Scheduling Mix-Coflows in Datacenter Networks

Renhai Xu, Wenxin Li, Keqiu Li, *Senior Member, IEEE*, Xiaobo Zhou, *Senior Member, IEEE*, and Heng Qi, *Member, IEEE*

Abstract*—Data-parallel applications generate a mix of coflows with and without deadlines. Deadline coflows are mission-critical and must be completed within deadlines, while the non-deadline coflows desire to be completed as soon as possible. Scheduling such mix-coflows is an important problem in modern datacenters. However, existing solutions only focus on one of the two types of coflows: they either solely concentrate on meeting the deadlines of deadline-aware coflows or reducing the coflow completion times (CCTs) of non-deadline coflows. In this article, we study the problem of optimizing deadline and non-deadline coflows simultaneously. To this end, we present a new optimization framework, *mixCoflow*, to schedule deadline coflows to minimize and balance their bandwidth footprint, such that non-deadline coflows can be scheduled as early as possible. Specifically, we develop the mathematical model and formulate the scheduling problem for deadline coflows as a lexicographical min-max integer linear programming (ILP) problem. Through rigorous theoretical analysis, this ILP problem has been proved to be equivalent to a linear programming (LP) problem that can be solved with standard LP solvers. By solving this LP, *mixCoflow* is able to balance the bandwidth footprint of deadline coflows while guaranteeing their deadlines. As a result, non-deadline coflows can be scheduled as soon as possible whenever they arrive. To demonstrate the effectiveness of our work, we have conducted extensive simulations based on a widely used Facebook data trace. The simulation results verify that *mixCoflow* can achieve significant improvement on the average CCT of non-deadline coflows, at no expense of increasing the deadline miss rates of deadline coflows, when compared to the state-of-art solutions.

***Index Terms*—Datacenter, deadline/non-deadline coflow, CCT, lexicographical optimization.**

I. Introduction

OVER the last decade, datacenter networks have witnessed an increasing wave of popularity of the data-parallel applications (e.g., Web search queries and MapReduce-like jobs) in dealing with the exponential growth of data [2], [3], [4], [5], [6], [7]. A common feature of these applications is that they generate a set of parallel flows to transfer the intermediate data between successive computation stages—known as *cowflows* [8]. A succeeding computation stage cannot start until all its required inputs are in place, leading to an *all-of-nothing* feature for a coflow: all flows must be completed before a coflow is considered to be completed [7], [9].

In modern datacenters, coflows can be roughly divided into two categories: *deadline coflows* and *non-deadline coflows*. *Deadline coflows* are often generated by online service triggered data-parallel jobs [8], [10], [11] or some mission-critical analytics jobs [12], [13]. For example, in an analytics job that monitors the user activity logs, the user of this job may want to obtain the top-k popular URLs by the number of clicks once every few seconds. Such stringent latency requirements will eventually enforce the underlying coflow to be completed within a deadline. Such deadline coflows are useful to the users if, and only if, they are completed within their deadlines. Otherwise, the user experience will be hurt, and this will, in turn, waste network bandwidth and incur revenue loss for the datacenter provider. On the other hand, *non-deadline coflows*, typically generated by cluster computing applications and data backups, have different performance requirements [7], [8], [9], [14]. More specifically, they impose no specific deadlines but generally desire to be completed as quickly as possible. When the two types of coflows coexist in a datacenter, an important problem is: how to schedule such a mix of coflows, with guaranteeing deadlines for deadline coflows and reducing CCTs for non-deadline coflows.

While recognizing the importance of such mix-coflow scheduling problem, no existing solutions [7], [9], [14], [15], [16], [17], [18], [19], [20], [21] are in place to optimize the deadline and non-deadline coflows simultaneously. The crux is that most of them only focus on optimizing one category of the coflows, which may hurt the performance of the other category of coflows. In other words, purely minimizing the CCTs of non-deadline coflows will cause high rates of deadline misses for the deadline coflows. Meanwhile, purely meeting deadlines of deadline coflows can arbitrarily prolong the CCTs of non-deadline coflows. It is worth noting that Varys [22] is perhaps the most related recent work that takes both deadline and non-deadline coflows into account. It separately designs two sets of strategies, i.e., SEBF (Smallest-Effective-Bottleneck-First) and MADD (Minimum-Allocation-for-Desired-Duration), to minimize the average CCT and the number of late coflow. However, Varys

Manuscript received March 16, 2020; revised July 5, 2020; accepted September 13, 2020. Date of publication September 29, 2020; date of current version June 10, 2021. This work was supported partly by the National Key R&D Program of China under Grant 2019YFB2102404; the NSFC under Grants 61772112; the Science Innovation Foundation of Dalian under Grant 2019J12GX037. A preliminary version of this paper was presented at the 26th IEEE/ACM International Symposium on Quality of Service (IWQoS) [1]. The associate editor coordinating the review of this article and approving it for publicationwas R. Pasquini. *(Corresponding author: Wenxin Li.)*

Renhai Xu, Keqiu Li, and Xiaobo Zhou are with the Tianjin Key Laboratory of Advanced Networking, College of Intelligence and Computing, Tianjin University, Tianjin 300350, China (e-mail: xurenhai@tju.edu.cn; keqiu@tju.edu.cn; xiaobo.zhou@tju.edu.cn).

Wenxin Li is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong 999077 (e-mail: toliwenxin@gmail.com).

Heng Qi is with the School of Computer Science and Technology, Dalian University of Technology, Dalian 116023, China (e-mail: hengqi@dlut.edu.cn).

Digital Object Identifier 10.1109/TNSM.2020.3027498

essentially considers the two types of coflows independently rather than jointly, even though Varys strategies are suited to both deadline/non-deadline coflows.

Bearing the above points in mind, one may wonder at this point that why not using Varys strategies to schedule the deadline and non-deadline coflows simultaneously. For example, one can schedule deadline coflows first with SEBF+MADD, and then similarly use SEBF+MADD to schedule non-deadline coflows with the residual network bandwidth. However, such a trivial combination is problematic and can hurt the CCTs of non-deadline coflows while incurring minor or no improvement for the deadline miss rate. The main reason is that Varys strategies are unaware of the bandwidth footprint of deadline coflows, resulting in *heterogeneous* residual bandwidth in different time and links. More precisely, some links may be congested and have little residual bandwidth, while some others may go underutilized. In such a case, Varys cannot fully utilize the residual bandwidth for scheduling the non-deadline coflows, thus hurting the CCTs of non-deadline coflows.

In this article, we study the mix-coflow scheduling problem, with the objective of meeting the deadlines of deadline coflows as well as reducing the CCTs of non-deadline coflows. To this end, we present *mixCoflow*, a new optimization framework that schedules the deadline coflows with *minimal* impact on the non-deadline coflows. More specifically, rather than directly considering one of the two types of coflows only, *mixCoflow* schedules deadline coflows first since they have higher priorities. But the thing is that when scheduling deadline coflows, *mixCoflow* attempts to minimize the impact on the non-deadline coflows by *minimizing the maximum bandwidth usage of deadline coflows across all time slots and all links*. We develop the mathematical model and formulate the deadline coflow scheduling problem as a lexicographical min-max integer linear programming (ILP) problem, which is inherently challenging to be resolved. Fortunately, after taking an in-depth investigation of the structure of the ILP problem, we observe that the original ILP problem meets the following two conditions: 1) a separable convex objective function and 2) a totally unimodular constraint matrix. These conditions guarantee that the original ILP problem can be transformed into a linear programming (LP) problem, by applying the λ-technique [23] and linear relaxation. It has been proved that the transformed LP problem can be guaranteed to have the same solution to the original ILP problem. Moreover, the transformed LP can be efficiently and quickly solved with standard LP solvers. After the deadline coflows are scheduled, the remaining bandwidth can be allocated to the non-deadline coflows by using any existing methods such as FIFO and SEBF+MADD.

The highlights of our original contributions in this article are summarized as follows:

- We focus on the problem of scheduling a mix of coflows with and without deadlines in a datacenter, to meet the deadlines of deadline coflows while reducing the CCTs of non-deadline coflows at the same time.
- We present *mixCoflow*, a new optimization framework that schedules deadline coflows with minimal impact on the CCTs of non-deadline coflows, by formulating and solving a lexicographical ILP problem of *minimizing the maximum bandwidth usage over time and links*.
- We conduct rigorous theoretical analysis to prove that the original ILP can be non-trivially transformed into an LP problem, which can be efficiently and quickly solved with standard LP solvers and returns exactly the same solution to the original ILP.
- We conduct large-scale simulations based on a widely adopted Facebook data trace, to verify the performance of our proposed *mixCoflow*. The simulation results demonstrate that *mixCoflow* can speed up the average CCT of non-deadline coflows by up to 9.35×, while incurring no increase on the deadline miss rate for deadline coflows.

The remainder of this article is organized as follows. In Section II, we elaborate the problem this article addressed. In Section III, we present the mathematical model and problem formulation. In Section IV, we show the design details of our *mixCoflow*. The extensive simulations are shown in Section V. We discuss current limitations of our work and relevant future research in Section VI. Finally, we discuss the related work in Section VII and conclude this article in Section VIII.

II. Problem Statement

In this article, the problem we are trying to solve is: how to schedule a mix of coflows with and without deadlines in a datacenter network, with the primary objective of meeting the deadlines of deadline coflows and reducing the CCTs of non-deadline coflows at the same time.

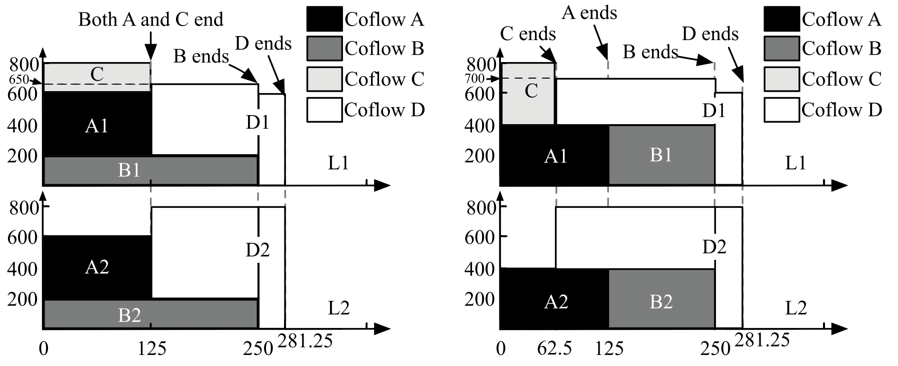
Scheduling such mix-coflows is inherently challenging because the performance requirements of the two types of coflows are conflicting with each other. More specifically, purely considering deadline coflows by aggressively using all available bandwidth to transmit deadline coflows, will increase the CCTs of non-deadline coflows. Even worse, purely considering non-deadline coflows will incur a high deadline miss rate for deadline coflows. This implies that we cannot complete one of the two types of coflows too aggressively or mildly. From this point of view, one may question that we can use Varys SEBF+MADD strategies to schedule the deadline coflows with the minimum possible bandwidth, such that the non-deadline coflows can get more residual bandwidth. Unfortunately, the residual bandwidth may turn out to be different across different time slots and different links. As such, non-deadline coflows cannot make full use of the residual bandwidth, as they usually have correlated data transmission across multiple links. Surprisingly, if we can minimize the maximum bandwidth usage across all time slots and all links after the deadline coflows are scheduled, the CCTs of non-deadline coflows can then be minimally impacted.

For a better intuition of our problem, we show a motivating example in Fig. 1. The example settings are shown in Fig. 1(a). There are two links (i.e., $L1$ and $L2$), with each link having a capacity of 800MB/s. We inject 4 coflows into this example, of which two are deadline coflows (i.e., A and B) and the other two are non-deadline coflows (i.e., C and D). Coflow A has

Coflows	Deadline	Flow information	
■ Coflow A	125 ms	A1 on L1	50 MB
		A2 on L2	50 MB
▨ Coflow B	250 ms	B1 on L1	50 MB
		B2 on L2	50 MB
░ Coflow C	N/A	C on L1	25 MB
□ Coflow D	N/A	D1 on L1	75 MB
		D2 on L2	100 MB
Both L1 and L2 have the capacity of 800MB/s			

N/A denotes the corresponding coflow is non-deadline coflow

(a) Example setting



(b) Varys SEBF+MADD scheme

(c) A better alternative

Fig. 1. A motivating example to show why balancing the bandwidth usage incurred by the deadline coflows can speed up the transmission of non-deadline coflows. (a) is the example setting. (b) When using Varys SEBF+MADD strategies to schedule both deadline/non-deadline coflows, the average CCT of the non-deadline coflows is 203.125ms. (c) In contrast, when scheduling the deadline coflows by minimizing and balancing the bandwidth usage across time and links, the average CCT of non-deadline coflows can then be reduced to 171.875ms.

two flows (i.e., $A1$ and $A2$) transmitted on links $L1$ and $L2$ respectively: the sizes and deadlines of both the two flows are 50MB and 125ms, respectively. Coflow B also has two flows (i.e., $B1$ and $B2$) with the sizes and the deadlines of both the two flows being 50MB and 250ms, respectively. Coflow C only has one flow transmitted over the link $L1$, with the size of 25MB. Coflow D has two flows $D1$ and $D2$ with the sizes of 75MB and 100MB respectively.

Before showing the detailed scheduling results, we first provide a brief primer on the state-of-the-art Varys solution [22]. Varys first prioritizes coflows with the SEBF heuristic, and then uses the MADD algorithm to *sequentially* allocate bandwidth for each coflow based on its priority. SEBF greedily schedules a coflow based on its bottleneck flow's completion time, while MADD allocates a coflow the least amount of bandwidth to match the completion time of the bottleneck flow that will take longest to finish or to attain the *maximum possible progress*. Formally speaking, defining R_i as the remaining bandwidth on link i and d_i as the data volume that a coflow needs to transfer on link i, the *maximum attainable progress* for this coflow can then be calculated as $\min_{i:d_i>0} R_i/d_i$. To achieve this maximum progress, MADD will allocate $d_i \min_{i:d_i>0} R_i/d_i$ amount of bandwidth on link i to this coflow. On the other hand, if a coflow has deadline t, then MADD will allocate d_i/t amount of bandwidth on link i to this coflow.

In this example, we first apply the SEBF+MADD strategies to schedule the deadline coflows A and B, as shown in Fig. 1(b). Under the SEBF heuristic, A will be prioritized before B. Since A has deadline 125ms, MADD will allocate 400MB/s amount of bandwidth to each of the flows of it from the very beginning to the deadline of this coflow. Similarly, each flow of B can get 200MB/s amount of bandwidth. After deadline coflows are scheduled, the residual bandwidth will be allocated to the non-deadline coflows (i.e., C and D) by using the SEBF+MADD strategies again. Following the policy of SEBF heuristic, we should schedule the non-deadline coflows with priority $C > D$. Because the priority of C is higher than that of D, C will exclusively use the entire link L_1, leaving the other idle link to D. Unfortunately, without $L1$, D cannot gain any progress because it needs to transfer data over $L1$. As a result, MADD allocates no bandwidth to D until the coflow C has been completed, and accordingly, the average of the two non-deadline coflows is 203.125 ms.

From Fig. 1(b), we can observe that the bandwidth usage caused by the deadline coflows is not well balanced across time and links. As a result, the non-deadline coflows may not have enough or more residual bandwidth, and hence their CCTs can be negatively impacted. Surprisingly, if we can minimize and balance the bandwidth usage of deadline coflows, the impact on the CCTs of non-deadline coflows can then be reduced. As shown in Fig. 1(c), when we schedule A during [0, 125]ms and schedule B in the range of [125, 250]ms, the resulting bandwidth usage can be balanced, and the deadlines of both A and B can be satisfied. After that, we similarly schedule the non-deadline coflows with the SEBF+MADD strategies, the CCTs of C and D are 62.5ms and 281.25ms, respectively. Accordingly, the average CCT of the non-deadline coflows can be reduced to 171.875ms.

III. Modeling and Problem Formulation

In this section, we develop a mathematical model to study the problem of minimizing and balancing the bandwidth footprint of deadline coflows, so as to minimize the impact on non-deadline coflows and thus accelerate non-deadline coflows.

A. Mathematical Model

In our analysis, we abstract the datacenter network as a non-blocking switch interconnecting all the machines [14], [22], [24], [25]. In other words, coflow scheduling and bandwidth competition only take place at the ingress/egress ports of this non-blocking switch, which corresponds to the incoming/outgoing links at each machine. Such a network abstraction is reasonable and has been widely used in many recent studies [14], [22], yet it is practical due to the recent full bisection bandwidth topologies [26], [27].

TABLE I
SYMBOLS AND DEFINITIONS

Symbol	Definition
$\mathcal{N}$	the set of servers/machines in a datacenter network
$\mathcal{T}$	the set of time slots
C	the bandwidth capacity on the outgoing/incoming link of each $i \in \mathcal{N}$
$\mathcal{K}$	the set of deadline coflows
s_k, d_k	the start time and deadline of coflow $k \in \mathcal{K}$
$f_{i,j}^k$	the flow of $k \in \mathcal{K}$ from i to j
$D_{i,j}^k$	the flow volume of $f_{i,j}^k$
$x_{i,j}^{k,t}$	the number of bandwidth units allocated to flow $f_{i,j}^k$ at time slot $t \in \mathcal{T}$
$\mathcal{Z}$	the maximum bandwidth usage of deadline coflows across all links and all time slots

Important notations used throughout our model are listed in Table I. In our mathematical model, we consider there are a set of machines in a datacenter network, which is denoted by $\mathcal{N} = \{1, 2, \dots, N\}$. We consider a discrete-time system and consider that there are a set of time slots, denoted as $\mathcal{T} = \{1, 2, \dots, T\}$. The rationales for such a discrete-time process are two folds. First, the discrete-time model is widely adopted in the network research community, especially in flow/coflow scheduling [19]. Second, with a discrete-time system, one can flexibly set the granularity of a time slot to control the amount of data that can be transmitted in each time slot. At each time slot $t \in \mathcal{T}$, each machine is capable of transmitting *C* units of data through its outgoing link and receiving *C* units of data through its incoming link. Note that each unit of bandwidth is considered to be 1. By doing this, the network operator can have high flexibility to control the minimum size of one data block that can be sent in the network. For example, the operator can set such bandwidth unit to 1500B per time slot, so as to make each data block to be a packet. Moreover, in order to improve the overall throughput, one can also set the bandwidth unit to 64KB or even a higher one, saying 100KB, such that each block can carry more data. On the other hand, assuming each unit of bandwidth to be 1 is also adopted in existing literature, e.g., [19].

A coflow is a set of parallel flows, which can only be considered to be completed after all its flows finish. As mentioned in the previous section, coflows can be divided into deadline coflows and non-deadline coflows. Since the objective is to minimize the impact on the non-deadline coflows when scheduling deadline coflows, we wish always to have free bandwidth after deadline coflows have been scheduled. So, in our mathematical model, we mainly consider the deadline coflows, and the non-deadline coflows can be scheduled later with existing methods. To indicate the deadline coflows, we denote $\mathcal{K} = \{1, 2, \dots, K\}$ as the set of deadline coflows. Let $f_{i,j}^k$ denote a flow of deadline coflow $k \in \mathcal{K}$ that needs to transfer $D_{i,j}^k$ units data from machine *i* to *j*. For each deadline coflow k, we denote s_k and d_k as its start time and deadline, respectively. Similar to existing studies Varys [22], we assume that all flows in a coflow start at the same time, and the information about all the flows can be known once the coflow has arrived at the network.

Decision variable: To indicate the coflow scheduling decision variable, we denote $x_{i,j}^{k,t}$ as the number of bandwidth units allocated to flow $f_{i,j}^k$ at time slot *t*. For simplicity, we assume that each unit of bandwidth is 1, and thus the decision variable $x_{i,j}^{k,t}$ is integer:

$$x_{i,j}^{k,t} \in \mathbb{Z}^+,\ \forall i, j \in \mathcal{N}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T} \tag{1}$$

Specifically, $x_{i,j}^{k,t} = 0$ means that the flow $f_{i,j}^k$ does not exist or this flow is waiting for transmission.

Link capacity constraints: When scheduling coflows, both the outgoing and incoming link capacities should be satisfied. Thus, we have the following two constraints:

$$\sum_{k \in \mathcal{K}} \sum_{j \in \mathcal{N}} x_{i,j}^{k,t} \leq C,\ \forall i \in \mathcal{N}, \forall t \in \mathcal{T} \tag{2}$$

$$\sum_{k \in \mathcal{K}} \sum_{i \in \mathcal{N}} x_{i,j}^{k,t} \leq C,\ \forall j \in \mathcal{N}, \forall t \in \mathcal{T} \tag{3}$$

Constraint (2) means that the total amount bandwidth allocated to all the flows on each outgoing link should be no more than the capacity of this link in any time slot. Similarly, for each incoming link, the summation of bandwidth allocated to all the flows in any time slot must not exceed the corresponding link capacity, as shown in constraint (3).

Deadline constraints: To meet the deadlines of deadline coflows, each flow in a coflow should be completed within the deadline. In our model, we consider that each flow can only be transferred within the deadline because transmitting a flow after its deadline is unnecessary. Hence, we have the following two constraints:

$$\sum_{t=s_k}^{d_k} x_{i,j}^{k,t} = D_{i,j}^k,\ \forall i, j \in \mathcal{N}, \forall k \in \mathcal{K} \tag{4}$$

$$x_{i,j}^{k,t} = 0,\ \forall i, j \in \mathcal{N}, \forall k \in \mathcal{K}, \forall t \in \mathcal{T} \setminus [s_k, d_k] \tag{5}$$

Here, the term $\sum_{t=s_k}^{d_k} x_{i,j}^{k,t}$ calculates the total amount of data that flow $f_{i,i}^k$ transmitted in the duration of $[s_k, d_k]$. Thus, constraint (4) means that each flow in a coflow should be fully transmitted within its deadline. Constraint (5) is used for eliminating the potential cases where a flow is still transmitting after its deadline.

B. Problem Formulation

To formally formulate our problem of minimizing the maximum bandwidth usage incurred by deadline coflows across all time slots and all links, we define $\mathcal{Z}$ as the maximum bandwidth usage across all links and all time slots, which can be expressed as follows:

$$\mathcal{Z} = \max_{i,j \in \mathcal{N}, t \in \mathcal{T}} \sum_{k \in \mathcal{K}} x_{i,j}^{k,t} \tag{6}$$

With the above definition, we are now ready to formulate our optimal problem **P1** as follows:

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{Minimize}}\ \ \mathcal{Z} \\ &\text{Subject to: Eqs. } (1), (2), (3), (4), (5). \end{aligned} \tag{7}$$

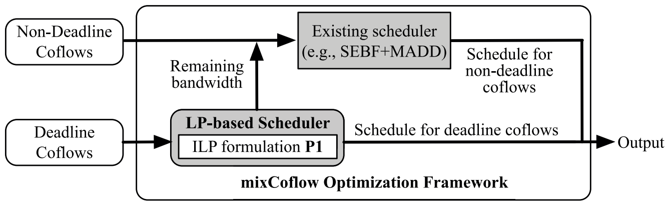

Non-Deadline Coflows
Existing scheduler (e.g., SEBF+MADD)
Remaining bandwidth
Schedule for non-deadline coflows
Deadline Coflows
LP-based Scheduler
ILP formulation P1
Schedule for deadline coflows
Output
mixCoflow Optimization Framework

Fig. 2. Overview of *mixCoflow* optimization framework.

where the objective function (7) is to minimize the maximum bandwidth usage among all links in all time slots, which means that each link has nearly balanced bandwidth usage in each time slot. When the bandwidth usage can be well balanced, non-deadline coflows can have more potential to be accelerated.

The above problem is an integer linear programming (ILP) problem, which is difficult to be resolved due to its min-max objective and the discrete nature. One potential way to solve it is to design some heuristics, which, however, suffers from the optimality issue. One can relax the integer variable in this ILP to a linear variable, and thus obtain a linear programming problem that can be easily solved. Nevertheless, such stiff relaxation also cannot obtain the optimal solution to the ILP. To obtain the optimal solution of this ILP, we make a surprising observation that this ILP can be transformed into an equivalent linear programming (LP) problem, which returns the same optimal solution to the ILP, as we will show in the following section.

IV. The Design of *mixCoflow*

In this section, we present the design of our *mixCoflow*. As we can see from Fig. 2, *mixCoflow* transforms the relevant ILP problem into an equivalent LP and solves this LP to obtain the scheduling decisions for deadline coflows. After that, *mixCoflow* simply employs existing scheduler (e.g., SEBF+MADD [22]) for non-deadline coflows. LP-based scheduler is the key step in *mixCoflow* optimization framework; in what follows, we present how the ILP **P1** can be equivalently solved by a LP-scheduler.

Generally, an integer programm can be transformed into a linear programm if the integer program has a *separable convex objective* and *totally unimodular linear constraints*. The first condition means that the objective function should be able to be represented as a summation of multiple independent convex functions with respect to a single variable. The second condition implies that the coefficient matrix of the integer program should be a totally unimodular matrix where all elements must be chosen from $\{-1, 0, 1\}$ and every square submatrix has determinant 0, 1, or -1. Surprisingly, after taking an in-depth of the structure of **P1**, we find that **P1** exactly has such properties.

A. Separable Convex Objective

We first present the following two definitions:

Lexicographical comparison: Given any $\boldsymbol{\phi} \in \mathbb{Z}^K$ and $\boldsymbol{\varphi} \in \mathbb{Z}^K$, the elements in $\boldsymbol{\phi}$ are sorted in a non-increasing order and the elements in $\boldsymbol{\varphi}$ are also sorted in a non-increasing order. If $\langle\boldsymbol{\phi}\rangle_1 < \langle\boldsymbol{\varphi}\rangle_1$ or $\exists k \in \{2, 3, \ldots, K\}$ such that $\langle\boldsymbol{\phi}\rangle_k < \langle\boldsymbol{\varphi}\rangle_k$ and $\langle\boldsymbol{\phi}\rangle_i = \langle\boldsymbol{\varphi}\rangle_i, \forall i \in \{1, 2, \ldots, k-1\}$, then $\boldsymbol{\phi}$ is *lexicographically smaller* than $\boldsymbol{\varphi}$, i.e., $\boldsymbol{\phi} \prec \boldsymbol{\varphi}$. Similarly, if $\langle\boldsymbol{\phi}\rangle_k = \langle\boldsymbol{\varphi}\rangle_k$, $\forall k \in \{1, 2, \ldots, K\}$ or $\boldsymbol{\phi} \prec \boldsymbol{\varphi}$, then $\boldsymbol{\phi}$ is *lexicographically no greater* than $\boldsymbol{\varphi}$, i.e., $\boldsymbol{\phi} \preceq \boldsymbol{\varphi}$.

Lexicographical minimization: Let $\operatorname{lexmin}_{\boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x})$ denote the *lexicographical minimization* of the vector $\boldsymbol{f} \in \mathbb{R}^N$, which consists of N objective functions of $\boldsymbol{x}$. More precisely, the optimal solution $\boldsymbol{x}^* \in \mathbb{R}^K$ that achieves the optimal $\boldsymbol{f}^*$ must have $\boldsymbol{f}^* = \boldsymbol{f}(\boldsymbol{x}^*) \preceq \boldsymbol{f}(\boldsymbol{x}),\ \forall \boldsymbol{x} \in \mathbb{R}^K$.

With the above definitions, we now show that the optimal solution of problem **P1** can be obtained by solving the following *lexicographically minimization* problem **P2**:

$$\begin{aligned} &\operatorname*{lexmin}_{\boldsymbol{x}}\ \boldsymbol{\xi} = \left(\xi_{1,1}^1, \ldots, \xi_{N,N}^1, \ldots, \xi_{N,N}^T\right) \\ &\text{Subject to: Eqs. } (1), (2), (3), (4), (5), \end{aligned} \tag{8}$$

where $\xi_{i,j}^t = \sum_{k \in \mathcal{K}} x_{i,j}^{k,t}\ \forall i, j \in \mathcal{N}\ \forall t \in \mathcal{T}$, and $\boldsymbol{\xi}$ is a vector with the dimension of $M = |\boldsymbol{\xi}| = TNN$. For this problem, the objective is to minimize the element in $\boldsymbol{\xi}$ which is the maximum bandwidth usage across all links and all time slots. Therefore, the optimal solution $\boldsymbol{x}^*$ that gives $\boldsymbol{\xi}^*$ is also the optimal solution for problem **P1**, i.e., denoted as **P2** $\Rightarrow$ **P1**. To solve problem **P2**, Let $g(\boldsymbol{\xi})$ denote a function of $\boldsymbol{\xi}$:

$$g(\boldsymbol{\xi}) = \sum_{m=1}^{M} M^{\xi_m} = \sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{N}} \sum_{j \in \mathcal{N}} M^{\xi_{i,j}^t} \tag{9}$$

where ξ_m is the m-th element of the vector $\boldsymbol{\xi}$. We can easily check that $g(\boldsymbol{\xi})$ is a summation of convex functions M^{ξ_m}, and accordingly $g(\boldsymbol{\xi})$ is also a convex function.

Theorem 1: $g(\cdot)$ preserves the order of lexicographically no greater $\preceq$, i.e., $\boldsymbol{\xi}^* \preceq \boldsymbol{\xi} \Longleftrightarrow g(\boldsymbol{\xi}^*) \le g(\boldsymbol{\xi})$.

Proof: Here we first prove $\boldsymbol{\xi}^* \preceq \boldsymbol{\xi} \Longrightarrow g(\boldsymbol{\xi}^*) \le g(\boldsymbol{\xi})$. We assume r $(r \ge 1)$ is the index of the first non-zero element in $\boldsymbol{\xi}^* - \boldsymbol{\xi}$, which means that $\xi_i^* = \xi_i\ \forall i < r, \xi_r^* < \xi_r$. Then, we have:

$$\begin{aligned} g(\boldsymbol{\xi}^*) - g(\boldsymbol{\xi}) &= \sum_{m=1}^{M} M^{\xi_m^*} - \sum_{m=1}^{M} M^{\xi_m} \\ &= M^{\xi_r^*} + \sum_{m=r+1}^{M} M^{\xi_m^*} - M^{\xi_r} - \sum_{m=r+1}^{M} M^{\xi_m} \\ &\le (M - r + 1) M^{\xi_r^*} - M^{\xi_r} \\ &\le M^{\xi_r^* + 1} - M^{\xi_r} \le 0 \end{aligned} \tag{10}$$

With the above inequalities, we get $\boldsymbol{\xi}^* \preceq \boldsymbol{\xi} \Longrightarrow g(\boldsymbol{\xi}^*) \le g(\boldsymbol{\xi})$. Now, we focus on the proof of $g(\boldsymbol{\xi}^*) \le g(\boldsymbol{\xi}) \Longrightarrow \boldsymbol{\xi}^* \preceq \boldsymbol{\xi}$ through its contrapositive: $\neg(\boldsymbol{\xi}^* \preceq \boldsymbol{\xi}) \Longrightarrow g(\boldsymbol{\xi}^*) > g(\boldsymbol{\xi})$, here $\neg(\boldsymbol{\xi}^* \preceq \boldsymbol{\xi}) \Longleftrightarrow \boldsymbol{\xi} \prec \boldsymbol{\xi}^*$, therefore, we should prove $\boldsymbol{\xi} \prec \boldsymbol{\xi}^* \Longrightarrow g(\boldsymbol{\xi}) < g(\boldsymbol{\xi}^*)$, it can be easily proved by (10). Hereto, theorem is proved. ■

Based on the above Theorem 1, we now formulate the following problem **P3** that is equivalent to the problem **P2**, in terms of the optimal solution:

$$\operatorname*{Minimize}_{\boldsymbol{x}}\ g(\boldsymbol{\xi})$$

$$\text{Subject to: Eqs. } (1),(2),(3),(4),(5). \tag{11}$$

Since $\boldsymbol{\xi}^*$ is lexicographically minimization, we have $\boldsymbol{\xi}^* \preceq \boldsymbol{\xi}\ \forall \boldsymbol{\xi}$. Using $\boldsymbol{\xi}^* \preceq \boldsymbol{\xi} \iff g(\boldsymbol{\xi}^*) \leq g(\boldsymbol{\xi})$, we can get the fact that the minimum value of $g(\boldsymbol{\xi})$ is $g(\boldsymbol{\xi}^*)$. Hence, **P3** has the same optimal solution as problem **P2**, denoted as **P3** = **P2**.

B. Totally Unimodular Constraint Matrix

In addition to the separable convex objective, the totally unimodular constraint matrix is an important factor that enforces a LP problem to have integral solutions. More precisely, denoting the feasible region of a LP problem as $\{x|A\boldsymbol{x} = b\}$ or $\{x|A\boldsymbol{x} \leq b\}$, if the constraint matrix A is totally unimodular and b is integral, then such feasible region is an integral polyhedron and it only has integral extreme points. Typically, an *m*-by-*n* matrix is a totally unimodular coefficient matrix, if it satisfies the following two conditions:

1) All elements of this matrix are in the range of $\{-1, 0, 1\}$;
2) For any subset $R \subset \{1, 2, \ldots, m\}$, it can be divided into two disjoint sets: R_1 and R_2, such that $|\sum_{i\in R_1} a_{ij} - \sum_{i\in R_2} a_{ij}| \leq 1\ \forall j \in \{1, 2, \ldots, n\}$.

The following theorem verifies that the coefficient matrixes for all constraints in the original ILP problem exactly form a totally unimodular matrix.

Theorem 2: The coefficient matrixes of the linear constraints (2), (3), (4) and (5) form a totally unimodular matrix.

Proof: We observe that both the constraints (2) and (3) have *TN* inequalities, while the constraint (4) has *KNN* equations and the constraint (5) has $NN\sum_{k=1}^{K}(S_k - 1 + T - D_k)$ equations. Denote $A_{m\times n}$ as the coefficient of all of the inequations and equations, where $m = 2TN + KNN + NN\sum_{k=1}^{K}(S_k - 1 + T - D_k)$, and $n = KNNT$. It should be noted that n is the dimension of the variable $\boldsymbol{x}$. We can easily check that any element of $A_{m\times n}$ is either 0 or 1, which means that the condition 1 can be satisfied. For any subset $R \subset \{1, 2, \ldots, m\}$, we can select all the elements that belong to $\{1, 2, \ldots, 2TN\}$ to compose the set R_1, and let the rest of the elements compose the set R_2. It is easy to check that the summation of all rows of R_1, is a $1 \times n$ vector with all elements equal to 2. Similarly, the summation of all rows of R_2, is a $1 \times n$ vector with elements equal to 1. Hence, we have $\sum_{i\in R_1} a_{i,j} = 2$ and $\sum_{i\in R_2} a_{i,j} = 1$, Eventually, $|\sum_{i\in R_1} a_{ij} - \sum_{i\in R_2} a_{ij}| \leq 1\ \forall j \in \{1, 2, \ldots, n\}$, implying that the Condition 2 is satisfied.

In summary, we have proved that both conditions for total unimodularity are satisfied, thus the theorem is proved. ■

C. Transform to LP Problem

LP Formulation: Given convex function $f_i(u_i)\ \forall u_i \in \mathcal{U}$, the broader class of problems of the form:

$$\min_{\boldsymbol{u}} \sum_{i\in\mathcal{N}} f_i(u_i)$$
$$\text{s.t. } A\boldsymbol{u} \leq b,\ \boldsymbol{u} \geq 0,\ \boldsymbol{u} \text{ integer.} \tag{12}$$

where A is a totally unimodular matrix and b is integer. Using λ-representation technique, we can transform problem (12) to a linear programming problem [23]:

$$\begin{aligned} \min_{\boldsymbol{\lambda},\ \boldsymbol{u}}\ & \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{U}\bigcap\mathbb{Z}} f_i(j)\lambda_{i,j} \\ \text{s.t.}\ & A\boldsymbol{u} \leq b,\ \boldsymbol{u} \geq 0 \\ & \sum_{j\in\mathcal{U}\bigcap\mathbb{Z}} \lambda_{i,j} = 1,\ \forall i \in \mathcal{N} \\ & \sum_{j\in\mathcal{U}\bigcap\mathbb{Z}} j\lambda_{i,j} = u_i,\ \forall i \in \mathcal{N} \\ & \lambda_{i,j} \in \mathbb{R}^+ \quad \forall i \in \mathcal{N},\ \forall j \in \mathcal{U}\cap\mathbb{Z} \end{aligned} \tag{13}$$

which has the same optimal solution to problem (12).

Now, we transform problem **P3** to a LP problem. Because **P3** is a convex problem and its coefficient matrix of linear constraints (2), (3), (4) and (5) form a totally unimodular matrix, with λ-representation technique, coincidentally, we can transform problem **P3** to a LP problem **P4** as follows:

$$\begin{aligned} \min_{\boldsymbol{\lambda},\boldsymbol{x}}\ & \sum_{t\in\mathcal{T}} \sum_{i\in\mathcal{N}} \sum_{j\in\mathcal{N}} \sum_{s\in\mathcal{S}} M^s \lambda_{i,j}^{t,s} \\ \text{s.t.}\ & \sum_{s\in\mathcal{S}} \lambda_{i,j}^{t,s} = 1,\ \forall i,j \in \mathcal{N}, \forall t \in \mathcal{T},\ \mathcal{S} = [0, C]\cap\mathbb{Z} \\ & \sum_{s\in\mathcal{S}} s\lambda_{i,j}^{t,s} = \xi_{i,j}^t = \sum_{k\in\mathcal{K}} x_{i,j}^{k,t},\ \forall i,j \in \mathcal{N}, \forall t \in \mathcal{T} \\ & \lambda_{i,j}^{t,s}, x_{i,j}^{k,t} \in \mathbb{R}^+,\ \forall i,j \in \mathcal{N}, \forall t \in \mathcal{T}, \forall s \in \mathcal{S}, \forall k \in \mathcal{K} \\ & \text{Constraints } (2),(3),(4),(5). \end{aligned} \tag{14}$$

where, problem **P4** and **P3** have the same optimal solution, denoted as **P4** = **P3**.

Theorem 3: Problem **P4** has the same optimal solution with problem **P1**.

Proof: we can get **P4** = **P3** and **P3** = **P2** from equations (14) and (11), respectively. We also have **P2** ⇒ **P1** due to equation (8). Hence, we have:

$$\mathbf{P4} = \mathbf{P3} = \mathbf{P2} \Rightarrow \mathbf{P1}, \tag{15}$$

where **P4** ⇒ **P1**, it means that the optimal assignment variables $\boldsymbol{x}^*$ that gives **P4** is also the optimal solution for Problem **P1**. Therefore, theorem is proved. ■

Given the above LP problem, *mixCoflow* can then schedule the deadline and non-deadline coflows with the following steps. First, whenever an existing deadline coflow completes, or a new deadline coflow arrives, our *mixCoflow* will solve the LP problem. As such, the amount of bandwidth that should be allocated to all the deadline coflows over all time slots and all links can be obtained. Second, *mixCoflow* will allocate the remaining bandwidth resource to non-deadline coflows with any existing method, such as FIFO and SEBF+MADD.

V. Performance Evaluation

In this section, we first evaluate *mixCoflow* by large-scale simulations based on a real-world data trace collected from

Facebook [28]. Summary of main results are highlighted as follows:

- Using *mixCoflow* in comparison to Varys, non-deadline coflows completes up to $9.35\times$ faster, and at the same time, $\sim 1\%$ more coflows meet their deadlines.
- Compared to Varys, *mixCoflow* can maintain more balanced bandwidth usage of deadline coflows across all links and all time slots.
- *mixCoflow* can always achieve a lower average CCT as well as a lower deadline miss rate than Varys, irrespective of the total number of coflows, the coflow size, the coflow width, and the inter-coflow arrival interval.

A. Trace-Driven Simulations

Simulation setup: We simulate a datacenter network with 150 machines. The incoming/outgoing link of each machine is uniformly set to be 800Mbps, which is a common setting in production datacenter [22].

Data trace: We use the Hive/MapReduce trace provided by Facebook as the workload in our simulations, which is a widely adopted trace in the existing studies [20], [22]. This trace is from a 3000-machine 150-rack cluster and contains 526 coflows. In the original trace, all coflows are scaled down to the rack-level. In other words, mappers in the same rack are consolidated as one rack-level mapper, and so are the reducers. Each line in the original trace contains: coflow id, arrival time, number of mappers, location of each mapper, number of reducers, location of each reducer: shuffle megabytes fetched by this reducer. We can easily check that each coflow in the original trace only contains the whole data size that each reducer needs to fetch. Hence, we uniformly sample the size for each flow within it and accordingly obtain the information of all the flows.

We use the entire data-trace in our simulation and divide all the 526 coflows into two categories of coflows, i.e., deadline coflows and non-deadline coflows, by using a *ratio*. For example, a ratio of 3:1 means that 75% of the 526 coflows are deadline coflows, and the rest are non-deadline coflows. Specifically, the deadline of each deadline coflow is set to be its minimum completion time in an empty network multiplied by $(1 + U(0, x))$, where $U(0, x)$ is a uniformly random number in the range $(0, x)$. Unless otherwise specified, $x = 1$. Such deadline settings are similar to the study of [22].

Comparing solutions: We compare the following schemes with *mixCoflow* in our simulations. It should be noted that each of the following schemes is only used to schedule the deadline coflows rather than non-deadline coflows. For completeness, after the deadline coflows are scheduled by each scheme, we use SEBF heuristic to schedule the non-deadline coflows.

- **FIFO (First-In-First-Out)**: schedules deadline coflows based on their arriving times [9]. This scheme aggressively takes all the bandwidth when scheduling each deadline coflow, which may seriously impact the CCTs of non-deadline coflows.
- **EDF (Earliest-Deadline-First)**: all the deadline coflows are scheduled in ascending order of their deadlines [29]. This scheme strictly prioritizes deadline coflows and could complete a coflow far before its deadline, which is actually unnecessary and may increase the CCTs of non-deadline coflows.
- **Varys**: schedules deadline coflows with the Shortest-Effective-Bottlence-First (SEBF) first, and then leverages Minimum-Allocation-for-Desired-Duration (MADD) to allocate bandwidth for each flow in a deadline coflow [22]. This scheme uses the exactly right bandwidth resources to guarantee the deadlines of deadline coflows. However, it makes no attempt to balance footprint, and thus may impact the non-deadline coflows.

Performance metrics: For deadline coflows, the primary metric is the deadline miss rate, which is the percentage of deadline coflows that miss their deadlines. For non-deadline coflows, we define the *factor of improvement* in the average CCT (coflow completion time) as the primary metric. More specifically, the factor of improvement of scheme 1 compared to scheme two can be calculated as

$$\text{Factor of Improvement} = \frac{CCT_2}{CCT_1} \tag{16}$$

where CCT_1 and CCT_2 are the average CCTs achieved by scheme 1 and scheme 2, respectively. If the factor of improvement is greater than 1, then scheme 1 achieves a lower average CCT than scheme 2: the larger the factor of improvement, the more performance gain scheme 1 can achieved, compared to scheme 2. On the other hand, if such factor is less than 1, then scheme 2 can do better in reducing average CCT than scheme 1.

For deadline coflows, we mainly show the results on the deadline miss rate. For non-deadline coflows, we mainly present the results of the factor of improvement on the average CCT. Detailed simulation results are shown as follows:

1) Deadline Miss Rate: As aforementioned, deadline coflows are mission-critical and are only useful to the applications when they are completed before their deadlines. By varying the percentage of deadline coflows from 10% to 100%, we show the deadline miss rates achieved by different methods in Fig. 3. It is easy to find that our *mixCoflow* incurs a lower deadline miss rate than the FIFO method, under all the settings of the percentages of deadline coflows. The root cause is that FIFO does not consider the deadlines, yet it may finish individual flows quickly while it is unaware of the progress of their corresponding coflows, thus leading a high deadline miss rate. On the other hand, we can further observe that *mixCoflow* incurs more or less deadline miss rate, compared to the EDF scheme. This is because that EDF aggressively takes all the bandwidth to complete deadline coflows. And that's why the CCTs of non-deadline coflows will be hurt after the deadline coflows are scheduled with EDF (we will show this point later). As for the Varys scheme, our *mixCoflow* can enjoy a little benefit on the deadline miss rate. The root reason is that Varys embraces a complicated admission control mechanism and heuristically allocates bandwidth to deadline coflows based on current remaining bandwidth resource, making it easy to overlook the optimal allocation. By contrast, our *mixCoflow* determines whether to admit a deadline coflow purely based on

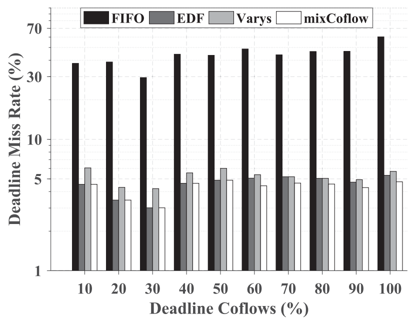


Fig. 3. The deadline miss rate under different percentages of deadline coflows. Y-axes are in logarithmic scale.

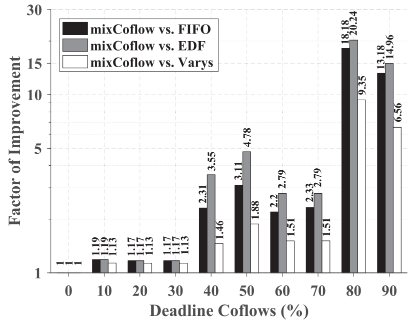


Fig. 4. The factor of improvement in the average CCT of non-deadline coflows. Y-axes are in logarithmic scale.

the entire network capacity and hence more deadline coflows can be accommodated.

2) Factor of Improvement: After deadline coflows are scheduled, the remaining bandwidth resource can be used for the non-deadline coflows. As aforementioned, the scheduling of deadline coflows will impact the CCTs of non-deadline coflows. We, therefore, use the factor of improvement in the average CCT of non-deadline coflows to show that our *mix-Coflow* can efficiently reduce such impact. We show the factor of the improvement in the average CCT in Fig. 4, with varying percentages of deadline coflows. It is clear that our *mixCoflow* can reduce the average CCT of non-deadline coflows, compared to all the other comparison methods. Especially, when the percentage of deadline coflows is 80%, our *mixCoflow* can improve the average CCT by up to 18.18×, 20.24×, 9.35×, when compared to FIFO, EDF and Varys schemes, respectively. To note that, *mixCoflow* brings such high improvements in the average CCT of non-deadline coflows because it makes efforts to minimize and balance the bandwidth usage of deadline coflows over all time slots and all links. As a result, the non-deadline coflows can be minimally impacted, and thus the average CCT can be significantly reduced (we will show this point in Section V-A6). One may wonder at this point that why the factor of improvement equals to 1 when there are no deadline coflows. This is because that all schemes (including the comparison schemes and *mixCoflow*) use the same heuristic (i.e., SEBF) to schedule the non-deadline coflows, but using different methods to schedule deadline coflows.

3) Remaining Bandwidth: It is important to keep in mind that the key idea of this article is to schedule deadline coflows

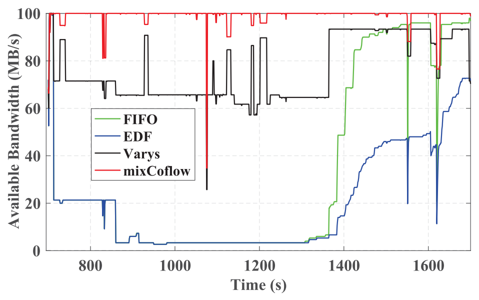


Fig. 5. The remaining available bandwidth on all links and all time slots.

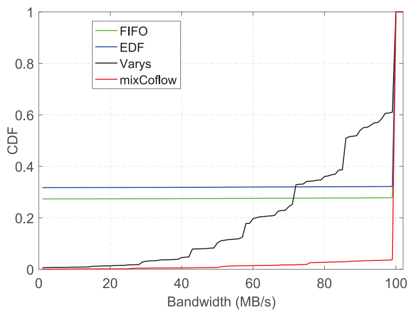
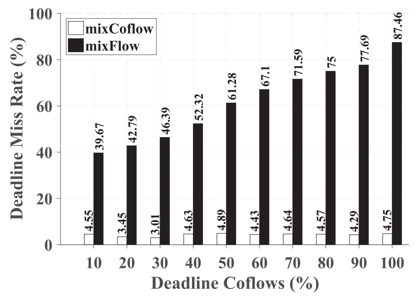


Fig. 6. The CDF of the rest bandwidth on all links and all time slots.

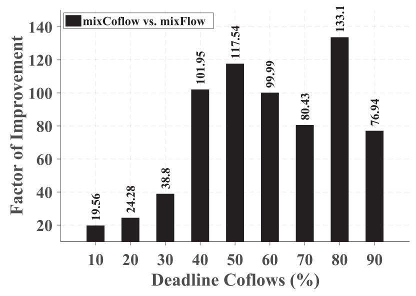
with minimal impact on non-deadline coflows by balancing and minimizing the bandwidth footprint of deadline coflows over all time slots and all links. To completely understand this point, we record the remaining bandwidth on each link at each time slot when the deadline coflows have been scheduled. To ease the presentation, we mainly present the average remaining bandwidth across all links in a scenario where the percentage of deadline coflows is 75%, as shown in Fig. 5. From this figure, we can observe that *mixCoflow* has more remaining bandwidth than FIFO, EDF, and Varys schemes most of the time. Moreover, the remaining bandwidth incurred by *mix-Coflow* is more balanced than the other compared methods. This is why *mixCoflow* can achieve a low average CCT of coflows.

To understand on a microscopic level, we also plot the CDF of the remaining bandwidth across all time slots and all links in Fig. 6. We can clearly find that the curve of *mixCoflow* is lower than each of the curves of the other comparison schemes. This implies that *mixCoflow* can leave more bandwidth to non-deadline coflows after the deadline coflows have been scheduled. We can further observe that the curve of *mixCoflow* maintains at a very low value and grows slowly at the beginning, and then quickly converges to 1. More precisely, under *mixCoflow*, most of the values of the remaining bandwidths can be maintained around a value (i.e., 100MB/s) except only a little portion of such values. Such property essentially means that our *mixCoflow* can achieve a more balanced bandwidth usage.

4) Comparison With Scheme Scheduling a Mix of Flows: Our *mixCoflow* schedules a mix of coflows with and without deadlines. One may wonder if *mixCoflow* can outperform a scheme that schedules a mix of individual flows with and without deadlines. To answer this query, we construct a



(a) The deadline miss rate of deadline coflows.



(b) The factor of improvement in the average CCT of non-deadline coflows.

Fig. 7. The comparison of the deadline miss rate of deadline coflows and average CCT of non-deadline coflows achieved by *mixCoflow* and Mix-Flow.

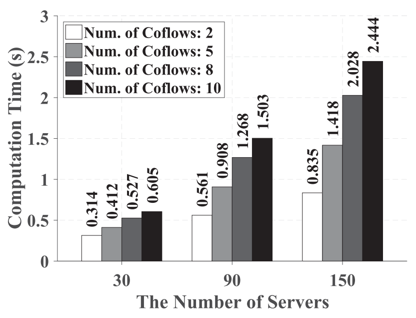


Fig. 8. Computation time of Problem **P4**.

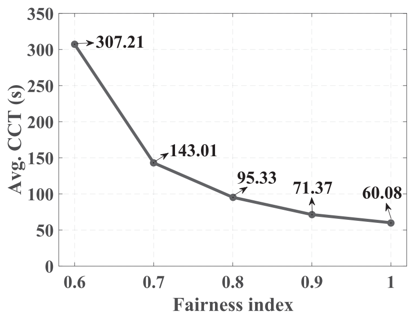
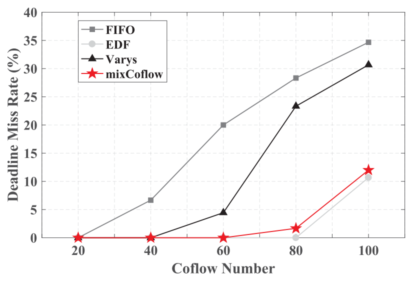


Fig. 9. The Avg. CCT of non-deadline coflows on the different *Jain's fairness index*.

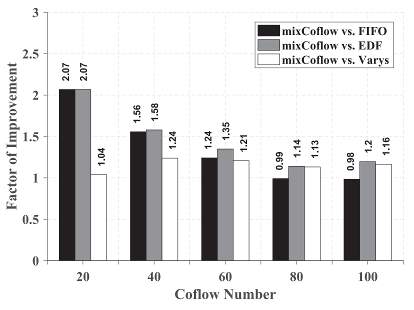
scheme called *mixFlow*. This scheme leverages the Shortest-Remaining-Time-First (SRTF) to schedule deadline flows and then uses the Shortest-Flow-First (SFF) to schedule non-deadline flows, which is conceptually equivalent to Karuna [30]. We conduct an experiment to compare our *mixCoflow* with *mixFlow*. The workload is still Facebook data-trace. The deadline for each flow is set to the deadline of its parent coflow. Fig. 7(a) first shows the deadline miss rates achieved by both *mixCoflow* and *mixFlow* under different percentages of deadline coflows. It is clear that *mixCoflow* can always achieve a lower deadline miss rate than *mixFlow*. We can further observe that the deadline miss rate incurred by *mixFlow* increases as deadline coflows increase while our *mixCoflow* remains relatively a stable deadline miss rate. The root cause for such high deadline miss rate of *mixFlow* is that *mixFlow* does not take into account the flow dependency semantics in a coflow, and thus some flows in a coflow can meet deadlines while some others cannot, leading to this coflow miss deadline either. Fig. 7(b) further presents the factor of improvements of *mixCoflow* over *mixFlow*, in the average CCT of non-deadline coflows, with varying ratios of deadline coflows to non-deadline coflows. We can easily check that *mixCoflow* outperforms *mixFlow* as the factor of improvement is always larger than 1. With further observations, the factor of improvement can be up to 133.1, while the average of all the factors of improvement is 76.95. The reason is still that *mixFlow* favors only the small flows irrespective of the progress of their parent coflows. In summary, compared to *mixFlow*, *mixCoflow* can make more deadline coflows meet deadlines while reducing the average CCT of non-deadline coflows.

5) LP Solver Efficiency: The dominant overhead of our *mixCoflow* is caused by solving the relevant LP problem **P4**. One may question if solving the LP will incur high scheduling latency and hurt the performance of deadline coflows. To answer this question, we conduct an experiment to show the running time of computing the LP decision. We use the MOSEK solver [31] on a standard computer with 2.70GHz Intel Core i7-7100U CPU (64-bit) and 16G of main memory. The algorithm used in this solver is the interior point method, which has a time complexity of $O(n^{3.5}L^2)$. Here, n is the dimension of the decision variable, and L is the number of variables; the two numbers are 4 and *KNNT*, respectively, corresponding to our LP problem **P4**. We evaluate the time to solve the LP problem **P4** under different number of coflows and servers. Note that after an in-depth analysis of the data-trace provided by Facebook, we observe that the number of concurrent coflows is at most 10 at any given time. Hence, we only focus on scenarios with concurrent coflows being no more than 10. The results are shown in Fig. 8. It is clear that the running time increases with the number of coflows increase as well as with the increase in the number of servers. More specifically, it takes about 2.444 seconds to solve the LP problem when there are 10 concurrent coflows and 150 servers. This is a relatively small time as compared to the coflow deadlines, which can range from 10 to 3629 time slots, while the length of each time slot is 1 second. It should be noted that such efficiency is because we transform the original ILP into the equivalent LP. For problems with more coflows and larger network size, one can use some commercial solvers CPLEX [32], which can return the LP results in 1-2 seconds.

6) Effectiveness of Balancing Bandwidth Usage: As aforementioned, our *mixCoflow* schedules the deadline coflows with the object of minimizing and balance their bandwidth usage over time and links. To demonstrate that such balanced bandwidth usage of deadline coflows can benefit non-deadline coflows, we conduct an experiment by scheduling

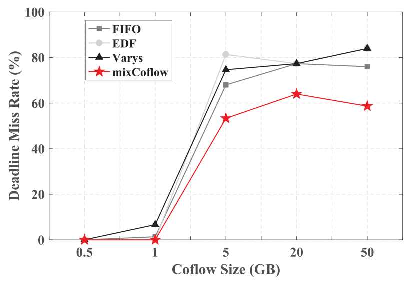


(a) Deadline miss rate vs. Coflow Number

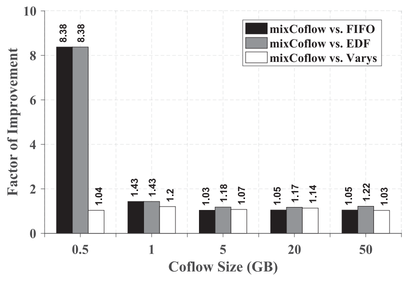


(b) Factor of improvement vs. Coflow Number

Fig. 10. The impact of Coflow Number.



(a) Deadline miss rate vs. Coflow Size



(b) Factor of improvement vs. Coflow Size

Fig. 11. The impact of Coflow Size.

non-deadline coflows under varying imbalanced bandwidth usage. To indicate the balancing level of the bandwidth usage, we introduce the Jain's fairness index [33]. Assume that $w_{i,j}(t)$ is the bandwidth usage of link (i, j) at t. Then, the Jain's fairness index of bandwidth usage of all links at t is defined as $F(t)=\frac{(\sum_{i,j} w_{i,j}(t))^2}{|w|\cdot\sum_{i,j} w_{i,j}(t)^2}$ where $|w|$ is the number of all links. Clearly, $F(t)$ is a value less than 1, and a value closer to 1 means the bandwidth usages are more balanced. Fig. 9 shows the average CCT of non-deadline coflows under different Jain's fairness index of bandwidth usage of deadline coflows. From this figure, we observe that the average CCT of non-deadline coflows decreases as the Jain's fairness index increases. This directly demonstrates that balancing the bandwidth usage of deadline coflows can promote the completion of non-deadline coflows. We can further observe that the average CCT under a high Jain's fairness index (e.g., 1) can be reduced by up to 80.4%, compared to that under a low Jain's fairness index (e.g., 0.6).

B. Impact of Coflow Parameters

Coflow parameters may affect the performance of our *mixCoflow*. For example, a dumpy coflow may occupy more ports/links than a lanky coflow; a large coflow requires more bandwidth than a short coflow. Considering these factors, we mainly evaluate 4 types of coflow parameters: the total number of coflows, the coflow size (i.e., the total amount of data in a coflow), the coflow width (the total number of flows in a coflow), and the inter-coflow arrival interval. Note that in this part, we randomly generate traffic for coflows. Moreover, even though the same set of coflow parameters may be taken in each group of experiments, the coflows may have different flow size distribution. Unless otherwise specified, the percentage of deadline coflows is set to be 75% for the following simulations.

1) The Total Number of Coflows: We first evaluate the impact of the total number of coflows on the performance of our *mixCoflow*. When doing this part of the evaluation, we fix the other parameters, i.e., setting the coflow size, the coflow width, and the mean inter-coflow arrival interval as 5GB, 100, 100ms, respectively. We then compare different coflow scheduling schemes under different numbers of coflows and plot the results in Fig. 10.

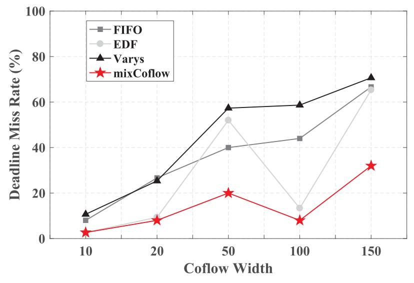
Fig. 10(a) first shows the deadline miss rates for different schemes. As we can see from this figure, the deadline miss rates for all schemes increase with the growth of the number of coflows. In general, *mixCoflow* can achieve a lower deadline miss rate than FIFO and Varys, while incurring a little more deadline miss rate than EDF when the number of coflows increases to 80. The underlying reason is that EDF uses the entire network bandwidth whenever it schedules a deadline coflow, while our *mixCoflow* only allocates the least amount of bandwidth to complete a coflow within the deadline and can leave more bandwidth to non-deadline coflows than EDF. That's why EDF will slow down the completion of the non-deadline coflows (as we will show later in Fig. 10(b)). In Fig. 10(b), we can observe that the factor of improvement has a declining trend with the growth of the coflow numbers. This is because an increase in the deadline miss rates can lead to more remaining bandwidth for the non-deadline coflows. We also find that the factor of improvement achieved by *mixCoflow* can be up to 2.07×, 2.07×, and 1.24×, compared to FIFO, EDF, and Varys schemes, respectively. The above results verify that our *mixCoflow* can reduce the CCTs of non-deadline coflows while guaranteeing the deadlines of deadline coflows, irrespective of the total number of coflows.

2) The Coflow Size: Now, we fix the coflow number, the coflow width, and the mean inter-coflow arrival interval to be 100, 100, and 100ms, respectively, and evaluate the impact of the coflow size on the performance of *mixCoflow*. The simulation results are presented in Fig. 11.

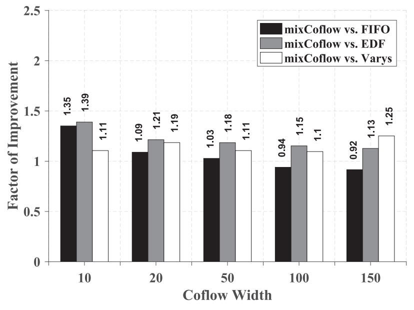
Fig. 11(a) first shows that the deadline miss rate increases with the increase of the coflow size. This is reasonable because the total network capacity remains unchanged in different coflow sizes. The larger the coflow size, the more load in the network, and eventually, the larger the deadline miss rate will be incurred. However, *mixCoflow* maintains a lower deadline miss rate than all the other comparison schemes, such as FIFO, EDF, and Varys. One may wonder at this point why Fig. 10(a) and Fig. 11(a) show different deadline miss rate for the same set of coflow parameters: number of coflows=100, size=5GB, width=100 and arrival interval=100ms. The root reason lies in the fact of random traffic generation. The coflows may have relatively balanced flow size distributions in Fig. 10(a) while exhibiting skewed distributions in Fig. 11(a). In addition, for different coflow sizes, Fig. 11(b) shows *mixCoflow* can improve the factor of improvement in the average CCT by up to 8.38×, 8.38× and 1.2×, compared to FIFO, EDF and Varys. These results have demonstrated that *mixCoflow* can always outperform the other schemes, irrespective of the coflow size.

3) The Coflow Width: In this part of the simulation, we fix the coflow number, the coflow size, and the mean inter-coflow arrival interval to be 100, 5GB, and 100ms, respectively. We conduct simulations to evaluate the impact of the coflow width, which is the number of the individual flows in each coflow. As shown in Fig. 12(a), the deadline miss rate has an increasing trend with the increase of the coflow width. Moreover, *mixCoflow* can achieve the lowest deadline miss rate among all of the scheduling schemes, across all the settings of coflow widths. In Fig. 12(b), we can further find that *mixcoflow* improves the factor of improvement in the average CCT by 1.35×, 1.39× and 1.25×, when compared with FIFO, EDF and Varys schemes, respectively. These results effectively verify that *mixCoflow* can reduce the average CCT of non-deadline coflows, regardless of the coflow width.

4) The Inter-Coflow Arrival Interval: Similarly, we fix the coflow number, the coflow size, and the coflow width to be 100, 5GB, and 100, respectively. Note that the mean inter-coflow arrival interval is set to be a fixed, and we investigate the intervals from 0 to 2000ms. The impact of the intervals on the performance of our *mixCoflow* is plotted in Fig. 13. First, in Fig. 13(a), we can see that under each scheduling scheme, the deadline miss rate decreases with the increasing of the inter-coflow arrival intervals. Moreover, we can further observe that *mixCoflow* can achieve a relatively low deadline miss rate for most settings of the intervals. When considering the CCTs of non-deadline coflows, we can find that *mixCoflow* can improve the factor of improvement in the average CCT by up to 7.72×, 7.72× and 1.45×, compared to FIFO, EDF and Varys schemes, respectively, as shown in Fig. 13(b). The reason is that *mixCoflow* is able to efficiently balance the bandwidth usage across all links and all the time slots. In this way, *mixCoflow* can reduce the average CCT of non-deadline coflows.

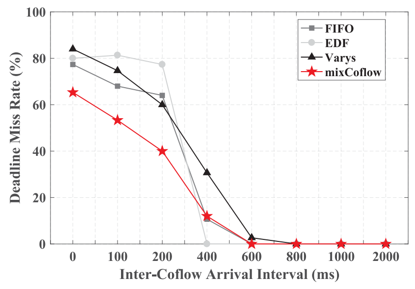


(a) Deadline miss rate vs. Coflow Width

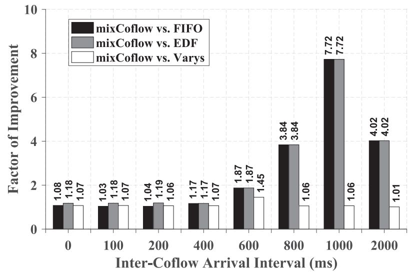


(b) Factor of improvement vs. Coflow Width

Fig. 12. The impact of Coflow Width.



(a) Deadline miss rate vs. Inter-Coflow Arrival Interval



(b) Factor of improvement vs. Inter-Coflow Arrival Interval

Fig. 13. The impact of Inter-Coflow Arrival Interval.

VI. Discussion

Unknown Coflow Size: Coflow size may not be available [14], but only limited to non-deadline coflows. This is because that for most applications, if the flows have deadlines, then these flows' sizes can also be known [30]. So, we can still minimize and balance the bandwidth footprint for deadline coflows. As such, it remains only to see how to schedule the non-deadline coflows with unknown sizes. One straightforward approach is to leverage the *Least-Attained-Service-First* (LASF) scheduling [14] policy, which, however, may lead to inferior performance. One better way for scheduling non-deadline coflows without size information would be to leverage some ensemble learning or even advance deep learning models to predict the coflow size first [34], [35] and then apply SEBF+MADD strategy for scheduling. We leave this point as one direction of our future work.

Heterogeneous Link Capacity: So far, we only focus on homogeneous clusters where the incoming/outgoing links have the same bandwidth capacity. However, datacenter-scale clusters may contain a mix of high-performance and low-power servers of disparate hardware architectures [36], leading to heterogeneous capacities in the incoming/outgoing links of servers. In this case, our solution can work, as well. First, we can reformulate the problem by replacing the homogeneous link capacities in constraints (2) (3) with heterogeneous capacities. Then, it can easily be checked that the new formulation can still satisfy the separable convex objective and totally unimodular linear constraints, meaning that it can still be transformed into an LP.

Practicality issues: Our *mixCoflow* may not suffer from serious practicality issues. On the one hand, the dominant overhead of *mixCoflow* lies in solving the LP, which actually takes negligible time compared to the CCT of a coflow. On the other hand, to enable deadline coflows can go ahead of non-deadline coflows, we can leverage the priority queues existed in most commodity switches. For example, we can tag the header of the packets of the deadline coflows with a high priority and configure a low priority for non-deadline coflows. In this way, the packets of deadline coflows will always be prioritized over that of non-deadline coflows in the network. In addition, to enforce each flow to be sent with computed rate, one can also leverage the Linux Traffic Control to perform per-flow rate-limiting.

Coflow Deadline Miss Rate Reduction: Although we spent a lot of effort to meet the coflow deadline, the coflow deadline miss rate can not be as low as 0. To further reduce the deadline miss rate, one can introduce a slack for the deadline coflows, as similar to [12]. For example, if the real deadline of a coflow is 50 seconds and the slack is 10 seconds, then we use 40 seconds as the deadline in our LP formulation. On the other hand, one may still want to use the coflows that slightly miss its deadline. In this case, we can relax the deadline of a coflow to a relatively larger value based on its coflow size and let the larger coflow to get a larger extension in its deadline. For example, if the deadline of a coflow is 50 seconds and its size is 100 MB, then we can introduce a relax factor (e.g., 0.01 second/MB) and extend its deadline to $50 + 0.01 * 100 = 51$ seconds.

VII. Related Work

mixCoflow focuses on jointly scheduling deadline and non-deadline coflows in a datacenter, with the objective of meeting deadlines of deadline coflows while reducing the CCTs of non-deadline coflows. There is a large body of recent work that optimizes the performance of deadline or non-deadline traffic. In this section, we only discuss some closely related ones.

Flow-level scheduling: Researchers have made continuous efforts on flow-level scheduling. For flows without deadline requirements, existing methods largely focus on minimizing the flow completion times. For instance, DCTCP [10] and HULL [37] reduce switch queue occupancy to shorten flow completion time by dynamically adjusting sending windows based on the level of congestions. pFabric [24] tags each packet a priority based on the remaining flow size and then schedules packets in the order of least remaining size at the switches. For the flows to be completed within deadlines, D^3 [11] and D^2TCP [38] incorporate deadline-awareness into rate-control techniques. Taking a step further, PDQ [39] emulates an earliest deadline first strategy by enabling preemptive scheduling. These approaches are effective in guaranteeing deadlines for individual flows. However, they are insufficient in providing deadline guarantees for coflows. The crux is that these approaches are likely to result in a case where the slowest flow in a coflow misses the deadline while most of the others meet the deadline. This case prevents the whole coflow from finishing on time. Hence, lacking coflow-level information, the above approaches cannot guarantee coflow deadlines. There are also some solutions on scheduling a mix of flows with and without deadlines [30], [40] as well as the solutions on providing performance guarantee for individual flows or even the upper-layer jobs [41], [42], [43], [44], [45]. However, all of them are coflow-agnostic.

Coflow-level scheduling: Existing work on coflow scheduling can be categorized into two categories: guaranteeing deadlines for deadline coflows and reducing CCTs for non-deadline coflows. For guaranteeing coflow deadlines, existing work mainly focuses on decreasing coflow deadline miss rate [21], [22]. For example, Varys [22] first leverages an admission control mechanism to reject coflows whose minimum possible CCT exceeds their deadlines. Then, Varys separately design two sets of strategies (i.e., SEBF and MADD) to schedule the admitted coflows. Taking one step further, Chronos [21] combines priority-based scheduling and limited multiplexing techniques to allocate bandwidth for coflows to just finish on time. However, these works are unaware of the bandwidth footprint of deadline coflows over time, and thus they will hurt the performance of non-deadline coflows. On the other hand, for reducing CCTs for non-deadline coflows, the typical research works (e.g., [14], [15], [16], [19], [20], [22], [46], [47], [48], [49]) mainly apply simple heuristics, such as FIFO, EDF, MRTF (Minimum-Remaining-Time-First), *x-Approximation* and SEBF, to schedule non-deadline coflows. While these works are efficient in reducing the CCTs of non-deadline coflows, they do not consider deadline coflow scheduling. The main difference between our *mixCoflow* and the above existing works lie in that *mixCoflow* jointly consider the deadline and non-deadline coflows, yet is able to reduce the impact on the non-deadline coflows when scheduling deadline coflows.

VIII. Conclusion

In this article, we present *mixCoflow*, a new coflow-aware optimization framework that jointly schedules a mix of coflows with and without deadlines. When scheduling deadline coflows, *mixCoflow* attempts to minimize and balance the bandwidth footprint across time slots and all links, so as to leave more bandwidth for non-deadline coflows and reduce the impact on the CCTs of non-deadline coflows. Specifically, in *mixCoflow*, we formulate a lexicographical min-max ILP problem for scheduling deadline coflows. With several steps of non-trivial transformations, we prove that the optimal solution to this ILP can be obtained by solving an equivalent LP problem. In such a case, *mixCoflow* can schedule the deadline coflows by solving the relevant LP problem and leave the remaining bandwidth for non-deadline coflows, which can be scheduled with any existing methods. Extensive trace-driven simulations demonstrate that our *mixCoflow* can significantly reduce the CCTs of non-deadline coflows without incurring increasing on the deadline miss rate for the deadline coflows when compared to the prevailing solutions.

References

[1] R. Xu, W. Li, K. Li, and X. Zhou, "Shaping deadline coflows to accelerate non-deadline coflows," in *Proc. IEEE/ACM IWQoS*, 2018, pp. 1–6.

[2] Q. Pu *et al.*, "Low latency geo-distributed data analytics," in *Proc. ACM SIGCOMM*, 2015, pp. 421–434.

[3] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "CLARINET: WAN-aware optimization for analytics queries," in *Proc. USENIX OSDI*, 2016, pp. 435–450.

[4] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[5] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "DRYAD: Distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, 2007.

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "SPARK: Cluster computing with working sets," in *Proc. Usenix HotCloud*, 2010, pp. 1–8.

[7] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. ACM SIGCOMM*, 2011, pp. 98–109.

[8] M. Chowdhury and I. Stoica, "CoFlow: A networking abstraction for cluster applications," in *Proc. ACM Workshop Hot Topics Netw.*, 2012, pp. 31–36.

[9] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," in *Proc. ACM SIGCOMM*, 2014, pp. 431–442.

[10] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, 2010.

[11] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM*, 2011, pp. 50–61.

[12] Z. Hu, B. Li, C. Chen, and X. Ke, "FlowTime: Dynamic scheduling of deadline-aware workflows and ad-hoc jobs," in *Proc. IEEE ICDCS*, 2018, pp. 929–938.

[13] S. A. Jyothi *et al.*, "Morpheus: Towards automated SLOs for enterprise clusters," in *Proc. USENIX OSDI*, 2016, pp. 117–134.

[14] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. ACM SIGCOMM*, 2015, pp. 393–406.

[15] S. Luo *et al.*, "Minimizing average coflow completion time with decentralized scheduling," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2015, pp. 307–312.

[16] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3366–3380, Feb. 2016.

[17] 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. ACM SIGCOMM*, 2016, pp. 160–173.

[18] W. Wang, S. Ma, B. Li, and B. Li, "Coflex: Navigating the fairness-efficiency tradeoff for coflow scheduling," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[19] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max–min fairness," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[20] Y. Li *et al.*, "Efficient online coflow routing and scheduling," in *Proc. ACM MobiHoc*, 2016, pp. 161–170.

[21] S. Ma, J. Jiang, B. Li, and B. Li, "CHRONOS: Meeting coflow deadlines in data center networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2016, pp. 1–6.

[22] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," in *Proc. ACM SIGCOMM*, Chicago, IL, USA, 2014, pp. 443–454.

[23] R. Meyer, "A class of nonlinear integer programs solvable by a single linear program," *SIAM J. Control Optim.*, vol. 15, no. 6, pp. 935–946, 1977.

[24] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, 2013.

[25] L. Popa *et al.*, "FairCloud: Sharing the network in cloud computing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 187–198, 2012.

[26] R. N. Mysore *et al.*, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, 2009.

[27] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, 2009.

[28] *Facebook Hive/MapReduce Trace*. Accessed: Dec. 18, 2017. [Online]. Available: https://github.com/coflow/coflow-benchmark

[29] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[30] L. Chen, K. Chen, W. Bai, and M. Alizadeh, "Scheduling mix-flows in commodity datacenters with Karuna," in *Proc. ACM SIGCOMM*, 2016, pp. 174–187.

[31] E. D. Andersen and K. D. Andersen, "The Mosek interior point optimizer for linear programming: An implementation of the homogeneous algorithm," in *High Performance Optimization*. Boston, MA, USA: Springer, 2000, pp. 197–232.

[32] *IBM Ilog Cplex Optimizer*. Accessed: Jan. 5, 2018. [Online]. Available: https://goo.gl/jyvDuV

[33] R. K. Jain, D.-M. W. Chiu, and W. R. Hawe, "A quantitative measure of fairness and discrimination," 1998. [Online]. Available: arxiv.cs.NI/9809099.

[34] P. Poupart *et al.*, "Online flow size prediction for improved network routing," in *Proc. IEEE ICNP*, 2016, pp. 1–6.

[35] V. Đukić, S. A. Jyothi, B. Karlaš, M. Owaida, C. Zhang, and A. Singla, "Is advance knowledge of flow sizes a plausible assumption?" in *Proc. USENIX NSDI*, 2019, pp. 565–580.

[36] R. Gandhi, D. Xie, and Y. C. Hu, "PIKACHU: How to rebalance load in optimizing mapreduce on heterogeneous clusters," in *Proc. USENIX ATC*, 2013, pp. 61–66.

[37] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. USNIEX NSDI*, 2012, pp. 253–266.

[38] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 115–126, 2012.

[39] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM*, 2012, pp. 127–138.

[40] T. Wang, H. Xu, and F. Liu, "AEMON: Information-agnostic mix-flow scheduling in data center networks," in *Proc. ACM ApNet*, 2017, pp. 106–112.

[41] J. Guo, F. Liu, J. C. Lui, and H. Jin, "Fair network bandwidth allocation in IaaS datacenters via a cooperative game approach," *IEEE/ACM Trans. Netw.*, vol. 24, no. 2, pp. 873–886, Apr. 2016.

[42] J. Guo, F. Liu, T. Wang, and J. C. Lui, "Pricing intra-datacenter networks with over-committed bandwidth guarantee," in *Proc. USENIX NSDI*, 2017, pp. 69–81.

[43] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Jun. 2017.

[44] X. Yi, F. Liu, Z. Li, and H. Jin, "Flexible instance: Meeting deadlines of delay tolerant jobs in the cloud with dynamic pricing," in *Proc. IEEE ICDCS*, 2016, pp. 415–424.

[45] F. Liu, J. Guo, X. Huang, and J. C. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.

[46] Y. Zhao *et al.*, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE INFOCOM*, 2015, pp. 424–432.

[47] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proc. ACM SIGCOMM*, 2018, pp. 16–29.

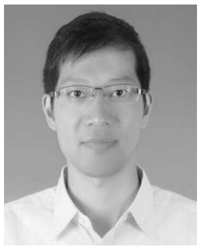
[48] W. Li, X. Yuan, K. Li, H. Qi, and X. Zhou, "Leveraging endpoint flexibility when scheduling coflows across geo-distributed datacenters," in *Proc. IEEE INFOCOM*, 2018, pp. 873–881.

[49] L. Wang, W. Wang, and B. Li, "UTOPIA: Near-optimal coflow scheduling with isolation guarantee," in *Proc. IEEE INFOCOM*, 2018, pp. 891–899.

Renhai Xu received the B.E. and M.S. degrees from the School of Computer Science and Technology, Dalian University of Technology, China, in 2014 and 2017, respectively. He is currently pursuing the Ph.D. degree with the College of Intelligence and Computing, Tianjin University, China. His research interests include datacenter networks and cloud computing.

Wenxin Li received the B.E. and Ph.D. degrees from the School of Computer Science and Technology, Dalian University of Technology in 2012 and 2018, respectively. He is currently a Postdoctoral Researcher with the Hong Kong University of Science and Technology. From May 2014 to May 2015, he was a Research Assistant with the National University of Defense Technology. From October 2016 to September 2017, he was a visiting student with the Department of Electrical and Computer Engineering, University of Toronto. His research interests include datacenter networks and cloud computing.

Keqiu Li (Senior Member, IEEE) received the bachelor's and master's degrees from the Department of Applied Mathematics, Dalian University of Technology, China, in 1994 and 1997, respectively, and the Ph.D. degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology in 2005. He also has a two-year Postdoctoral experience with the University of Tokyo, Japan. He was a Professor with the Dalian University of Technology from 2007 to 2015. He is currently a Professor with the College of Intelligence and Computing, Tianjin University, China. He has published more than 100 technical papers, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *ACM Transactions on Internet Technology*, and *ACM Transactions on Multimedia Computing, Communications, and Applications*. His research interests include Internet technology, data center networks, cloud computing, and wireless networks. He is an Associate Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS and IEEE TRANSACTIONS ON COMPUTERS.



Xiaobo Zhou (Senior Member, IEEE) received the B.Sc. degree in electronic information science and technology from the University of Science and Technology of China, Hefei, China, in 2007, the M.E. degree in computer application technology from the Graduate University of Chinese Academy of Science, Beijing, China, in 2010, and the Ph.D. degree from the School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Japan, in 2013. He was a Researcher with the Centre for Wireless Communications, University of Oulu, Finland, from 2014 to 2015. He is currently an Associate Professor with the College of Intelligence and Computing, Tianjin University. His research interests include joint source-channel coding, cooperative wireless communications, network information theory, cloud computing, and software defined networking.

Heng Qi (Member, IEEE) received the bachelor's degree from Hunan University in 2004, and the master's and doctorate degrees from the Dalian University of Technology, China, in 2006 and 2012, respectively, where he was a Lecture with the School of Computer Science and Technology. He servered as a Software Engineer with GlobalLogic-3CIS from 2006 to 2008. He has published more than 20 technical papers in international journals and conferences, including *ACM Transactions on Multimedia Computing, Communications and Applications* and *Pattern Recognition*. His research interests include computer network, multimedia computing, and mobile cloud computing.