




Article

Job-Deadline-Guarantee-Based Joint Flow Scheduling and Routing Scheme in Data Center Networks

Long Suo ^{1,*} , Han Ma ², Wanguo Jiao ¹  and Xiaoming Liu ¹ 

¹ College of Information Science and Technology, Nanjing Forestry University, Nanjing 210037, China; wgjiao@njfu.edu.cn (W.J.); lxm@njfu.edu.cn (X.L.)

² State Key Laboratory of Integrated Service Networks, Xidian University, Xi'an 710071, China; hanma@stu.xidian.edu.cn

* Correspondence: lsuo@njfu.edu.cn

Abstract: Many emerging Internet of Things (IoT) applications deployed on cloud platforms have strict latency requirements or deadline constraints, and thus meeting the deadlines is crucial to ensure the quality of service for users and the revenue for service providers in these delay-stringent IoT applications. Efficient flow scheduling in data center networks (DCNs) plays a major role in reducing the execution time of jobs and has garnered significant attention in recent years. However, only few studies have attempted to combine job-level flow scheduling and routing to guarantee meeting the deadlines of multi-stage jobs. In this paper, an efficient heuristic joint flow scheduling and routing (JFSR) scheme is proposed. First, targeting maximizing the number of jobs for which the deadlines have been met, we formulate the joint flow scheduling and routing optimization problem for multiple multi-stage jobs. Second, due to its mathematical intractability, this problem is decomposed into two sub-problems: inter-coflow scheduling and intra-coflow scheduling. In the first sub-problem, coflows from different jobs are scheduled according to their relative remaining times; in the second sub-problem, an iterative coflow scheduling and routing (ICSR) algorithm is designed to alternately optimize the routing path and bandwidth allocation for each scheduled coflow. Finally, simulation results demonstrate that the proposed JFSR scheme can significantly increase the number of jobs for which the deadlines have been met in DCNs.

Keywords: data center networks; cloud computing; coflow; flow scheduling; deadline



Citation: Suo, L.; Ma, H.; Jiao, W.; Liu, X. Job-Deadline-Guarantee-Based Joint Flow Scheduling and Routing Scheme in Data Center Networks. *Sensors* **2024**, *24*, 216. <https://doi.org/10.3390/s24010216>

Academic Editor: Anfeng Liu

Received: 17 November 2023

Revised: 25 December 2023

Accepted: 25 December 2023

Published: 30 December 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The last decade has witnessed explosive growth in the number of devices and data traffic from the Internet of Things (IoT). Cloud computing can provide flexible and scalable computing, storage, and networking resources in an on-demand, pay-as-you-go service model, which makes it more convenient for users to deploy and maintain various IoT applications [1]. For most IoT solutions, devices collect and send massive raw data to cloud data centers for storage, analysis, and decision making. Many emerging IoT applications have strict latency requirements or deadline constraints, such as for virtual reality, augmented reality, smart homes, smart cities, smart energies, and smart vehicles [2]. Keeping low latency and meeting deadlines is vital for both the users' quality of service (QoS) and the service providers' revenue in these delay-stringent IoT applications. Taking an intelligent monitoring service for example, if the response times for user requests are too long, users will resubmit or just give up their requests. If so, the abandoned overdue responses will degrade the user experience and waste the computing and communication resources. According to the statistics in [3], a 100 ms latency increase generates a 1% income loss at Amazon, while a 400 ms delay increase in search responses can reduce search volume by 0.7% for Google.

Offloading computation-intensive and delay-sensitive tasks to edge data centers is an efficient way to reduce the end-to-end communication delays in IoT applications [4,5].

However, edge computing is unable to cut down the processing time of IoT tasks in data centers. Though infrastructure providers continually updates the CPU and storage capacity of commercial servers and the link bandwidth of switches, network resources still have a performance bottleneck when handling cloud computing tasks due to the distributed nature of parallel computing and the massive communication demands among servers within data centers [6]. Popular cluster computing frameworks such as MapReduce [7] and Spark [8] have been widely applied to process the explosively growing data volume. In those typical cloud computing architectures, a cloud computing task, or a job, is usually divided into multiple stages, and each stage also includes several computing tasks that are parallel processed on different physical servers. In the cluster computing manner, the output data resulting from upper-stage computing tasks are exactly the necessary inputs for the lower-stage computing tasks, and thus a large quantity of intermediate data should be exchanged among different physical servers in a data center via data center networks (DCNs). Further, a lower-stage computing task shall not begin unless the result data from all of its upper-stage tasks have been completely delivered. It is estimated that parallel intermediate data transfers account for 33–50% of whole-job completion times [9]. Therefore, improving DCN transmission efficiency is critical for reducing the processing delays of cloud computing jobs.

Some researchers have studied deadline-aware flow scheduling in DCNs [10–14]. However, due to the distributed nature of cloud computing, these traditional deadline-aware scheduling schemes for individual flows are no longer applicable. After analyzing the communication patterns of typical cluster applications, Chowdhury et al. found that computing machines are grouped according to job stages or functionality, and concurrent intermediate flows between two adjacent computing stages usually have associated semantics and a collective objective [15]. This semantically related flow collection between two computing machine groups is referred to as coflow. According to the manner of cluster computing, a coflow's completion time depends on the completion time of the last flow [16]. Thus, traditional flow-level scheduling schemes are inefficient for reducing the completion times or meeting the deadline of coflows. Further, since one cloud computing job usually contains multistage computing tasks and coflows, the dependency relationships among coflows will influence the final completion time of the job, which should be carefully considered in job scheduling. Moreover, a DCN can provide multiple routing paths between each server pair to guarantee connection reliability in typical Fat-Tree or Spine-Leaf architectures [17], and thus routing is another vital factor for reducing the job completion time for cloud data centers.

For cluster computing applications, existing deadline-aware flow scheduling schemes can be roughly classified into three types: single-stage coflow scheduling, multi-stage coflow scheduling, and joint flow scheduling and routing. In the first type, researchers usually treat different coflows as multiple independent entities and focus on meeting the deadline of each coflow separately. Chowdhury et al. modeled the inter-coflow scheduling problem as a concurrent open-shop scheduling problem and proposed heuristic algorithms to minimize the coflow completion time (CCT) or to maximize the number of deadline-guaranteed coflows [18]. They designed the Smallest Effective Bottleneck First (SEBF) algorithm and the Minimum Allocation for Desired Duration (MADD) algorithm to perform inter-coflow scheduling and intra-coflow scheduling, respectively. The SEBF algorithm calculates the predicted CCTs of all coflows and preferentially schedules the coflow with the minimum predicted CCT; the MADD algorithm determines the transmitted rates of all flows in the same coflow so that they shall have the same completion time. Chowdhury et al. also developed the coordinated coflow scheduler Varys and provided coflow APIs for cluster computing frameworks. S. Ma et al. proposed the deadline-driven coflow scheduling framework Chronos [19], which allocates bandwidth proportionally to the selected flows and reserves some residual bandwidth for unselected coflows. Therefore, Chronos is more work-conserving and starvation-free than Varys and can avoid complicated coflow admission control. S. Luo et al. realized that the deadline-missed coflow mini-

mization problem is equivalent to the late job minimization problem in a concurrent open shop [20,21]. Thus, on the basis of concurrent open-shop scheduling solutions, they designed a decentralized, deadline-driven, coflow-aware scheduling (D^2 -CAS) system. The deadline-aware coflow scheduling schemes in [18–21] all assume that prior knowledge of coflows, such as the number, size, and endpoint of each flow, can be obtained before scheduling, but in many practical cases, detailed coflow information is not known or is incomplete [22]. Faced with this problem, researchers have proposed some online information-agnostic coflow scheduling schemes [22–24] in which they deduce each coflow's remaining transmission time and divide coflows into different priority queues according to the known attributes or arrived bytes. In the second multi-stage coflow scheduling type, researchers scheduled dependent coflows from precedence-constrained job stages [25–29]. From the perspective of job-level performance rather than the coflow-level, meeting job deadlines is not equivalent to simply minimizing each CCT. Y. Liu et al. formulated the average job completion time (JCT) minimization problem with job deadline and coflow dependency constraints and proposed a two-level heuristic scheduling solution [25]. In their solution, the first level performs the Most Bottleneck First (MBF) algorithm to determine job orders, and the second level performs intra-job and intra-coflow scheduling while considering different coflow dependencies. W. Borjigin et al. divided dependent coflows into different stages and presented a heuristic multi-objective time-saving first (MTF) scheduling algorithm to guarantee meeting job deadlines [26]. S. Zhang formulated the multi-stage inter-coflow scheduling (MICS) problem, partitioned MICS into multiple convex single-stage inter-coflow scheduling (SICS) problems, and designed three online heuristics to balance the fairness and completion times of coflows [27]. Besides heuristic solutions, B. Tian et al. provided a deterministic approximation algorithm [28]. They formulated the multi-stage coflow scheduling problem as a weighted JCT minimization problem, relaxed it into a linear programming problem for a lower bound, and further constructed a $(2M + 1)$ -approximation scheduling algorithm, where M is the number of hosts. J. Wang et al. represented multi-stage jobs as directed acyclic graphs (DAGs) and proposed a genetic-algorithm-based scheduling method to reduce the time complexity while meeting job deadline demands [29].

In the third joint flow scheduling and routing type, the influence of DCN architecture is taken into consideration. The previous two scheduling types all assume that perfect traffic balancing and no over-subscription are achieved in DCNs. As a result, a DCN can be modeled as a large non-blocking switch. However, perfect traffic balancing is unrealistic, since many commercial DCN architectures adopt the equal-cost multipath (ECMP) hashing-based packet forwarding strategy [17,30]. Though some researchers have optimized flow scheduling and routing jointly to reduce the energy consumption in DCNs, they did not take the coflow communication feature into account [31]. Y. Zhao et al. were the first to jointly consider coflow scheduling and routing to optimize the average CCT, and they developed the scheduling framework RAPIER, which is compatible with commodity switches [32]. They formulated the single coflow scheduling optimization problem with routing constraints, proposed a minimum-CCT-based heuristic algorithm to determine the rate and path for each flow, and scheduled coflows with longer waiting times and less CCT as a priority. J. Jiang et al. proposed the coflow scheduler Tailor to monitor the flow bottlenecks and reroute flow to lighter load links [33]. Since RAPIER enables bandwidth preemption and Tailor enables dynamic routing, they cause frequent rerouting operations, which seems unrealistic for real-time implementation and large DCNs. Y. Li et al. designed the OneCoflow and OMCoflow algorithms to address the joint flow scheduling and routing problems for a single coflow and multiple coflows, respectively [34,35]. OneCoflow is based on convex programming and rounding and determines the routing path and bandwidth allocation for a newly arrived coflow; OMCoflow reschedules the bandwidth for each existing coflow when a new coflow arrives or when an old coflow is completed. In OneCoflow and OMCoflow, once a coflow arrives, its routing path is determined, and frequent re-routings are not allowed. Y. Chen et al. proposed the multi-hop coflow routing and scheduling (MCRS) strategy in the popular Spine-Leaf topology and allocated

longer detour paths to coflows to alleviate link congestion [36,37], which is applicable to over-subscribed Spine-Leaf networks. The authors of [32–37] only focused on joint flow scheduling and routing cases at the coflow level, whereas Y. Zeng et al. were the first to study the job-level case [38]. They formulated the multi-stage job joint scheduling and routing problem as a non-linear weighted JCT minimization problem and designed a polynomial-time Multi-stage Job Scheduling (MJS) algorithm. This algorithm can achieve constant approximation ratios in various typical DCN architectures. The MJS algorithm determines the job scheduling order according to the optimal solution of the relaxed linear programming problem and schedules active jobs and coflows one by one. When a new job arrives or when a flow is completed, MJS recalculates the scheduling result until all jobs are finished. However, the assumption that a flow can be suspended in MJS is impractical and will increase the scheduling complexity.

In this paper, we aim to guarantee meeting the deadlines of as many delay-stringent IoT jobs as possible by integrating scheduling-dependent coflows and optimizing routing paths in the DCN topology. The main contributions of this paper are as follows. First, we formalize the multi-job joint scheduling and routing problem with the object of maximizing the number of jobs for which the deadlines have been met. Second, the problem is decomposed and solved by the proposed heuristic two-stage joint flow scheduling and routing (JFSR) scheme. In the first stage, the smallest relative remaining time first (SRRTF) criterion determines the scheduling order of coflows; in the second stage, the Iterative Coflow Scheduling and Routing (ICSR) algorithm calculates the rate and path allocation for each scheduled coflow. Finally, simulation results show that the proposed joint flow scheduling and routing scheme can significantly increase the number of jobs for which the deadlines have been met.

The rest of this paper is organized as follows. In Section 2, we introduce the system model, and in Section 3, we present the deadline-met job number maximization problem with coflow dependency and network constraints. The proposed two-stage JFSR scheme is introduced in Section 4. Simulation results are shown in Section 5, and Section 6 concludes the paper.

2. System Model

In this paper, the DCN topology is modeled as a graph $G = \langle \mathbb{V}, \mathbb{E} \rangle$, where \mathbb{V} is the node set and \mathbb{E} is the link set. The available bandwidth of link $e \in \mathbb{E}$ is denoted by B_e .

The delay-stringent IoT job set to be scheduled is denoted by $\mathbb{J} = \{J_n, 1 \leq n \leq N\}$, and we assume the n -th job J_n contains N_n coflows. The m -th coflow of J_n is denoted by $C_{n,m}$, and the coflow set of J_n is denoted by $\mathbb{C}_n = \{C_{n,m}, 1 \leq m \leq N_n\}$. We also assume that coflow $C_{n,m}$ contains $N_{n,m}$ flows, and the flow set of $C_{n,m}$ is represented by $\mathbb{F}_{n,m} = \{F_{n,m,k}, 1 \leq k \leq N_{n,m}\}$. The k -th flow in $C_{n,m}$ is further defined as $f_{n,m,k} = \langle s_{n,m,k}, u_{n,m,k}, d_{n,m,k} \rangle$, where $s_{n,m,k}$, $u_{n,m,k}$, and $d_{n,m,k}$ respectively represent the source node, destination node, and data volume of flow $f_{n,m,k}$. It is assumed that job information, including source nodes, destination nodes, data volumes of all coflows, and arrival times and deadlines of jobs, can be obtained once the job arrives. Note that a job's arrival time only represents the arrival time of its first coflow, and the arrival times of subsequent coflows depend on the completion times of their upstream coflows. We assume that a coflow being transmitted cannot be preempted by other coflows, and the residual bandwidth information for each link is available to the central job scheduler whenever needed. The notations to be used are listed in Table 1.

In this paper we focus on the *starts-after* type coflow dependency, where the downstream coflow can not start before the upstream coflow ends, and a computer stage exists between them [22]. To illustrate the relationship between multi-stage jobs and coflows, a DAG-based job model from [29] is shown in Figure 1; it includes six computation stages and five communication stages. The five communication stages can be also referred to as five different coflows: denoted by C_1, C_2, C_3, C_4 , and C_5 , respectively. The DAG can visually show the data dependencies between adjacent coflows. For example, in Figure 1, T_3 cannot start before both C_1 and C_2 have ended, and C_3 begins after T_3 ends. The job completion

time is from the beginning of T_1 , T_2 , and T_4 to the end of T_6 . Further, the durations of all computing stages are assumed to be identical.

Table 1. Notations of variable and constants.

Symbol	Definition
\mathbb{V}	Node set
\mathbb{E}	Link set
B_e	Bandwidth of link e
\mathbb{J}	Job set
N	Number of jobs
\mathbb{C}_n	Coflow set of job J_n
N_n	Number of coflows in job J_n
$\mathbb{F}_{n,m}$	Flow set of coflow $C_{n,m}$
$N_{n,m}$	Number of flows in coflow $C_{n,m}$
$s_{n,m,k}$	Source node of flow $f_{n,m,k}$
$u_{n,m,k}$	Destination node of flow $f_{n,m,k}$
$d_{n,m,k}$	Data volume of flow $f_{n,m,k}$
r_n	Arrival time of job J_n
D_n	Acceptable longest duration of job J_n
T_n	Completion time of job J_n
$T_{n,m}$	Completion time of coflow $C_{n,m}$
$T_{n,m,k}$	Completion time of flow $f_{n,m,k}$
$b_{n,m,k}(t)$	bandwidth of flow $f_{n,m,k}$ at time t
G_n	Number of remaining coflow stages of job J_n
$R_{n,m}$	Relative remaining time of coflow $C_{n,m}$
$P_{n,m,k}^i$	i -th candidate path of $f_{n,m,k}$

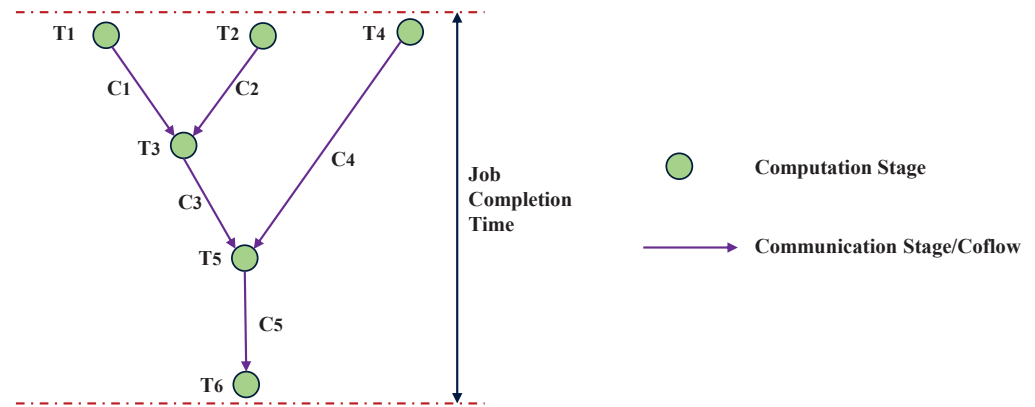


Figure 1. A job example represented by the DAG model; the job consists of multiple computation stages and communication stages.

3. Deadline-Guaranteed Job Number Maximization Problem Formulation

3.1. Motivating Example of Joint Flow Scheduling and Routing

In this subsection, a simple motivating example is given to demonstrate how combining flow scheduling and routing can improve the JCT performance compared with isolate coflow scheduling or routing.

In the two-level Spine-Leaf DCN shown in Figure 2, two jobs are deployed. Assume job J_1 contains two coflows $C_{1,1}$ and $C_{1,2}$, and job J_2 contains coflow $C_{2,1}$. Coflows $C_{1,1}$, $C_{1,2}$, and $C_{2,1}$ are respectively represented by an orange solid line, an orange dotted line, and a blue solid line. For simplicity, we assume each coflow only contains one flow, and the sizes of $C_{1,1}$, $C_{1,2}$, and $C_{2,1}$ are, respectively, 10 MB, 50 MB, and 30 Mb. The link bandwidths are all 10 Mbps. We also assume $C_{1,1}$ and $C_{2,1}$ are sent from S_1 to S_5 , and $C_{1,2}$ is sent from S_2 to S_6 .

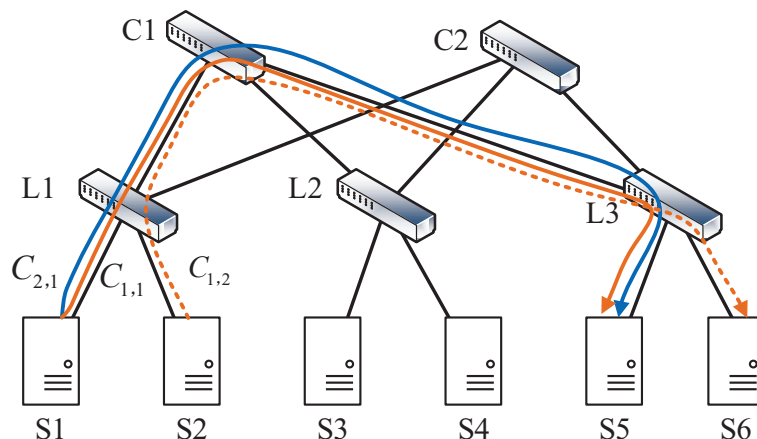


Figure 2. A routing case from ECMP containing coflows $C_{1,1}$, $C_{1,2}$, and $C_{2,1}$.

With a random routing policy such as equal-cost multipath (ECMP), one possible routing result is shown in Figure 2. $C_{1,1}$ and $C_{2,1}$ are transmitted via path $S_1 - L_1 - C_1 - L_3 - S_5$, and $C_{1,2}$ passes through $S_2 - L_1 - C_1 - L_3 - S_6$. In this case, the completion times of J_1 and J_2 under the Smallest Coflow First (SCF) strategy [18] and the Job Completion Time Aware (JCTA) strategy [25] are shown in Figure 3a,b, respectively. In Figure 3a, the completion times of two jobs under SCF are respectively 9 s and 4 s, while the JCTs under JCTA are respectively 9 s and 3 s in Figure 3b. Though the JCT of J_2 remains unchanged, the JCT of J_1 can be optimized by JCTA.

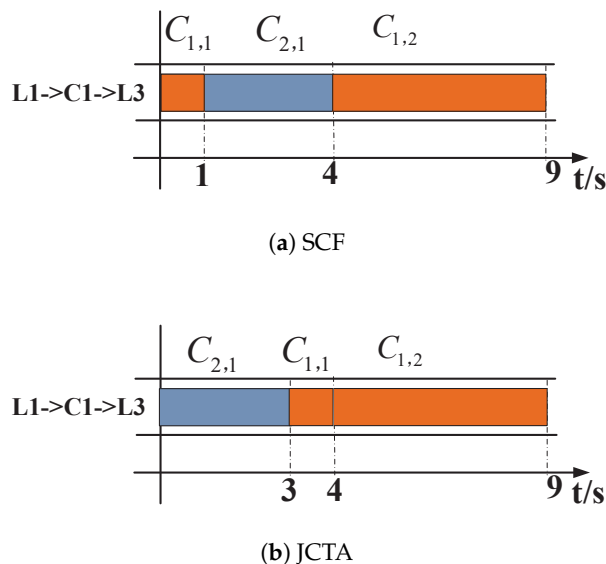


Figure 3. Scheduling results of SCF and JCTA strategies with the routing case above. (a) Completion times of J_1 and J_2 under the Smallest Coflow First (SCF) strategy. (b) Completion times of J_1 and J_2 under the Job Completion Time Aware (JCTA) strategy.

By integrating routing and flow scheduling, an optimized routing solution is shown in Figure 4, where $C_{1,2}$ are allocated to path $S_2 - L_1 - C_2 - L_3 - S_6$. The corresponding scheduling results of SCF and JCTA are also shown in Figure 5a,b. In Figure 5a, the completion times of two jobs under SCF are respectively 5 s and 4 s, while the JCTs under JCTA are respectively 5 s and 3 s. Similarly, the JCT of J_1 can be reduced by JCTA.

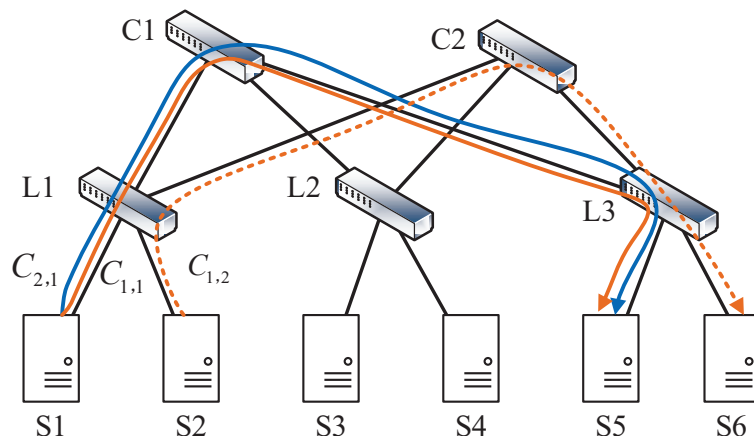
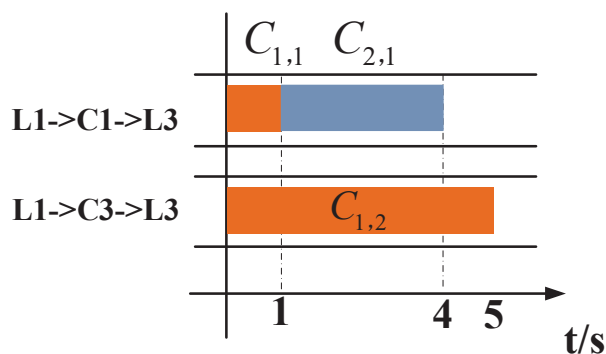
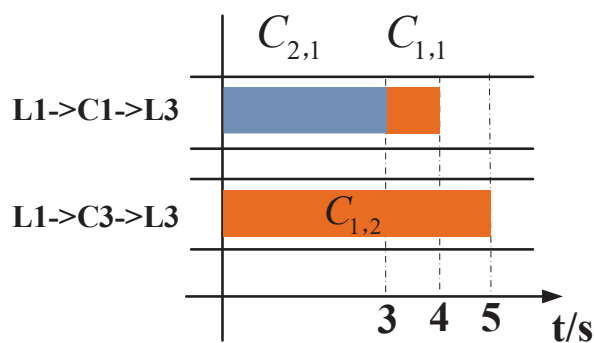


Figure 4. An improved routing case containing coflows $C_{1,1}$, $C_{1,2}$, and $C_{2,1}$.



(a) SCF



(b) JCTA

Figure 5. Scheduling results of SCF and JCTA strategies with the improved routing case above. (a) Completion times of J_1 and J_2 under the SCF strategy. (b) Completion times of J_1 and J_2 under the JCTA strategy.

According to the results above, the average JCT can be optimized by properly scheduling flows in the time domain after routing is determined. As shown by comparing Figure 3a,b, the SCF strategy determines the transmission sequence according to the coflow size but does not consider the coflow dependency or whether multiple coflows belong to the same job; the JCTA strategy pays more attention to the coflow dependency within the same job, which can contribute to reducing some jobs' completion times.

Moreover, by comparing the results under the same scheduling policy but different routing paths, such as Figures 3b and 5b, the JCT's performance can be further improved by introducing the path-choosing dimension. The random routing policy may result in load unbalancing, leaving some links utilized inefficiently. As shown in Figure 2, three coflows all pass through links $L_1 - C_1$ and $C_1 - L_3$, making the two links the bottleneck for job processing while an idle candidate path still exists. Therefore, to guarantee meeting as many job deadlines as possible, one should properly design joint flow scheduling and routing policies rather than performing isolated optimization of flow scheduling or routing.

3.2. Deadline-Guaranteed Job Number Maximization Optimization Problem

We aim to maximizing the number of jobs for which the deadlines have been met, i.e.,

$$\max \sum_{n=1}^N y_n. \quad (1)$$

The binary variable y_n indicates whether job J_n can be completed before its deadline D_n , denoted by

$$y_n = \begin{cases} 1, & \text{if } T_n \leq r_n + D_n \\ 0, & \text{if } T_n > r_n + D_n. \end{cases} \quad (2)$$

Here, r_n represents the arrival time of job J_n . The completion times of job J_n , coflow $C_{n,m}$, and flow $f_{n,m,k}$ are respectively denoted by T_n , $T_{n,m}$, and $T_{n,m,k}$. Since a job's completion time depends on the completion time of its last coflow, and a coflow's completion time depends on the completion time of its last flow, we have the following two completion time constraints:

$$T_n = \max_{C_{n,m} \in \mathbb{C}_n} T_{n,m}, \forall n \quad (3)$$

$$T_{n,m} = \max_{f_{n,m,k} \in \mathbb{F}_{n,m}} T_{n,m,k}, \forall C_{n,m} \quad (4)$$

Variable $b_{n,m,k}(t)$ represents the transmission rate of flow $f_{n,m,k}$ at time t , and thus the traffic volume constraint for flow $f_{n,m,k}$ can be written as

$$\int_{r_n}^{T_{n,m,k}} b_{n,m,k}(t) dt = d_{n,m,k}, \forall f_{n,m,k}. \quad (5)$$

The *starts-after* dependency between the m -th and the m' -th coflow of job J_n is denoted by $C_{n,m'} > C_{n,m}$, where $C_{n,m}$ begins after $C_{n,m'}$ ends. The precedence constraint in (6) ensures that $C_{n,m}$ will not start during the transmission of $C_{n,m'}$.

$$\int_{r_n}^{T_{n,m'}} b_{n,m,k}(t) dt = 0, \forall C_{n,m'} > C_{n,m}, \forall f_{n,m,k} \in \mathbb{F}_{n,m}. \quad (6)$$

The binary variable $x_e^{n,m,k}$ represents whether flow $f_{n,m,k}$ will pass through link e , so there are flow conservation constraints [39] as (7) and (8), where $out(v)$ and $in(v)$ represent the set of outgoing links from node v and the set of incoming links to node v , respectively.

$$\sum_{e \in out(v)} x_e^{n,m,k} - \sum_{e \in in(v)} x_e^{n,m,k} = 0, \forall f_{n,m,k}, \forall v \notin \{s_{n,m,k}, u_{n,m,k}\} \quad (7)$$

$$\sum_{e \in out(s_{n,m,k})} x_e^{n,m,k} - \sum_{e \in in(s_{n,m,k})} x_e^{n,m,k} = 1, \forall f_{n,m,k} \quad (8)$$

To guarantee that the accumulated rate of all flows allocated to link e will not exceed the bandwidth, we have the bandwidth constraint as

$$\sum_n \sum_m \sum_k x_e^{n,m,k} b_{n,m,k}(t) \leq B_e, \forall e, \forall t \in [\Gamma_0, \Gamma_1], \quad (9)$$

where $\Gamma_0 = \min_n r_n$, $\Gamma_1 = \min_n (r_n + D_n)$.

Since the transmission of flow $f_{n,m,k}$ should be between the arrival time and the deadline of job J_n , we have two rate constraints as (10) and (11).

$$b_{n,m,k}(t) = 0, \forall f_{n,m,k}, \forall t \in [\Gamma_0, r_n) \cup [D_n, \Gamma_1] \quad (10)$$

$$b_{n,m,k}(t) \geq 0, \forall f_{n,m,k}, \forall t \in [r_n, \Gamma_1] \quad (11)$$

Thus, the multi-job joint flow scheduling and routing problem in a DCN can be formulated as the non-linear optimization problem **P** in (12).

$$\begin{aligned} \mathbf{P}: \quad & \max_{x_e^{n,m,k}, b_{n,m,k}(t)} \sum_{n=1}^N y_n \\ \text{s.t.} \quad & x_e^{n,m,k} \in \{0, 1\}, \forall e, \forall f_{n,m,k} \\ & \text{Constraint(2) - (11)} \end{aligned} \quad (12)$$

The joint scheduling and routing problem for coflows has been proved to be NP-hard [32], while **P** is more complicated due to the nonlinear constraints and binary variables. Therefore, we decompose **P** and design an alternative heuristic suboptimal solution.

4. Two-Stage Joint Flow Scheduling and Routing

Due to its intractability, we decompose **P** into two decoupled sub-problems: the inter-coflow scheduling problem and the intra-coflow scheduling problem, and we develop a two-stage JFSR scheme. In JFSR, the first stage performs inter-coflow scheduling based on the relative remaining time (RRT) criterion and determines the target coflow to be further processed in the second stage. The second stage aims to figure out whether the target coflow can be scheduled immediately by arranging the routing path and transmission rate for each flow in the target coflow with the proposed Iterative Coflow Scheduling and Routing (ICSR) algorithm. For simplicity, we made three assumptions. First, once a coflow has been scheduled, the routing paths and transmission sequences of its flows cannot be preempted by any other coflow that is scheduled later. Second, the bandwidth allocated to each flow remains constant during its transmission. Third, each flow is unsplitable, i.e., it is not allowed to be divided into several segments and transmitted via different paths.

4.1. The Smallest Relative Remaining Time First Criterion

In the first stage of JFSR, the key is to determine the target coflow to be scheduled at time t_0 . In summary, there are three times when coflows will be checked for possible scheduling. The first when a new job arrives, and thus the system immediately investigates whether its first-stage coflows do not have any *starts-after*-type dependencies from upstream coflows and can thus be scheduled via the following ICSR algorithm in the second stage. If granted, a first-stage coflow from a newly arrived job can be transmitted at once; if denied due to limited bandwidth, the first-stage coflow will be put on the waiting list to wait for the next scheduling chance.

The second kind of schedule timing is when a coflow from a previously arrived job is ready, i.e., when all of its upstream computation stages are finished. Similarly, the system checks whether this coflow can begin its transmission or should go on the waiting list.

The third kind of schedule timing is when a coflow from a previously arrived job is finished and some bandwidth resource is released. In this case, the system investi-

gates whether any queuing coflow on the waiting list can have a chance. In this case, the scheduling priority of a queuing coflow is determined by the SRRTF criterion. If coflow $C_{n,m}$ is on the waiting list, job J_n has G_n unscheduled coflow stages at time t_0 , and coflow $C_{n,m}$ belongs to the g -th stage, then the RRT of coflow $C_{n,m}$ at time t_0 is defined as $R_{n,m} = g(r_n + D_n - t_0)/G_n$. For consistency, the RRT values of ready-to-go coflows from the first two kinds of timings are all set as zero. As a result, in all the three kinds of timings, ready-to-go coflows and queuing coflows all update their RRT values, and the coflow with the minimum RRT is chosen as the most urgent one and is handled by the second stage.

4.2. The Iterative Coflow Scheduling and Routing Algorithm

In the second stage, we focus on joint scheduling and routing for the single coflow chosen from the first stage. If coflow $C_{n,m}$ is chosen, problem **P** is simplified into **P1**, which aims to minimize the CCT of $C_{n,m}$. In **P1**, (14)–(17) respectively represent the binary variable constraint, rate variable constraint, traffic volume constraint, and bandwidth constraint, while (18) and (19) denote the flow conservation constraints.

$$\mathbf{P1} : \min_{b_{n,m,k}, x_e^{n,m,k}} T_{n,m} \quad (13)$$

$$s.t. x_e^{n,m,k} \in \{0, 1\}, \forall e, \forall f_{n,m,k} \in \mathbb{F}_{n,m} \quad (14)$$

$$b_{n,m,k} \geq 0, \forall f_{n,m,k} \in \mathbb{F}_{n,m} \quad (15)$$

$$T_{n,m} \geq \frac{d_{n,m,k}}{b_{n,m,k}}, \quad \forall f_{n,m,k} \in \mathbb{F}_{n,m} \quad (16)$$

$$\sum_{f_{n,m,k} \in \mathbb{F}_{n,m}} x_e^{n,m,k} b_{n,m,k} \leq B_e, \forall e \quad (17)$$

$$\sum_{e \in \text{out}(v)} x_e^{n,m,k} = \sum_{e \in \text{in}(v)} x_e^{n,m,k}, \forall f_{n,m,k} \in \mathbb{F}_{n,m}, \forall v \notin \{s_{n,m,k}, u_{n,m,k}\} \quad (18)$$

$$\sum_{e \in \text{out}(s_{n,m,k})} x_e^{n,m,k} - \sum_{e \in \text{in}(s_{n,m,k})} x_e^{n,m,k} = 1, \forall f_{n,m,k} \in \mathbb{F}_{n,m} \quad (19)$$

In fact, **P1** is an integer multi-commodity flow problem, which has been proved to be NP-hard [32]. Therefore, based on the alternating optimization principle, an Iterative Coflow Scheduling and Routing (ICSR) algorithm is proposed to alternately update the bandwidth and path allocation for the coflow being scheduled. The basic idea of ICSR is to fix one of the two kinds of variables—the bandwidth allocation variable $b_{n,m,k}$ or the path allocation variable $x_e^{n,m,k}$ —in turns and solve for the other one.

At first, the flow rate $b_{n,m,k}$ is initialized to obtain the candidate paths. For flow $f_{n,m,k}$, its initial rate is set as $b_{n,m,k}(0) = d_{n,m,k}/(r_n + D_n - t_0)$. By treating the bandwidth allocation variable $b_{n,m,k}$ as constant and relaxing the binary constraint of the path allocation $x_e^{n,m,k}$, **P1** is simplified into the feasibility problem **P2**. **P2** is a linear programming (LP) problem and can be efficiently solved. Though the object of **P2** is a constant, a feasible solution of **P2** is a group of routing paths for which the available bandwidths can satisfy the bandwidth constraints. When the available bandwidth is limited, it is possible that the value of $x_e^{n,m,k}$ obtained from **P2** is a fraction. In this case, flow $f_{n,m,k}$ can be split into multiple sub-flows, which are transmitted through different paths. Since each flow is unsplitable in our fundamental assumption, at the end of the ICSR algorithm, the values of $x_e^{n,m,k}$ should be recovered to be binary.

During the first iteration loop, it is possible that **P2** has no feasible solutions. In this case, there does not exist a candidate group of routing paths with adequate available bandwidth to accommodate coflow $C_{n,m}$. Thus, the ICSR algorithm ends and coflow $C_{n,m}$ is put on the waiting list. When a previously scheduled coflow is finished and the bandwidth

is released, all coflows on the waiting list update their RRT values and wait to be processed by the two stages of JFSR again.

$$\begin{aligned}
 \mathbf{P2} : \min & \quad 0 \\
 & \quad x_e^{n,m,k} \\
 \text{s.t.} & \quad 0 \leq x_e^{n,m,k} \leq 1, \forall e, \forall f_{n,m,k} \in \mathbb{F}_{n,m} \\
 & \quad \text{Constraint(17) - (19)}
 \end{aligned} \tag{20}$$

With the feasible solutions of $x_e^{n,m,k}$ obtained from **P2**, we calculate the transmission rate $b_{n,m,k}$. By treating $x_e^{n,m,k}$ as constant and introducing the auxiliary variable $a = 1/T_{n,m}$, **P1** is simplified and reformulated into the LP problem **P3**. The resulting values of $b_{n,m,k}$ will be the input of **P2** in the next iteration.

$$\begin{aligned}
 \mathbf{P3} : \max & \quad a \\
 & \quad b_{n,m,k} \\
 \text{s.t.} & \quad b_{n,m,k} \geq 0, \forall f_{n,m,k} \in \mathbb{F}_{n,m} \\
 & \quad b_{n,m,k} \leq d_{n,m,k}a, \forall f_{n,m,k} \in \mathbb{F}_{n,m} \\
 & \quad \sum_{f_{n,m,k} \in \mathbb{F}_{n,m}} x_e^{n,m,k} b_{n,m,k} \leq B_e, \forall e
 \end{aligned} \tag{21}$$

The pseudocode of the ICSR algorithm is given in Algorithm 1. At first, the path allocation variable $x_e^{n,m,k}$ and the rate allocation variable $b_{n,m,k}$ are alternately optimized according to **P2** and **P3**, respectively. After enough iterations, the resulting CCT may be satisfying, but there is a high probability that the values of $x_e^{n,m,k}$ are not binary. To ensure $x_e^{n,m,k}$ is binary, which guarantees each flow only goes through one path, for flow $f_{n,m,k}$ we check the value of $w_{n,m,k}^i = \min_{e \in P_{n,m,k}^i} x_e^{n,m,k}$ among all candidate paths, where $P_{n,m,k}^i$ represents the i -th candidate path of $f_{n,m,k}$. The candidate path with the maximum value of $w_{n,m,k}^i$ is chosen as the final transmission path, denoted by $P_{n,m,k}^*$. Thereafter, the final transmission rate of flow $f_{n,m,k}$, denoted by $b_{n,m,k}^*$ and the corresponding CCT of $C_{n,m}$, denoted by $T_{n,m}^*$, are determined by solving **P3** again with the recovered binary $x_e^{n,m,k}$. Thus, the joint scheduling for the finishing and routing of coflow $C_{n,m}$ is finished. As time goes by, the system continues to repeat the two steps of the JSFR scheme.

Algorithm 1 Iterative Coflow Scheduling and Routing Algorithm

Require: The prior information of coflow $C_{n,m}$.

Ensure: The CCT, path, and rate allocation of $C_{n,m}$.

- 1: Initialization: Set iteration number $n = 0$, $b_{n,m,k}(0) = d_{n,m,k} / (r_n + D_n - t_0)$ for all flows in $C_{n,m}$.
 - 2: **if** **P2** is infeasible **then**
 - 3: Put $C_{n,m}$ on the waiting list.
 - 4: **else**
 - 5: **repeat**
 - 6: 1) Update $x_e^{n,m,k}(n+1)$ for all flows in $C_{n,m}$ and all links by solving **P2** with known $b_{n,m,k}(n)$.
 - 7: 2) Update $b_{n,m,k}(n+1)$ for all flows in $C_{n,m}$ by solving **P3** with known $x_e^{n,m,k}(n+1)$, and $n \leftarrow n+1$.
 - 8: **until** the predetermined iteration number.
 - 9: For flow $f_{n,m,k}$, choose the final path $P_{n,m,k}^* = \arg \max_{e \in P_{n,m,k}^i} \min x_e^{n,m,k}$, and set $x_e^{n,m,k} = 1, \forall e \in P_{n,m,k}^*$ and $x_e^{n,m,k} = 0, \forall e \notin P_{n,m,k}^*$.
 - 10: Calculate the final rate $b_{n,m,k}^*$ and CCT $T_{n,m}^*$ by solving **P3** with the final $x_e^{n,m,k}$.
 - 11: **end if**
-

5. Simulation Results

In this section, the validity of the proposed JFSR scheme is confirmed by Monte Carlo simulation. The Fat-Tree topology with $k = 4$ is adopted as the DCN topology [40], and we utilize Pulp as the LP problem solver [41]. Job arrivals follow the Poisson process with an average arrival interval λ . The longest acceptable job duration, the number of coflows in each job, and the number of flows in each coflow all obey uniform distribution. The data size of flow $f_{n,m,k}$ follows a Gaussian distribution as $d_{n,m,k} \sim N(100, 30)$ Mb. The link bandwidth B_e is set as 10 Gbps.

The performance of three scheduling schemes were simulated and compared, i.e., the Baseline scheme, the Scheduling-Only scheme, and the proposed JFSR scheme. The Baseline scheme adopted the ECMP routing strategy, shared the bandwidth fairly, and scheduled coflows based on the SCF policy; the Scheduling-Only scheme also adopted ECMP and fair bandwidth sharing but scheduled coflows according the SRRTF criterion.

At first, we examine the convergence property of ICSR. The iterative CCT values in one coflow realization are shown in Figure 6. There are 130 flows in this coflow, and its RRT is set as 1 s. As shown in Figure 6, the CCT gradually approaches the minimal value. It should be noticed that this smooth convergence is only guaranteed in the iteration steps of ICSR. After its last iteration step, the ICSR algorithm recovers the binary variable $x_e^{n,m,k}$, which may not be optimal for the relaxed version of P1. However, the low-complexity iterative searching of ICSR allows for the performance of more iterative steps and may capture a satisfactory solution within the limited decision time.

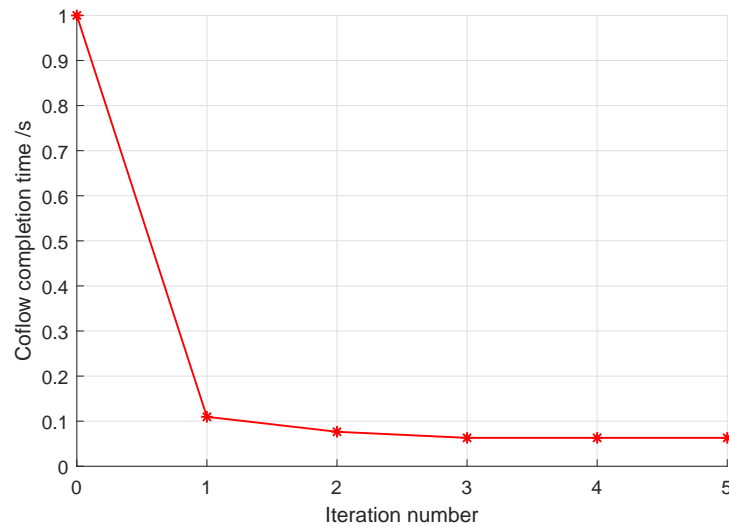


Figure 6. Convergence performance of ICSR for a single coflow instance.

The performance metric used to measure the deadline-guarantee capability of scheduling schemes in the simulation is the average normalized number of jobs for which the deadlines have been met, which is defined as the mean of the ratios of the number of jobs for which the deadlines have been met to the total number of all jobs in different snapshots with the same simulation parameters. The average normalized numbers of jobs for which the deadlines have been met with different flow numbers in each coflow from three scheduling schemes are compared in Figure 7. The average job arrival interval is $\lambda = 3s$, the longest acceptable job duration D_n is uniformly distributed in the interval $[10s, 50s]$, and the coflow number of each job is subject to a discrete uniform distribution in the interval $[10, 20]$. The flow number in each coflow is also subject to a discrete uniform distribution, and the end points of the distribution interval increase with a step size of 20 until $[180, 200]$ in different simulation cases. As shown in Figure 7, when the flow number in each coflow is below 20, the average normalized numbers of jobs for which the deadlines have been met from the three scheduling schemes are close to one. In this case, the flow load is relatively light

and almost all jobs' deadlines can be met. As the flow number in each coflow gradually increases, the proposed JFSR scheme achieves significantly better performance than the other two scheduling schemes, while the Scheduling-Only scheme also outperforms the Baseline scheme. Therefore, integrating optimizing the routing path into coflow scheduling can efficiently guarantee the deadline-met performance for multi-stage time-sensitive jobs. The Baseline and Scheduling-Only schemes both adopt ECMP as the random routing strategy and use fair bandwidth sharing, and the hash-table-based path selection may arrange for too many flows to share the bandwidth of the same bottlenecked link. As a result, the limited bandwidth allocated to every flow passing through the bottlenecked link will remarkably increase the completion times of these unlucky flows, which will further raise the completion times of the coflows as well the jobs that contain these delayed flows. As the flow number in each coflow increases, the effect becomes more and more pronounced. At the same time, the proposed JFSR scheme can optimize the routing paths and the bandwidth allocations for the flows of each coflow via the designed ICSR algorithm. Thus, the completion time of each coflow can be reduced, which will further contribute to reducing the JCT. Therefore, the increase in each coflow's flow number has a smaller impact on the curve of JFSR in Figure 7. Further, the performance difference between the Baseline and Scheduling-Only schemes can be attributed to their inter-coflow scheduling policies. The Baseline scheme determines the scheduling priorities of coflows according to their data volumes, i.e., via the SCF policy, but does not involve the deadline information. The Scheduling-Only scheme considers prior deadline information of jobs as well as the RRT of coflows and thus can guarantee more jobs' deadlines are met than the Baseline scheme.

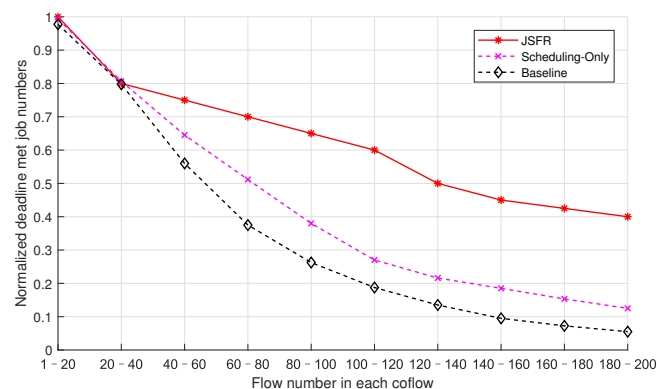


Figure 7. The average normalized numbers of jobs for which the deadlines have been met with the increase in the flow number in each coflow under different scheduling policies.

The influence of the coflow number in each job on the average normalized number of jobs for which the deadlines have been met from three scheduling schemes is illustrated in Figure 8. The settings of λ and D_n are identical as those in Figure 7, and the flow number in each coflow is subject to a discrete uniform distribution on the interval $[80, 100]$. The coflow number in each job is also subject to a discrete uniform distribution, and the end points of the distribution interval increase from $[10, 12]$ to $[28, 30]$ in different simulation cases. As the coflow number in each job increases, the average normalized numbers of punctual jobs from the three scheduling schemes all gradually decrease in Figure 8. Similar to Figure 7, the proposed JFSR scheme can still guarantee meeting many more job deadlines than the other two scheduling schemes, and the Scheduling-Only scheme is also superior to the Baseline scheme in Figure 8. With the increase in the coflow number in each job, heavier flow loads are deployed into the network, and the *starts-after*-type coflow dependency relationships becomes more and more complicated. The JFSR scheme can both optimize the completion of each coflow via the ICSR algorithm and arrange for a more appropriate dispatching sequence for coflows via the SRRTF criterion. These two factors ensure the JFSR scheme can maintain better performance at guaranteeing meeting job deadlines than the other two scheduling schemes under different flow loads.

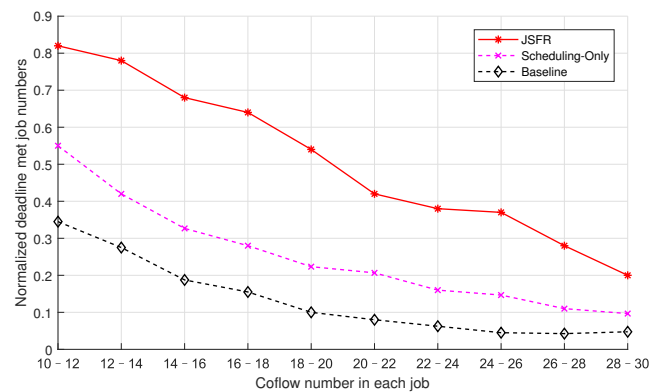


Figure 8. The average normalized numbers of jobs for which the deadlines have been met with the increase in the coflow number in each job under different scheduling policies.

6. Conclusions

In this paper we focused on how to guarantee meeting the deadlines of time-sensitive IoT jobs by jointly considering flow scheduling and routing in DCNs. First, the multi-job joint flow scheduling and routing problem was formulated as a non-linear optimization problem with the object of maximizing the number of jobs for which the deadlines have been met. Then, we decomposed the problem into two inter-coflow scheduling and intra-coflow scheduling sub-problems to reduce the complexity of solving the problem. In the inter-coflow scheduling subproblem, coflows are ordered by their relative remaining times; in the intra-coflow scheduling subproblem, an iterative coflow scheduling and routing (ICSR) algorithm was designed to determine the transmission rates and routing paths for the scheduled coflow. Simulation results verified the proposed two-stage joint flow scheduling and routing can efficiently improve the number of jobs for which the deadlines have been met in DCNs. However, the proposed JFSR scheme is based on some ideal assumptions, and these assumptions are less practical in real data center scenarios. For example, we assume that complete job information, including the source nodes, destination nodes, and data volumes of all coflows in each job, can be obtained once the job arrives, and the residual bandwidth information for each link is available whenever needed. In fact, in practical scenarios, the detailed coflow information is not known or is incomplete, and the link's residual bandwidth information is usually outdated. Even so, we think the basic principle of the proposed JFSR scheme is still applicable for designing online information-agnostic coflow scheduling schemes and may be helpful for researchers or engineers in this field. For example, incomplete coflow information can be replaced by statistical information derived from massive historical records of the same kind of cluster computing jobs, and link residual bandwidth may be obtained by advanced forecasting algorithms. In the future, these realistic constraints should be considered for designing a more practical flow scheduling scheme for cloud data centers, which is attractive for researchers or engineers developing software programs to configure and manage physical or virtual network resources for cloud data center infrastructure providers or service providers.

Author Contributions: Conceptualization and methodology, L.S. and W.J.; Formal analysis, validation, and writing—original draft preparation, L.S. and H.M.; software and writing—review and editing, X.L. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the National Natural Science Foundation of China (62101415).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to thank the Editor-in-Chief, Editor, and anonymous Reviewers for their valuable reviews.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Zhao, W.; Liu, J.; Guo, H.; Hara, T. ETC-IoT: Edge-Node-Assisted Transmitting for the Cloud-Centric Internet of Things. *IEEE Netw.* **2018**, *32*, 101–107. [[CrossRef](#)]
2. Pan, J.; McElhannon, J. Future Edge Cloud and Edge Computing for Internet of Things Applications. *IEEE Internet Things J.* **2018**, *5*, 439–449. [[CrossRef](#)]
3. Bozkurt, I.; Aguirre, A.; Chandrasekaran, B. Why is the Internet so Slow? In Proceedings of the International Conference on Passive and Active Network Measurement, Sydney, NSW, Australia, 30–31 March 2017; Volume 10176; pp. 173–187. [[CrossRef](#)]
4. Liu, B.; Liu, C.; Peng, M. Resource Allocation for Energy-Efficient MEC in NOMA-Enabled Massive IoT Networks. *IEEE J. Sel. Areas Commun.* **2021**, *39*, 1015–1027. [[CrossRef](#)]
5. Zhang, J.; Li, T.; Ying, Z.; Ma, J. Trust-Based Secure Multi-Cloud Collaboration Framework in Cloud-Fog-Assisted IoT. *IEEE Trans. Cloud Comput.* **2023**, *11*, 1546–1561. [[CrossRef](#)]
6. Giroire, F.; Huin, N.; Tomassilli, A.; Pérennes, S. When network matters: Data center scheduling with network tasks. In Proceedings of the IEEE INFOCOM 2019-IEEE Conference on Computer Communications, Paris, France, 29 April–2 May 2019; pp. 2278–2286.
7. Dean, J.; Ghemawat, S. MapReduce: simplified data processing on large clusters. *Commun. ACM* **2008**, *51*, 107–113. [[CrossRef](#)]
8. Zaharia, M.; Chowdhury, M.; Franklin, M.J.; Shenker, S.; Stoica, I. Spark: Cluster Computing with Working Sets. In Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, Boston, MA, USA, 22–25 June 2010; p. 10.
9. Chowdhury, M.; Zaharia, M.; Ma, J.; Jordan, M.I.; Stoica, I. Managing Data transfers in computer clusters with orchestra. *ACM Sigcomm Comput. Commun. Rev.* **2011**, *41*, 98–109. [[CrossRef](#)]
10. Hong, C.Y.; Caesar, M.; Godfrey, P.B. Finishing Flows Quickly with Preemptive Scheduling. In Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Helsinki, Finland, 13–17 August 2012; pp. 127–138.
11. Guo, Z.; Hui, S.; Xu, Y.; Chao, H.J. Dynamic flow scheduling for power-efficient data center networks. In Proceedings of the 2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), Beijing, China, 20–21 June 2016; pp. 1–10. [[CrossRef](#)]
12. Gopalakrishna, V.K.; Kaymak, Y.; Lin, C.B.; Rojas-Cessa, R. PEQ: Scheduling Time-Sensitive Data-Center Flows using Weighted Flow Sizes and Deadlines. In Proceedings of the 2020 IEEE 21st International Conference on High Performance Switching and Routing (HPSR), Newark, NJ, USA, 11–14 May 2020; pp. 1–6. [[CrossRef](#)]
13. Xu, Y.; Luo, H.; Ren, F. Is minimizing flow completion time the optimal way in meeting flow’s deadline in datacenter networks. *China Commun.* **2016**, *13*, 6–15. [[CrossRef](#)]
14. Ho, J.M.; Hsiu, P.C.; Chen, M.S. Deadline Flow Scheduling in Datacenters with Time-Varying Bandwidth Allocations. *IEEE Trans. Serv. Comput.* **2020**, *13*, 437–450. [[CrossRef](#)]
15. Chowdhury, M.; Stoica, I. Coflow: a networking abstraction for cluster applications. In Proceedings of the 11th ACM Workshop on Hot Topics in Networks, Association for Computing Machinery, Redmond, WA, USA, 29–30 October 2012; pp. 31–36. [[CrossRef](#)]
16. Qiu, Z.; Stein, C.; Zhong, Y. Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks. In Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures. Association for Computing Machinery, Portland, OR, USA, 13–15 June 2015; pp. 294–303. [[CrossRef](#)]
17. Chen, T.; Gao, X.; Chen, G. The features, hardware, and architectures of data center networks: A survey. *J. Parallel Distrib. Comput.* **2016**, *96*, 45–74. [[CrossRef](#)]
18. Chowdhury, M.; Zhong, Y.; Stoica, I. Efficient coflow scheduling with Varys. *ACM Sigcomm Comput. Commun. Rev.* **2014**, *44*, 443–454. [[CrossRef](#)]
19. Ma, S.; Jiang, J.; Li, B.; Li, B. Chronos: Meeting coflow deadlines in data center networks. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 22–27 May 2016; pp. 1–6. [[CrossRef](#)]
20. Luo, S.; Yu, H.; Li, L. Decentralized deadline-aware coflow scheduling for datacenter networks. In Proceedings of the 2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia, 23–27 May 2016; pp. 1–6. [[CrossRef](#)]
21. Luo, S.; Yu, H.; Zhao, Y.; Wang, S.; Yu, S.; Li, L. Towards Practical and Near-Optimal Coflow Scheduling for Data Center Networks. *IEEE Trans. Parallel Distrib. Syst.* **2016**, *27*, 3366–3380. [[CrossRef](#)]
22. Chowdhury, M.; Stoica, I. Efficient Coflow Scheduling without Prior Knowledge. *ACM SIGCOMM Comput. Commun. Rev.* **2015**, *45*, 393–406. [[CrossRef](#)]
23. Zhang, T.; Ren, F.; Shu, R.; Wang, B. Scheduling Coflows with Incomplete Information. In Proceedings of the 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS), Banff, AL, Canada, 4–6 June 2018; pp. 1–10. [[CrossRef](#)]
24. Wang, Z.; Zhang, H.; Shi, X.; Yin, X.; Li, Y.; Geng, H.; Wu, Q.; Liu, J. Efficient Scheduling of Weighted Coflows in Data Centers. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2003–2017. [[CrossRef](#)]
25. Liu, Y.; Li, W.; Li, K.; Qi, H.; Tao, X.; Chen, S. Scheduling Dependent Coflows with Guaranteed Job Completion Time. In Proceedings of the 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, 23–26 August 2016; pp. 2109–2115. [[CrossRef](#)]

26. Borjigin, W.; Ota, K.; Dong, M. Time-Saving First: Coflow Scheduling for Datacenter Networks. In Proceedings of the 2017 IEEE 86th Vehicular Technology Conference (VTC-Fall), Toronto, ON, Canada, 24–27 September 2017; pp. 1–5. [[CrossRef](#)]
27. Zhang, S.; Zhang, S.; Qian, Z.; Zhang, X.; Xiao, M.; Wu, J.; Ge, J.; Wang, X. Efficient scheduling for multi-stage coflows. *CCF Trans. Netw.* **2019**, *2*, 83–97. [[CrossRef](#)]
28. Tian, B.; Tian, C.; Wang, B.; Li, B.; He, Z.; Dai, H.; Liu, K.; Dou, W.; Chen, G. Scheduling dependent coflows to minimize the total weighted job completion time in datacenters. *Comput. Netw.* **2019**, *158*, 193–205. [[CrossRef](#)]
29. Wang, J.; Zhou, H.; Yang, H.; Laat, C.D.; Zhao, Z. Deadline-aware coflow scheduling in a DAG. In Proceedings of the 2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Hong Kong, 11–14 December 2017; pp. 341–346. [[CrossRef](#)]
30. Yuang, M.; Tien, P.L.; Ruan, W.Z.; Lin, T.C.; Wen, S.C.; Tseng, P.J.; Lin, C.C.; Chen, C.N.; Chen, C.T.; Luo, Y.A.; et al. OPTUNS: Optical intra-data center network architecture and prototype testbed for a 5G edge cloud [Invited]. *J. Opt. Commun. Netw.* **2020**, *12*, A28–A37. [[CrossRef](#)]
31. Zhu, H.; Liao, X.; de Laat, C.; Grosso, P. Joint flow routing-scheduling for energy efficient software defined data center networks A prototype of energy-aware network management platform. *J. Netw. Comput. Appl.* **2016**, *63*, 110–124. [[CrossRef](#)]
32. Zhao, Y.; Chen, K.; Bai, W.; Yu, M.; Tian, C.; Geng, Y.; Zhang, Y.; Li, D.; Wang, S. Rapiet: Integrating routing and scheduling for coflow-aware data center networks. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon, Hong Kong, 26 April–1 May 2015; pp. 424–432. [[CrossRef](#)]
33. Jiang, J.; Ma, S.; Li, B.; Li, B. Tailor: Trimming Coflow Completion Times in Datacenter Networks. In Proceedings of the 2016 25th International Conference on Computer Communication and Networks (ICCCN), Waikoloa, HI, USA, 1–4 August 2016; pp. 1–9. [[CrossRef](#)]
34. Li, Y.; Jiang, H.C.; Tan, H.; Zhang, C.; Lau, F. Efficient online coflow routing and scheduling. In Proceedings of the the 17th ACM International Symposium. Association for Computing Machinery, Trento, Italy, 12–16 December 2016; pp. 161–170. [[CrossRef](#)]
35. Tan, H.; Jiang, S.H.C.; Li, Y.; Li, X.Y.; Zhang, C.; Han, Z.; Lau, F.C.M. Joint Online Coflow Routing and Scheduling in Data Center Networks. *IEEE/ACM Trans. Netw.* **2019**, *27*, 1771–1786. [[CrossRef](#)]
36. Chen, Y.; Wu, J. Multi-Hop Coflow Routing and Scheduling in Data Centers. In Proceedings of the 2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA, 20–24 May 2018; pp. 1–6. [[CrossRef](#)]
37. Chen, Y.; Wu, J. Joint coflow routing and scheduling in leaf-spine data centers. *J. Parallel Distrib. Comput.* **2021**, *148*, 83–95. [[CrossRef](#)]
38. Zeng, Y.; Ye, B.; Tang, B.; Guo, S.; Qu, Z. Scheduling coflows of multi-stage jobs under network resource constraints. *Comput. Netw.* **2021**, *184*, 107686. [[CrossRef](#)]
39. Kai, H.; Hu, Z.; Luo, J.; Liu, X. RUSH: RoUting and Scheduling for Hybrid Data Center Networks. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), Kowloon, Hong Kong, 26 April–1 May 2015; pp. 415–423. [[CrossRef](#)]
40. Al-Fares, M.; Loukissas, A.; Vahdat, A. A Scalable, Commodity Data Center Network Architecture. In Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication. Association for Computing Machinery, Seattle, WA, USA, 17–22 August 2008; pp. 63–74. [[CrossRef](#)]
41. PuLP. Available online: <https://pypi.org/project/PuLP/> (accessed on 5 May 2005).

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.