithm* 4 in *Line* 12 enables the provisioning of the Coflows other than c^* with the remaining bandwidth resources in the AOI, and we will discuss it later. Otherwise, if there exists a main active Coflow c^* but its necessary CCT satisfies $\tau_{\min,1} + \epsilon < \tau_{\min,*}$ (*i.e.*, the main active Coflow should be replaced with the first Coflow in $\mathbf{C}$), we stop the provisioning of c^* and update it as the first one in $\mathbf{C}$ (*Lines* 13-16).

Algorithm 4 explains how to leverage the remaining bandwidth resources during serving the main active Coflow for provisioning other Coflows in $\mathbf{C}$. *Line* 1 is for the initialization. Then, in the for-loop that covers *Lines* 2-5, we check each pend-

Algorithm 2. Topology Management for Single-Coflow

Input: $N, c, \mathbf{D}, \epsilon$
apply *Algorithm* 1 with $\tilde{\mathbf{D}} = \mathbf{D}$ to get $G = (V, E)$, $\tau_{\min}$, and $\{b_{i,j},\ i,j \in [1,N]\}$, and serve Coflow c accordingly
while Coflow c has not been completed **do**
  if a data transfer $d_{i,j} \in \mathbf{D}$ is completed **then**
    record the current running time in t
    get the remaining DTM $\tilde{\mathbf{D}}$, and assign $\hat{t} = 0$
    for each ToR $j' \in [1,N]$ **do**
      $\hat{t} = \max\left(\hat{t}, \frac{\tilde{d}_{i,j'}}{b_{i,j'}}\right)$
    if $t + \hat{t} > \tau_{\min}$ **then**
      apply *Algorithm* 1 with $\tilde{\mathbf{D}}$ to get new $G = (V, E)$, $\tau'_{\min}$, and $\{b_{i,j},\ i,j \in [1,N]\}$
      if $\hat{t} > \tau'_{\min} + \epsilon$ **then**
        reconfigure AOI according to new $G = (V, E)$ for serving Coflow c

Algorithm 3. Topology Management for Multi-Coflow

Input: $N, \epsilon, \mathbf{C}$
while there still exist unfinished Coflows **do**
  update $\mathbf{C}$ for newly-arrived/finished Coflow(s)
  for each Coflow $c_k \in \mathbf{C}$ **do**
    get remaining DTM $\tilde{\mathbf{D}}_k$ of c_k
    $\tau_{\min,k} = 0$
    for each ToR $i \in [1,N]$ **do**
      $B_1 = \frac{1}{N}\sum_{j=1}^{N} \tilde{d}^k_{i,j}$, $B_2 = \frac{1}{N}\sum_{j=1}^{N} \tilde{d}^k_{j,i}$
      $\tau_{\min,k} = \max\left(\tau_{\min,k}, B_1, B_2\right)$
  sort Coflows in $\mathbf{C}$ in ascending order of $\{\tau_{\min,k}\}$
  if there does not exist an active Coflow **then**
    select the first Coflow c_1 in $\mathbf{C}$ as c^*
    use *Algorithm* 2 to serve c^* (with *Algorithm* 4)
  else if $\tau_{\min,1} + \epsilon < \tau_{\min,*}$ **then**
    stop *Algorithm* 2
    update c^* to be the first Coflow c_1 in $\mathbf{C}$
    use *Algorithm* 2 to serve c^* (with *Algorithm* 4)

ing Coflow in $\mathbf{C} \setminus c^*$, and try to relieve its bottleneck by assigning edges to it greedily. Next, if there still exist available ports, we allocate them to the data transfers of pending Coflows in sorted order. Finally, for the updated graph $G(V, E)$, *Line* 8 assigns WDM channels to the newly-added edges under the constraints in Eq. 2. Note that, to reduce the number of reconfigurations, *Algorithm* 4 is only invoked when an AOI reconfiguration is about to happen in *Algorithm* 2.

C. Complexity Analysis

Table 1 lists the time complexity of our algorithms, which suggests that all of them run in polynomial time. Meanwhile, the mutual calls among the algorithms are also in constant levels. Therefore, our algorithms can solve the problems of topology management and Coflow scheduling for the single- and multi-Coflow scenarios in Hyper-FleX-LION time-efficiently.

5. PERFORMANCE EVALUATIONS

In this section, we discuss the numerical simulations that evaluate the performance of Hyper-FleX-LION on serving Coflow and our proposals.

Algorithm 4. Filling Free Ports with Coflows

Input: $G(V, E)$, $\mathbf{C}$
Output: $G(V, E)$
calculate $\{N_i^+, N_i^-, \forall i \in [1,N]\}$ according to $G(V, E)$
for each Coflow $c_k \in \mathbf{C} \setminus c^*$ **do**
  get its remaining DTM $\tilde{\mathbf{D}}_k$
  find the bottleneck that prolongs its CCT
  add edges into E greedily to relieve the bottleneck and update the corresponding $\{N_i^+, N_i^-\}$
while there still exist $N_i^+ > 0$ and $N_i^- > 0$ and pending Coflows **do**
  add edges into E greedily for pending Coflows and update the corresponding $\{N_i^+, N_i^-\}$
use available WDM channels to color edges in $G(V, E)$
return $G(V, E)$

Table 1. Time Complexity of Algorithms

Algorithms	Time Complexity
Algorithm 1	$O(3N^2 + N^3)$
Algorithm 2 (each loop)	$O(N + 3N^2 + N^3)$
Algorithm 3 (each loop)	$O(N + (\lvert\mathbf{C}\rvert + 3) \cdot N^2 + N^3 + \lvert\mathbf{C}\rvert^2)$
Algorithm 4	$O((2\lvert\mathbf{C}\rvert + \lvert\mathbf{C}\rvert^2) \cdot N^2)$

A. Simulation Setup

In the simulations, we assume that the Coflows are generated by MapReduce. Specifically, we leverage the MapReduce traces collected by Facebook and Yahoo [40] and adopt them to generate the Coflows in the simulations. In order to describe the Coflows clearly, we define three parameters as follows.

- *Size of Coflow* (α): It denotes the value of the largest element in the DTM $\mathbf{D}_k$ of a Coflow c_k, *i.e.*, $\alpha = \max_{\forall i,j}(d^k_{i,j})$.
- *Density of Coflow* (β): It denotes the ratio of nonzero elements in the DTM $\mathbf{D}_k$ of a Coflow c_k. The higher the density of a Coflow is, the more complex its inter-rack communications are. If the density of a Coflow equals 1, its DTM requires all-to-all communications.
- *Number of Coflows* (γ): It describes the number of Coflows that come in within a fixed interval.

With the parameters above, the simulations generate the Coflows as follows. We first follow a specific density of Coflow β to finalize the number of nonzero elements in the DTM of a Coflow, and randomly choose the nonzero elements. Then, for each nonzero element in the DTM, we randomly set its value according to a specific size of Coflow α within $[\frac{\alpha}{10}, \alpha]$ Gb. In order to emulate realistic traces, we leverage the scheme used in [40] to combine the Coflows of different sizes as three workload scenarios, as shown in Table 2.

The simulations consider four densities of Coflow as $\beta \in \{0.2, 0.4, 0.6, 0.8\}$, and set the duration of each simulation as 1,000 seconds, *i.e.*, dynamic Coflows are generated and served within $[0, 1000]$ seconds. In order to verify the algorithms' performance under different arrival speeds of Coflows, we simulate four scenarios, which have their number of Coflows set as $\gamma \in \{200, 400, 600, 800\}$ in each simulation, respectively.

Table 2. Workload Scenarios in Simulations

Scenarios \ Size in Gb (α)	10	100	500	1,000
Low	60%	40%	-	-
Medium	40%	40%	20%	-
High	30%	30%	20%	20%

We use the number of racks N to denote the scale of an AOI in Hyper-FleX-LION. Since the scale of Hyper-FleX-LION can be limited by a few factors, we set the upper-bound of N as 64 in the simulations, according to the analysis in [28]. Specifically, the simulations consider five types of Hyper-FleX-LION in different scales, with $N \in \{4, 8, 16, 32, 64\}$. The bandwidth capacity of each TRX is set as $B = 10$ Gbps, and the latency of each reconfiguration is assumed to be $\epsilon = 1$ second, according to the experimental results in [30].

For performance evaluations, we introduce the metrics of *acceleration ratio*. Specifically, we use the average CCT (defined in Eq. (3)) of Hyper-FleX-LION or our proposals as the baseline, compare the average CCT of the benchmarks to it, and obtain the ratio between them as the acceleration ratio. For instance, if the average CCT of our algorithm is 2 seconds while that of a benchmark is 5 seconds, the acceleration ratio achieved by our algorithm is 2.5. To maintain sufficient statistical accuracy, the simulations average the results from 60 independent runs to get each data point. Meanwhile, to show the stability of algorithms in the simulations, we mark the range of 95% confidence interval in our results.

B. Comparison of AOIs with OXC and Hyper-FleX-LION

We first compare an AOI in Hyper-FleX-LION with that built with OXC to see their performance on Coflow scheduling. The detailed simulation setup is as follows

- AOI in Hyper-FleX-LION: The AOI is built with an N-Hyper-FleX-LION, and the Coflows in it are served with *Algorithms* 1-4 designed in Section 4.
- AOI with OXC: The AOI is built with an $N \times N$ OXC, which can switch at the port level. Specifically, as shown in Fig. 4, we replace the WSS' and AWGR in Hyper-FleX-LION with such an OXC, *i.e.*, each ToR still equips N TRXs whose outputs are multiplexed and connect to one port on the OXC. Then, the capacity of each port on OXC is $N \cdot B$. We use the algorithm developed in [25] to serve Coflows in the AOI.

The simulations set the parameters of Coflows as $\beta = 0.4$, $\gamma = 400$, and consider all the workload scenarios in Table 2.

Fig. 5 shows the results on normalized average CCT obtained in AOIs with OXC and Hyper-FleX-LION. Here, for illustrative comparisons, we first normalize the average CCT obtained in the AOI in Hyper-FleX-LION as 1, and then respectively normalize the results obtained in the AOI with OXC accordingly. Therefore, the normalized average CCT of the AOI with OXC can also be understood as the acceleration ratios of the AOI in Hyper-FleX-LION achieved over it. We first only consider the medium workload scenario and simulate AOIs in various scales. Fig. 5(a) illustrates the results, which indicate that the AOI in Hyper-FleX-LION always provides a shorter average CCT than that with OXC. More importantly, the advantage of

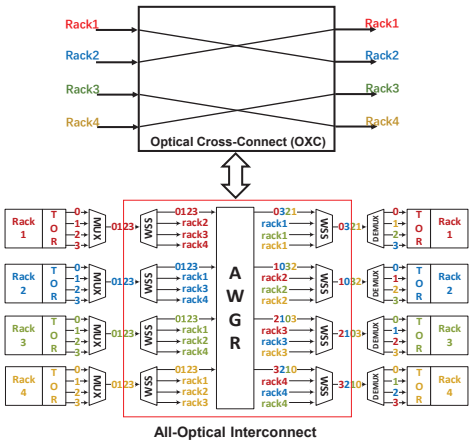


Fig. 4. Structures of AOI in Hyper-FleX-LION and AOI with OXC.

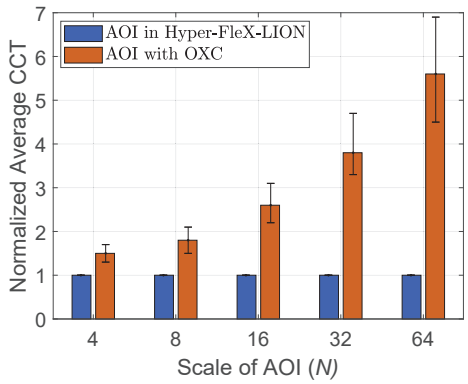
Hyper-FleX-LION increases significantly with the scale of the AOI. The reason for this is two-fold: 1) compared with OXC, Hyper-FleX-LION provides more flexibility for managing the AOI topology, which is essential for scheduling Coflows adaptively, and 2) our proposed algorithms are well-designed to fully explore the flexibility of Hyper-FleX-LION. In Fig. 5(a), the acceleration ratios of the AOI in Hyper-FleX-LION over that with OXC have an average of 3.06 and the maximum of 5.6.

Fig. 5(b) shows the results on normalized average CCT for different workloads, which are obtained by fixing the scale as $N = 16$. We can see that the AOI in Hyper-FleX-LION still always outperforms that with OXC. This time, when the workload increases, the performance gap between the two types of AOIs actually decreases. This is because for a higher workload, more of the TRXs in the AOI are saturated, which makes the flexibility of Hyper-FleX-LION less beneficial. Note that, according to the analysis in [40], the Coflows in very large sizes in the medium and high workload scenarios (*i.e.*, those with $\alpha = \{500, 1000\}$ Gb) are actually rare in realistic DCNs. Therefore, the AOI's performance for the low workload scenario is more important in practice. In Fig. 5(b), the acceleration ratios of the AOI in Hyper-FleX-LION have an average of 3.3 and the maximum of 5.1.

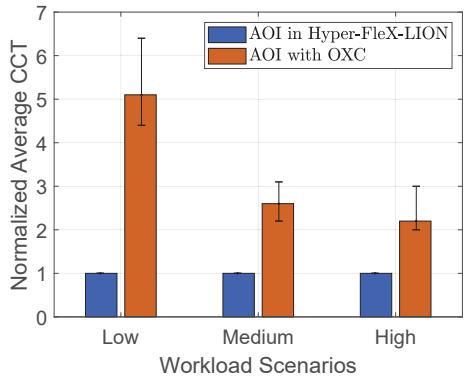
C. Coflow Scheduling in Hyper-FleX-LION

We then further evaluate the performance of our proposed algorithms for the topology management and Coflow scheduling in AOIs in Hyper-FleX-LION. The simulations consider the following algorithms. As we have explained, due to the uniqueness of Hyper-FleX-LION, the algorithms designed for scheduling Coflows in other types of AOIs become inapplicable in it. Therefore, it is difficult for us to use existing algorithms in the literature as benchmarks.

- **Ours**: Our complete proposal that uses *Algorithms* 1-4.
- **Rand**: The algorithm that follows the general procedure of Ours but allocates TRXs to Coflows randomly, not based on their necessary CCTs.
- **SimpleFill**: The algorithm that follows the general procedure of Ours but does not use *Algorithm* 4. Instead, it randomly selects unserved Coflows to fill the unused TRXs in the AOI.



(a) Performance of AOIs in various scales (medium workload)



(b) Performance for different workloads ($N = 16$)

Fig. 5. Normalized average CCT in AOIs with OXC and Hyper-FleX-LION.

- **FAFS**: The algorithm that follows the general procedure of Ours, except for the preemptive scheduling. Specifically, it serves the Coflows in the order of their arrival time (*i.e.*, first arrive and first serve (FAFS)), and only proceeds to serve the next Coflow when the current one has been accomplished.

We first perform simulations to compare the algorithms' performance in different scales of Hyper-FleX-LION. Specifically, we set $\beta = 0.6$ and $\gamma = 600$, select the medium workload scenario, and change N from 4 to 64. Fig. 6 shows the results on normalized average CCT. This time, we first normalize the average CCT obtained with Ours as 1, and then respectively normalize the results of other algorithms accordingly. Therefore, the results of other algorithms can also be understood as the acceleration ratios of Ours achieved over it. We observe that Ours performs the best to provide the shortest average CCT, followed by FAFS, while the performance of Rand is the worst. Moreover, the performance gap between Ours and Rand increases significantly with N. This actually confirms the effectiveness of our bandwidth allocation scheme based on Coflow's necessary CCTs, *i.e.*, the scheme can effectively relieve the bottleneck of the data transfers in each Coflow. Meanwhile, the advantages of Ours over SimpleFill and FAFS verify the effectiveness of *Algorithm* 4 and our preemptive scheduling. In Fig. 6, the acceleration ratios of Ours over Rand, SimpleFill and FAFS have averages of 3.08, 1.68, and 1.32, respectively.

Then, we fix $N = 32$ and consider different workload scenarios. The results are shown in Fig. 7, which indicates that Ours still performs the best, and the Rand's performance is still the worst. This further justifies the need of our necessary CCT based bandwidth allocation scheme. Meanwhile, it is interesting to notice that the performance gap between Ours and FAFS increases with the workload, while that between Ours and the SimpleFill decreases, and SimpleFill can perform worse than FAFS in the high workload scenario. This suggests that when the workload is higher, the importance of preemptive scheduling becomes more but *Algorithm* 4 tends to be less beneficial. In Fig. 7, the acceleration ratios of Ours over Rand, SimpleFill and FAFS have averages of 4, 1.86, and 1.43, respectively.

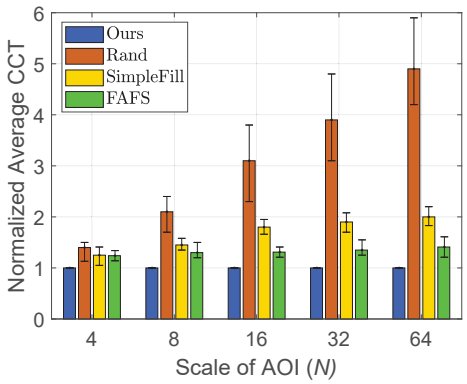


Fig. 6. Performance of algorithms for Hyper-FleX-LION in different scales.

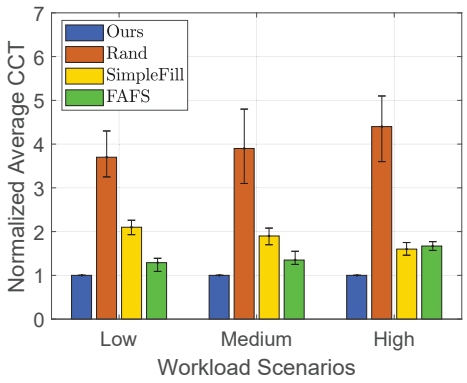


Fig. 7. Performance of algorithms for different workloads.

Finally, we choose the medium workload scenario, fix $N = 32$, and change the values of β and γ to study their effects on the algorithms' performance, respectively. Fig. 8 explains the effect of the density of Coflows β. Ours still performs the best, followed by FAFS, and the performance of Rand is the worst, except for the case of $\beta = 0.8$, where SimpleFill is the second best algorithm. It can be seen that the performance gaps between Ours and other algorithms generally decrease with the density of Coflows β except FAFS. This is because when β increases, the DMTs of Coflows become less sparse, which makes the adaptive topology management enabled by Ours less effective over the benchmarks. Meanwhile, a larger β can indirectly lead to more Coflows being queued after their arrivals, which increases the advantage of preemptive scheduling. Note that, according to the study in [41], the inter-rack traffic density in realistic DCNs

is usually low. In Fig. 8, the acceleration ratios of Ours over Rand, SimpleFill and FAFS have averages of 4.17, 2.08, and 1.39, respectively.

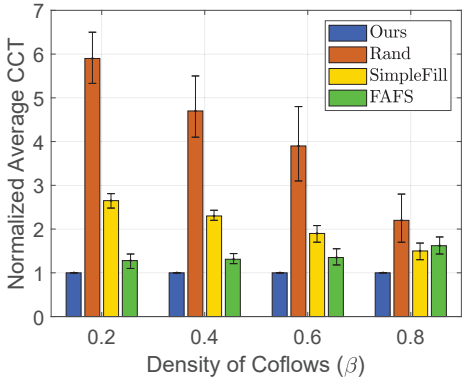


Fig. 8. Performance of algorithms for different densities of Coflows.

Fig. 9 shows the effect of the number of Coflows γ. Once again, Ours still performs the best, followed by FAFS, and the performance of Rand is the worst. This time, the performance gaps between Ours and other algorithms increase with the number of Coflows γ. This is because when γ increases, the arrival interval between adjacent Coflows decreases, which makes the adaptive topology management and Coflow scheduling enabled by Ours more effective over the benchmarks. In Fig. 9, the acceleration ratios of Ours over Rand, SimpleFill and FAFS have averages of 4.22, 1.9, and 1.4, respectively.

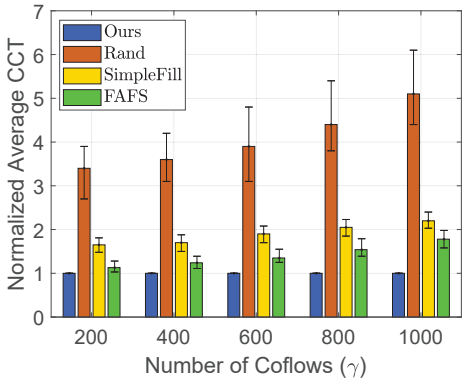


Fig. 9. Performance of algorithms for different numbers of Coflows.

In all, the simulation results in Figs. 6-9 prove that Ours is well-designed to fully explore the flexibility of Hyper-FleX-LION for optimizing the topology management and Coflow scheduling in AOIs in Hyper-FleX-LION.

6. CONCLUSION

In this work, we studied the topology management and Coflow scheduling in AOIs in Hyper-FleX-LION. We first proposed time-efficient algorithms to address the provisioning of a single Coflow and then extended them for multi-Coflow scenarios. Our simulation results confirmed that AOIs in Hyper-FleX-LION could accelerate Coflows better than the traditional OXC-based AOIs, and achieved an acceleration of up to 5.6×. As for the Coflow scheduling in Hyper-FleX-LION, our proposal could effectively reduce the average CCT of Coflows and outperform all the benchmarks significantly.

ACKNOWLEDGMENTS

This work was supported by NSFC project 61871357 and Fundamental Fund for Central Universities (WK3500000006).

DISCLOSURES

The authors declare no conflicts of interest.

REFERENCES

1. P. Lu, L. Zhang, X. Liu, J. Yao, and Z. Zhu, "Highly-efficient data migration and backup for Big Data applications in elastic optical inter-datacenter networks," IEEE Netw. **29**, 36–42 (2015).
2. W. Lu, L. Liang, B. Kong, B. Li, and Z. Zhu, "AI-assisted knowledge-defined network orchestration for energy-efficient data center networks," IEEE Commun. Mag. **58**, 86–92 (2020).
3. S. Li, D. Hu, W. Fang, S. Ma, C. Chen, H. Huang, and Z. Zhu, "Protocol oblivious forwarding (POF): Software-defined networking with enhanced programmability," IEEE Netw. **31**, 58–66 (2017).
4. N. Bitar, S. Gringeri, and T. Xia, "Technologies and protocols for data center and cloud networking," IEEE Commun. Mag. **51**, 24–31 (2013).
5. Z. Zhu, W. Lu, L. Zhang, and N. Ansari, "Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing," J. Light. Technol. **31**, 15–22 (2013).
6. L. Gong, X. Zhou, X. Liu, W. Zhao, W. Lu, and Z. Zhu, "Efficient resource allocation for all-optical multicasting over spectrum-sliced elastic optical networks," J. Opt. Commun. Netw. **5**, 836–847 (2013).
7. Y. Yin, H. Zhang, M. Zhang, M. Xia, Z. Zhu, S. Dahlfort, and S. Yoo, "Spectral and spatial 2D fragmentation-aware routing and spectrum assignment algorithms in elastic optical networks," J. Opt. Commun. Netw. **5**, A100–A106 (2013).
8. W. Shi, Z. Zhu, M. Zhang, and N. Ansari, "On the effect of bandwidth fragmentation on blocking probability in elastic optical networks," IEEE Trans. Commun. **61**, 2970–2978 (2013).
9. N. Farrington, G. Porter, S. Radhakrishnan, H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," ACM SIGCOMM Comput. Commun. Rev.. **40**, 339–350 (2010).
10. G. Wang, D. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-Through: Part-time optics in data centers," ACM SIGCOMM Comput. Commun. Rev.. **41**, 327–338 (2011).
11. J. Benjamin, T. Gerard, D. Lavery, P. Bayvel, and G. Zervas, "PULSE: Optical circuit switched data center architecture operating at nanosecond timescales," J. Lightw. Technol.. **38**, 4906–4921 (2020).
12. H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, B. Thomsen, K. Shi, and H. Williams, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. of ACM SIGCOMM 2020,* (2020), pp. 782–797.
13. G. Liu, R. Proietti, M. Fariborz, P. Fotouhi, X. Xiao, and B. Yoo, "Architecture and performance studies of 3D-Hyper-FleX-LION for reconfigurable All-to-All HPC networks," in *Proc. of SC 2020,* (2020), pp. 1–16.
14. Y. Lu and H. Gu, "Flexible and scalable optical interconnects for data centers: Trends and challenges," IEEE Commun. Mag. **57**, 27–33 (2019).
15. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *Proc. of OSDI 2016,* (2016), pp. 265–283.
16. L. Gong and Z. Zhu, "Virtual optical network embedding (VONE) over elastic optical networks," J. Light. Technol. **32**, 450–460 (2014).

17. Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted NFV service chain deployment based on affiliation-aware vNF placement," in *Proc. of GLOBECOM 2016,* (2016), pp. 1–6.
18. J. Liu, W. Lu, F. Zhou, P. Lu, and Z. Zhu, "On dynamic service function chain deployment and readjustment," IEEE Trans. Netw. Serv. Manag. **14**, 543–553 (2017).
19. K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," IEEE/ACM Trans. Netw. **22**, 498–511 (2013).
20. M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. of ACM HotNets 2012,* (2012), pp. 31–36.
21. M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient Coflow scheduling with Varys," in *Proc. of ACM SIGCOMM 2014,* (2014), pp. 443–454.
22. H. Zhang, X. Shi, X. Yin, and Z. Wang, "Yosemite: Efficient scheduling of weighted coflows in data centers," in *Proc. of ICNP 2017,* (2017), pp. 1–2.
23. M. Shafiee and J. Ghaderi, "An improved bound for minimizing the total weighted completion time of Coflows in datacenters," IEEE/ACM Trans. Netw. **26**, 1674–1687 (2018).
24. H. Wang, X. Yu, H. Xu, J. Fan, C. Qiao, and L. Huang, "Integrating Coflow and circuit scheduling for optical networks," IEEE Trans. Parallel Distrib. Syst. **30**, 1346–1358 (2019).
25. H. Tan, C. Zhang, C. Xu, Y. Li, Z. Han, and X. Li, "Regularization-based Coflow scheduling in optical circuit switches," IEEE/ACM Trans. Netw. **29**, 1280–1293 (2021).
26. Z. Li and H. Shen, "Co-Scheduler: A Coflow-aware data-parallel job scheduler in hybrid electrical/optical datacenter networks," IEEE/ACM Trans. Netw., Press. pp. 1–14 (2022).
27. Y. Lu, H. Gu, X. Yu, and P. Li, "Fast control plane for flexible and scalable optical interconnects," Opt. Express **30**, 3316–3328 (2022).
28. X. Xiao, R. Proietti, G. Liu, H. Lu, P. Fotouhi, S. Werner, Y. Zhang, and B. Yoo, "Silicon photonic Flex-LIONS for bandwidth-reconfigurable optical interconnects," IEEE J. Sel. Top. Quantum Electron. **26**, 1–10 (2020).
29. X. Xiao, R. Proietti, G. Liu, H. Lu, Y. Zhang, and B. Yoo, "Multi-FSR silicon photonic Flex-LIONS module for bandwidth-reconfigurable all-to-all optical interconnects," J. Lightw. Technol.. **38**, 3200–3208 (2020).
30. H. Yang, Z. Zhu, R. Proietti, and B. Yoo, "Which can accelerate distributed machine learning faster: Hybrid optical/electrical or optical reconfigurable DCN?" in *Proc. of OFC 2022,* (2022), pp. 1–3.
31. T. Zhang, R. Shu, Z. Shan, and F. Ren, "Distributed bottleneck-aware Coflow scheduling in data centers," IEEE Trans. Parallel Distrib. Syst. **30**, 1565–1579 (2019).
32. C. Chiu, D. Singh, Q. Wang, K. Lee, and S. Park, "Minimal Coflow routing and scheduling in OpenFlow-Based cloud storage area networks," in *Proc. of IEEE CLOUD 2017,* (2017), pp. 222–229.
33. H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward automatically identifying and scheduling Coflows in the dark," in *Proc. of ACM SIGCOMM 2016,* (2016), pp. 160–173.
34. S. Zhao and Z. Zhu, "On virtual network reconfiguration in hybrid optical/electrical datacenter networks," J. Light. Technol. **38**, 6424–6436 (2020).
35. Q. Li, H. Fang, D. Li, J. Peng, J. Kong, W. Lu, and Z. Zhu, "Scalable knowledge-defined orchestration for hybrid optical/electrical datacenter networks," J. Opt. Commun. Netw.. **12**, A113–A122 (2020).
36. X. Xue, F. Yan, K. Prifti, F. Wang, B. Pan, X. Guo, S. Zhang, and N. Calabretta, "ROTOS: A reconfigurable and cost-effective architecture for high-performance optical data center networks," J. Light. Technol. **38**, 3485–3494 (2020).
37. S. Liu, B. Niu, D. Li, M. Wang, S. Tang, J. Kong, B. Li, X. Xie, and Z. Zhu, "DL-assisted cross-layer orchestration in software-defined IP-over-EONs: From algorithm design to system prototype," J. Lightw. Technol.. **37**, 4426–4438 (2019).
38. G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," in *Proc. of ACM SIGCOMM 2013,* (2013), pp. 447–458.
39. "Edge coloring," Wikipedia [Online] .
40. Y. Chen, A. Ganapathi, R. Griffith, and R. Katz, "The case for evaluating MapReduce performance using workload suites," in *Proc. of MASCOTS 2011,* (2011), pp. 390–399.
41. S. Chandrasekaran, *Understanding Traffic Characteristics in a Server to Server Data Center Network* (Master Thesis, Rochester Institute of Technology, 2017).