Endpoint-Flexible Coflow Scheduling Across Geo-Distributed Datacenters

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

Abstract—Over the last decade, we have witnessed growing data volumes generated and stored across geographically distributed datacenters. Processing such geo-distributed datasets may suffer from significant slowdown as the underlying network flows have to go through the inter-datacenter networks with relatively low and highly heterogeneous available link bandwidth. Thus, optimizing the transmissions of inter-datacenter flows, especially *coflows* that capture application-level semantics, is important for improving the communication performance of such geo-distributed applications. However, prior solutions on coflow scheduling have significant limitations: they schedule coflows with already-fixed endpoints of flows, making them insufficient to optimize the coflow completion time (CCT). In this article, we focus on the problem of jointly considering endpoint placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters. To solve this problem without any prior knowledge of coflow arrivals, we present a coflow-aware optimization framework called *SmartCoflow*. In *SmartCoflow*, we first apply an approximate algorithm to obtain the endpoint placement and scheduling decisions for a single coflow. Based on the single-coflow solution, we then develop an efficient online algorithm to handle the dynamically arrived coflows. Through rigorous theoretical analysis, we prove that *SmartCoflow* has a non-trivial competitive ratio. We also extend *SmartCoflow* to incorporate various design choices or requirements of applications and operators, such as enforcing an inter-datacenter bandwidth usage budget and considering coflow deadline. Through experimental results from testbed implementation and trace-driven simulations, we demonstrate that *SmartCoflow* can reduce the average CCT, lower bandwidth usage, and improve coflow deadline meet rate, when compared to the state-of-the-art scheduling-only method.

Index Terms—Inter-datacenter, coflow scheduling, CCT, deadline, endpoint flexibility

1 Introduction

To ENABLE end-users can access to services with low latency, large organizations, such as Google, Microsoft, and Amazon, build many 10s-100s of datacenters *all around the globe* [2], [3], [4]. As a result, large volumes of data, e.g., end-user sessions logs and server monitoring logs, will be generated and stored at geographically distributed datacenters. On the other hand, many applications require a global view of these data to compute exact analytics query results [5] or to build accurate machine learning models [6]. Instead of inefficiently aggregating all the data required for the computation of an application/job to a single datacenter, a recent trend is to leave data *in-place* and execute jobs over geo-distributed datasets directly [5], [7], [8].

- *W. Li is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. E-mail: toliwenxin@gmail.com.*
- *X. Yuan is with the School of Computing and Informatics, University of Louisiana at Lafayette, Lafayette, LA 70503. E-mail: xu.yuan@louisiana.edu.*
- *K. Li, X. Zhou, and R. Xu are with the Tianjin Key Laboratory of Advanced Networking (TANK), College of Intelligence and Computing, Tianjin University, Tianjin 300350, China. E-mail: {keqiu, xiaobo.zhou, xurenhai}@tju.edu.cn.*
- *H. Qi is with the School of Computer Science and Technology, Dalian University of Technology, No 2, Linggong Road, Dalian 116023, China. E-mail: hengqi@dlut.edu.cn.*

Manuscript received 20 Oct. 2019; revised 7 Mar. 2020; accepted 25 Apr. 2020. Date of publication 6 May 2020; date of current version 21 May 2020. (Corresponding author: Keqiu Li.)
Recommended for acceptance by W. Yu.
Digital Object Identifier no. 10.1109/TPDS.2020.2992615

A common denominator of these jobs is that they produce a set of network flows to transfer the intermediate data between successive computation stages (e.g., map and reduce)—known as *coflows* [9]. The coflow abstraction captures the *all-of-nothing* communication requirements of data-parallel jobs: *all* flows must be finished before a coflow is considered complete. Traditionally, when a job is running within a single datacenter, all flows in a coflow are restricted in the intra-datacenter network. However, in the geo-distributed setting, the flows in a coflow necessarily have to traverse the inter-datacenter links. The available bandwidth on those inter-datacenter links is limited and can vary significantly across different links [5], [6], [10]. Meanwhile, the data volume of flows in a coflow could be enormous for a geo-distributed data-parallel job [5], yet a coflow's completion time (CCT) can account for more than 50 percent of job completion time [11], [12]. Moreover, as computation devices are witnessed to get faster [13], [14], communication is more likely to become the performance bottleneck for a job.

Therefore, optimizing the CCTs becomes critical to improve the performance of jobs running across geo-distributed datacenters. Many existing works [11], [12], [15], [16], [17], [18] have focused on reducing such CCTs by efficiently scheduling the network flows within each coflow. Unfortunately, they are insufficient to optimize the CCT of a coflow as the endpoints of all flows are assumed to be fixed. In other words, they do not consider the impact of flow endpoints (i.e., destinations) on the CCT. As a result, coflow scheduling can have little space to take effect for optimizing the CCTs of coflows.

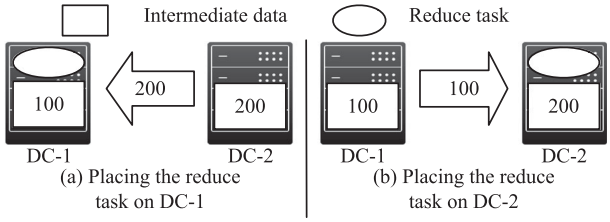


Fig. 1. An illustrating example, where a coflow has one network flow that transfers 200 units of data from DC-2 to DC-1 when placing the reduce task on DC-1. On the other hand, if placing the reduce task on DC-2, this coflow will only need to transfer a 100-unit flow from DC-1 to DC-2.

In fact, coflows do not require the destinations of their flows to be in specific locations as long as certain constraints are satisfied. The reason is that the endpoints of flows in a coflow are closely correlated to the reduce tasks of the corresponding job, while each reduce task can be placed in the machine of any datacenter that has available computational resources. In this case, we can flexibly select the endpoints of flows in a coflow, by changing the locations of reduce tasks. As an example, Fig. 1 demonstrates that changing the locations of reduce tasks will lead to different endpoint placement strategies. Needless to say, a better distribution of endpoints can significantly reduce the data sizes of flows in a coflow, and thus directly speeds up the completion of coflows. Hence, there is a pressing need to leverage endpoint flexibility when scheduling coflows across geo-distributed datacenters.

In this paper, we focus on the problem of jointly considering endpoint placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters. We first develop the mathematical model and formulate a mixed integer programming problem to seamlessly integrate endpoint placement and coflow scheduling for optimizing the average CCT of coflows. In this program, we take into account heterogeneous link bandwidth capacities, different coflow arrival times, and available computation resources of datacenters. Since the information about future coflows is usually unknown in advance, it is challenging to obtain the optimal solution for this problem.

Therefore, we present *SmartCoflow*, an online coflow-aware optimization framework. In *SmartCoflow*, we first propose an approximate algorithm to derive the endpoint placement and scheduling decisions for one single coflow. Based on the single-coflow solution, we then propose an efficient online algorithm to minimize the average CCT when multiple coflows dynamically arrive at the network. Without requiring the prior knowledge of coflow arrivals, *SmartCoflow* has been proved to guarantee a theoretical upper bound for the average CCT.

We have also extended *SmartCoflow* to incorporate two practical requirements. The first requirement is to save inter-datacenter bandwidth as the bytes transferred over the inter-datacenter network have important cost implications [5], [19]. To this end, we introduce a simple knob to ensure fast coflow completion and reasonable bandwidth usage. The second requirement is to guarantee coflow deadline. For this requirement, we first let the single-coflow solution in *SmartCoflow* be an admission control rule. Once admitted, we then sort all active coflows based on their deadline and use the minimum amount of bandwidth to guarantee each coflow's deadline.

We proceed to implement *SmartCoflow* as a real-world coflow-aware scheduler that enforces our endpoint placement and scheduling strategies in the Varys coflow scheduling framework [15], [20]. Finally, to evaluate the performance of *SmartCoflow*, we use a small-scale testbed implementation based on Google's Cloud Compute Engine, and also conduct large-scale simulations with a real-world data trace collected from Facebook. The experimental results demonstrate that *SmartCoflow* can reduce the average CCT by up to 28.6 percent, lower inter-datacenter bandwidth usage nearly by 20 percent, accommodate 20.3 percent more coflows with deadlines guaranteed, compared to the state-of-the-art scheduling-only method [15].

The main contributions of this paper are as follows:

- We study the problem of jointly considering endpoint placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters. We develop the mathematical model and formulate a mixed integer programming to characterize the intertwined relationship between endpoint placement and coflow scheduling to reveal their impact on the average CCT of coflows.
- We present a new coflow-aware optimization framework, *SmartCoflow*, to solve the studied problem. In *SmartCoflow*, we develop fast and efficient online algorithms to derive the endpoint placement and scheduling decisions for coflows that dynamically arrive at the network.
- We conduct rigorous theoretical analysis to demonstrate that *SmartCoflow* can achieve a good competitive ratio in minimizing the average CCT of coflows. We also extend *SmartCoflow* to make our solution be mindful of inter-datacenter bandwidth usage as well as coflow deadline.
- We conduct a small-scale testbed implementation and extensive trace-driven simulations to evaluate the performance of *SmartCoflow*, in terms of reducing average CCT of coflows, lowering inter-datacenter bandwidth usage and guaranteeing coflow deadline.

The rest of this paper is organized as follows. In Section 2, we show some background and describe our problem for this paper. In Section 3, we develop the system model and present our problem formulation. We show an overview of *SmartCoflow* in Section 4 and the algorithm details in Section 5. We present the extension of our *SmartCoflow* framework in Section 6. The implementation details and experiment results are presented in Section 7. We discuss the limitations of our work in Section 8. Section 9 discusses the related work and Section 10 concludes this paper.

2 Background and Problem Statement

2.1 Background

Modern organizations have a planetary footprint. Data is born and generated in multiple datacenters, *all around the globe*. Examples of such data include user activity logs, server monitor logs, and URL clicks. To process such geo-distributed data, two types of applications have become increasingly common. (1) *Geo-distributed Data Analytics (GDA)* [5], [7], [8] refers to leave data in-place and execute a job on geo-distributed data

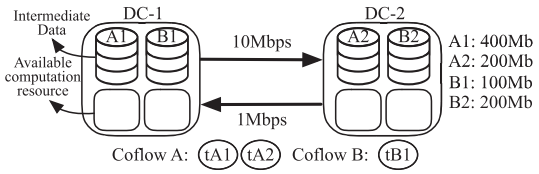
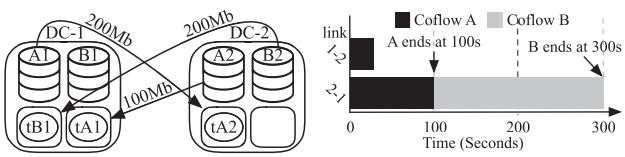


Fig. 2. An example with two coflows (i.e., A and B) and two datacenters (i.e., DC-1 and DC-2). For coflow A, there are two pieces of intermediate data (i.e., A1=400Mb and A2=200Mb) and two reduce tasks (i.e., tA1 and tA2). For coflow B, there is only one reduce task (tB1), and the intermediate data on these two datacenters are B1=100Mb and B2=200Mb. The bandwidth capacity of link from DC-1 to DC-2 is 10Mbps, while the link from DC-1 to DC-2 has 1Mbps bandwidth capacity. Both DC-1 to DC-2 have 2 computing slots that are available for accommodating the reduce tasks.

directly with popular data analytic frameworks such as MapReduce [21] and Spark [22], [23]. For example, an operator would like to submit a job to query the geo-distributed user logs to get the top-10 URLs by the number of clicks. (2) *Geo-distributed Machine Learning (GDML)* [6], [24] refers to machine learning jobs that run over the geo-distributed data to build accurate models and leverage WANs to communicate between datacenters. For example, an image classification system would use pictures located at different datacenters as its input data to keep improving its classification using the pictures generated continuously all over the world.

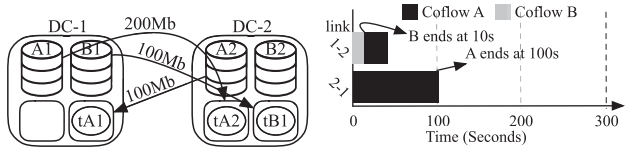
In many GDA and GDML jobs, large volumes of intermediate data need to be transferred across the inter-datacenter networks. Existing studies [9], [15], [26] have demonstrated that the intermediate data transfers of GDA jobs can be modeled as *coflows*. On the other hand, GDML jobs can generate coflows as well. More specifically, in a GDML job, the workers need to iteratively push (pull) parameter updates to (from) the parameter servers to refine the ML model. If Bulk Synchronous Parallel (BSP) is in use, the flows in each parameter push or pull phase can be treated as a coflow, because there is a strict barrier at the end of each phase. And, the progress of each parameter exchange phase is determined by the collective behavior of its flow transfers rather than individual ones. Note that there are indeed other synchronization models, i.e., stale synchronous parallel (SSP) [27] and total synchronous parallel (TAP) [28]. However, BSP is the most commonly used one in production [29].

The flow volume in a coflow could be enormous, accounting for 55 percent of the input data size on average in production analytics jobs [5], despite local aggregation of the map outputs for associative reduce operations [30]. Whereas inter-datacenter networks have a different network model as compared to traditional datacenter networks. In many recent



(a) Equal spread [24] (b) Optimal scheduling up on (a)

Fig. 3. Scheduling-only scheme, where reduce tasks are equally spread across datacenters for each coflow (as shown in (a)), leading to the optimal coflow scheduling results shown in (b). With this scheme, the average CCT is 200s.



(a) Optimal reduce task placement (b) Optimal scheduling up on (a)

Fig. 4. The optimal scheme, where reduce task placement and scheduling are jointly considered. The average CCT is reduced to 55s, compared to the above scheduling-only scheme.

coflow studies [15], [16], [17], [18], [26], [31], datacenter networks are usually abstracted as a big switch, where only the ingress and egress links at the end-hosts are considered as the bottleneck. Nevertheless, it is no longer valid for inter-datacenter networks, as the available inter-datacenter link bandwidth is significantly lower than the edge capacity [5], [6], [10]. So, optimizing the CCTs of coflows is critical for enhancing the performance of application-level jobs, especially when coflows need to go through the inter-datacenter networks.

2.2 Problem Statement

As mentioned in the previous section, the reduce task placement is closely related to the CCT of a coflow. Hence, we are motivated to combine the reduce task placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters. More specifically, we study the problem of *where* to place the reduce tasks[1] to derive a better endpoint distribution for the flows in each coflow, *when* to start these flows, and *at what rate* to serve them on the inter-datacenter links, to minimize the average CCT of coflows.

For a better intuition of our problem, we use a motivating example with two coflows and two geo-distributed datacenters; detailed settings for this example are shown in Fig. 2. As a reference point, the optimal average CCT for this example is 55s. Fig. 3 first illustrates a scheduling-only scheme. Such scheme only focuses on coflow scheduling without considering the optimization of reduce task placement, implying that the endpoints of each flow might be placed improperly. In this context, coflow scheduling could have little space to minimize the average CCT. Fig. 3a shows a possible case of reduce task placement with the naive equal spreading method [25], which assigns an equal number of reduce tasks to each datacenter. In such a case, coflow A has two flows: one transfers half of the data A1, i.e., 200 Mb, from DC-1 to DC-2 while another one transfers half of the data A2, i.e., 100 Mb, from DC-2 to DC-1. Coflow B has only one flow that needs to transfer all the data B2 from DC-2 to DC-1. When these two coflows meet at the inter-datacenter links, the optimal solution [15] is to schedule coflow B after coflow A, as shown in Fig. 3b. The CCTs of coflows A and B, achieved by the scheduling-only scheme, are 100s and 300s, respectively. Hence, the average CCT is 200s, which has a 145s gap to the optimal value 55s. The key reason for such large CCT is that these two coflows are congested on the link from DC-2 to DC-1.

Surprisingly, if placing the reduce task tB1 on DC-2 while keeping the locations of tA1 and tA2 (Fig. 4a), the

1. Since the reduce task is directly correlated with the endpoints of flows in a coflow, we will use reduce task and endpoint interchangeably hereafter.

TABLE 1
Important Notations Used Throughout This Article

Symbol	Definition
$\mathcal{N}$	the set of geo-distributed datacenters
$\mathcal{E}$	the set of inter-datacenter links
U_i	the amount of available computing slots in datacenter $i \in \mathcal{N}$
$C_{i,j}$	the bandwidth capacity of inter-datacenter link $l_{ij} \in \mathcal{E}$
$\mathcal{K}$	the set of coflows
t_k	the arrival time of coflow $k \in \mathcal{K}$
T_k	the completion time of coflow $k \in \mathcal{K}$
D_i^k	the amount of intermediate data associated with k stored on i
R_k	the number of reduce tasks associated with k (each task is denoted by p)
$I_{p,i}^k$	a binary variable indicating whether the task p associated with k is assigned to i
$B_{i,j}^k(x)$	the amount of bandwidth allocated to coflow k's flow from i to j, at time x

data sizes of flows can be significantly reduced, i.e., the data size of the flow in coflow B is reduced to 100 Mb. Moreover, coflow B can smartly avoid the bottleneck link from DC-2 to DC-1. In this case, the CCTs of coflows A and B are 100s and 10s respectively, and the average CCT is minimized (Fig. 4b). The implies that both reduce task placement and scheduling must be jointly considered to minimize the average CCT.

The above example looks straightforward with simple settings. But the general problem of jointly considering reduce task placement and scheduling to minimize the average CCT of coflows can be difficult due to the following two challenges. *First*, the placement of reduce tasks will determine the flow size on each inter-datacenter link, which thus directly impacts the scheduling decisions, implying that reduce task placement and scheduling are deeply intertwined with each other. In this case, how to obtain the optimal solution is a challenge. *Second*, the arrival pattern of coflow is usually unknown in advance and is difficult to be accurately predicted. In most practical scenarios, we can only get the information about the coflows that have arrived at the inter-datacenter network. So, how can we guarantee that the current task placement and scheduling decisions will not harm the CCTs of future coflows? This makes another challenge to obtain the optimal solution.

3 Model and Problem Formulation

In this section, we describe the system model and present the problem formulation, with important notations used throughout this paper being listed in Table 1.

3.1 System Model

In our model, we make the following assumptions. First, we assume each reduce task processes the same amount of intermediate data. Thus, the intermediate data on each datacenter should be distributed to other datacenters proportionally to the fractions of reduce tasks that are placed on them. This assumption is similar to that in [5]. Second, we also assume that all reduce tasks have the same resource demand (i.e., one computational slot), which is also a common assumption in literature [10], [32].

Under the assumptions above, we consider a network with multiple geo-distributed datacenters denoted as $\mathcal{N} = \{1, \ldots, N\}$. In this network, there are a set of inter-datacenter links, which are denoted as $\mathcal{E}$. For each link $l_{i,j} \in \mathcal{E}$ between datacenters $i \in \mathcal{N}$ and $j \in \mathcal{N}$, we denote $C_{i,j}$ as its bandwidth capacity. Suppose that there are a set of coflows in the network, which is denoted as $\mathcal{K} = \{1, \ldots, K\}$. Each coflow $k \in \mathcal{K}$ arrives at the inter-datacenter network at time t_k. The information associated with each coflow k is assumed to be known as soon as this coflow arrives,[2] which includes the amount of intermediate data on each datacenter (i.e., $D_i^k, \forall i \in \mathcal{N}$) and the number of reduce tasks (i.e., R_k) to be launched on available computing slots in datacenters. We use U_i to denote the capacity of available computing slots in datacenter $i \in \mathcal{N}$.

Reduce Task Placement Constraints. To indicate the reduce task placement, we denote $I_{p,i}^k$ as whether the pth reduce task associated with coflow $k \in \mathcal{K}$ is placed on datacenter $i \in \mathcal{N}$. Then, we have

$$I_{p,i}^k \in \{0,1\}, \forall i \in \mathcal{N}, \forall k \in \mathcal{K}, \forall p \in \{1, \ldots, R_k\}. \tag{1}$$

Since each reduce task can be processed by only one datacenter, i.e., $\forall i \in \mathcal{N}$, there is only one $I_{p,i}^k = 1$, and thus we have the following constraint:

$$\sum_{i=1}^{N} I_{p,i}^k = 1, \forall k \in \mathcal{K}, \forall p \in \{1, \ldots, R_k\}. \tag{2}$$

Coflow Scheduling Constraints. To indicate the coflow scheduling, we denote $B_{i,j}^k(x)$ as the amount of bandwidth allocated to coflow k for supporting the data transmission between datacenters i and j at time x ($x \geq 0$). Note that $B_{i,j}^k(x)$ can be zero for some x's, implying that the network flow between datacenters i and j is waiting for transmission or there is no such flow, for coflow k.

We denote T^k as the CCT of coflow k. Since all flows in a coflow k must finish transmitting their data between the arrival time t_k and the completion time T^k, we have

$$D_i^k \frac{\sum_{p=1}^{R_k} I_{p,j}^k}{R_k} \leq \sum_{x=t_k}^{t_k+T^k} B_{i,j}^k(x), \forall l_{ij} \in \mathcal{E}, k \in \mathcal{K}, \tag{3}$$

where $D_i^k \frac{\sum_{p=1}^{R_k} I_{p,j}^k}{R_k}$ can calculate the data size of the flow traversing link l_{ij} [5]. Constraint (3) means that T^k is determined by when the last flow of coflow k finishes.

Capacity Constraints. When performing reduce task placement and coflow scheduling, both the capacities of computational resource and link bandwidth should be satisfied. Specifically, we have the following two constraints:

2. This assumption is reasonable because many recent studies (e.g., [11], [12], [15], [31]) assume that they know all the information about a coflow. Besides, the information associated with a coflow is readily available in data-parallel frameworks. For instance, in Spark, the intermediate data can be obtained through the MapOutputTracker [10], and the number of reduce tasks can also be known through the TaskSet in Spark DAG scheduler [23].

$$\sum_{k=1}^{K}\sum_{p=1}^{R_k} I_{p,i}^k \le U_i, i \in \mathcal{N}, \tag{4}$$

$$\sum_{k=1}^{K} B_{i,j}^k(x) \le C_{i,j}, \forall l_{ij} \in \mathcal{E}, \forall x \ge 0. \tag{5}$$

Constraint (4) means that the total number of reduce tasks assigned to i should not exceed U_i, which is the total number of available computing slots in datacenter $i \in \mathcal{N}$. Meanwhile, constraint (5) ensures that the summation of bandwidth allocated to all flows on a link l_{ij} should not exceed the link bandwidth capacity $C_{i,j}$.

3.2 Problem Formulation

We now formulate the problem of jointly considering reduce task placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters, as shown in the following problem *P1*:

$$\begin{aligned} &\underset{\{I_{p,i}^k\},\{B_{i,j}^k(x)\}}{\text{Minimize}} \quad \frac{1}{K}\sum_{k=1}^{K} T^k \\ &\text{Subject to: Eqs. } (1), (2), (3), (4), (5), \end{aligned} \tag{6}$$

where the objective function (6) represents the minimum of the average CCT across all coflows. In problem *P1*, the reduce task placement (i.e., $I_{p,i}^k$) and scheduling (i.e., $B_{i,j}^k(x)$) decisions of current coflows will impact that of future coflows. To solve *P1*, one may design an offline optimal algorithm, which, however, encounters two challenges. *First*, Problem *P1* is an mixed integer programming which is NP-hard (as shown in the following Theorem 1), making it very hard to obtain the optimal solution. *Second*, The offline algorithm inevitably relies on a prior knowledge of the intermediate data (i.e., D_i^k) and the number of reduce tasks (i.e., R_k) for future coflows. Such knowledge can only be known at the arrival of a coflow, yet is difficult to be predicted accurately. Thus, an online algorithm is desired to solve problem *P1* more efficiently and handle the dynamically arrived coflows. In the following, we present *SmartCoflow* to address the challenges of problem *P1*.

Theorem 1. *Problem* P1 *is NP-hard.*

Proof. We will prove this theorem by demonstrating that the problem *P1* can generalize some well-known NP-hard problems. *On the one hand*, when there is only one datacenter, i.e., $N = 1$, all reduce tasks can only be placed in this datacenter. In such a case, it is easy to see that one can reduce the non-preemptive Single-Machine Scheduling Problem (denoted as SMSP for brevity) with release dates to our problem *P1*. The implication is that *P1* is at least as hard as the SMSP problem. Since SMSP has been proved to be NP-hard [33], *P1* is NP-hard as well. *On the other hand*, the well-known NP-hard Coflow Scheduling Problem (CSP) [15] of minimizing the average CCT is a special case of our problem *P1*. More specifically, if the reduce task placements are fixed (i.e., $I_{p,i}^k$ is known), then *P1* is equivalent to a CSP instance where there are N machines and K coflows with each coflow having the traffic matrix $\boldsymbol{D}^k = [d_{i,j}]_{N\times N}$ ($d_{i,j} = D_i^k \sum_{p=1}^{R_k} I_{p,j}^k / R_k$). Since the special case of *P1* is NP-hard, the general form of *P1* is even harder. □

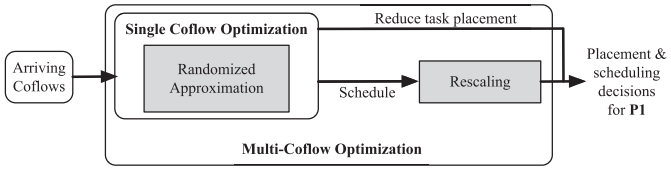


Fig. 5. The overview of *SmartCoflow* optimization framework.

4 *SmartCoflow* in a Nutshell

SmartCoflow is an online coflow-aware optimization framework that optimizes the average CCT of coflows over the inter-datacenter networks by coordinating reduce task placement and coflow scheduling. In this section, we present a brief overview of *SmartCoflow* to help the reader follow the analysis and design of our scheduling algorithm details.

4.1 Desirable Properties

We first show the desired properties of *SmartCoflow*:

- *Practicality*: *SmartCoflow* is necessarily an online system, which means that it must quickly decide the reduce task placement and scheduling decisions for a coflow once the coflow arrives. Hence, the *SmartCoflow* algorithms must run in real-time with low time complexity.
- *Performance guarantee*: *SmartCoflow* must be able to provide theoretical performance guarantees, such that the average CCT of coflows can be bounded with an upper bound.
- *Flexibility*: We expect *SmartCoflow* to be a flexible design, which means that it can easily be extended to incorporate various optimization goals, such as meeting coflow deadlines and reducing inter-datacenter bandwidth usage.

4.2 Design Overview

Fig. 5 shows an overview of *SmartCoflow* optimization framework. In particular, once a coflow arrives at the network, we will formulate an integer linear programming (ILP) problem. Given the ILP for a certain coflow, *SmartCoflow* will then apply a randomized approximate algorithm to solve it for deriving the reduce task placement and scheduling decisions for this coflow. This algorithm first relaxes the ILP into linear programming (LP). It then uses a randomized rounding technique to enforce the optimal solution of the LP to be a feasible one of the original ILP.

Based on the single-coflow solution, *SmartCoflow* rescales the bandwidth allocated to all existing coflows while keeping the reduce task placement unchanged. The rationale is that avoiding frequent reduce task placement will make *SmartCoflow* more efficient and more implementable in practical scenarios. Besides, when rescaling bandwidth, *SmartCoflow* attempts to assign more bandwidth to large coflows while giving relatively less bandwidth to small coflows.

5 Algorithm Details

In this section, we first present a randomized approximate algorithm to minimize the CCT by focusing on one single coflow, and then extend this algorithm to handle the online multi-coflow scenarios.

5.1 Minimizing Single Coflow Completion Time

If there is only one single coflow in the network, the problem can be formulated as following (denoted as *P2*):

$$\underset{\{I_{p,i}\}}{\text{Minimize}} \quad T \tag{7}$$

$$\text{Subject to: } I_{p,i} \in \{0,1\}, \forall i \in \mathcal{N}, \forall p \in \{1,\dots,R\}, \tag{8}$$

$$\sum_{i=1}^{N} I_{p,i} = 1, \forall p \in \{1,\dots,R\}, \tag{9}$$

$$\sum_{p=1}^{R} I_{p,i} \le U_i, i \in \mathcal{N}, \tag{10}$$

$$B_{i,j} \le C_{i,j}, \forall l_{ij} \in \mathcal{E}, \tag{11}$$

$$B_{i,j}T = D_i \frac{\sum_{p=1}^{R} I_{p,j}}{R}, \forall l_{ij} \in \mathcal{E}, \tag{12}$$

where T is the CCT of this coflow. Parameters R and D_i are the number of reduce tasks and the amount of intermediate data on datacenter i, respectively. $I_{p,i}$ and $B_{i,j}$ are the reduce task placement and scheduling decisions, respectively. Note that in our mathematical model (in Section 3), the bandwidth is a function of time, but it is a constant value in problem *P2*. This implies that if a flow traverses link l_{ij}, the bandwidth usage should be equal to $B_{i,j}$ for transmission or zero if this flow finishes. In such a case, $B_{i,j}$ can be calculated directly by fixing $I_{p,i}$, as shown in Eq. (12).

Theorem 2. *Problem* P2 *is NP-hard.*

Proof. By taking an in-depth analysis of the structure of problem *P2*, we find that it only contains the decision variable $I_{p,i}$ and can be re-organized as the following equivalent problem:

$$\underset{\{I_{p,i}\}}{\text{Maximize}} -\max_{l_{i,j}} \frac{D_i}{RC_{i,j}} \sum_{p=1}^{R} I_{p,j} \text{ S.t.: Eqs. } (8),(9),(10).$$

This problem can be interpreted as a Generalized Assignment Problem (GAP) in the following way. The datacenters and the reduce tasks can be viewed as agents and tasks, respectively. Each task has a cost of 1. Each agent j has a budget of U_j. Any task can be assigned to any agent, incurring some cost and profit. The sum of the costs of tasks assigned to each agent cannot exceed the corresponding budget. The profit of assigning task p to agent j can be defined as $\Re_{p,j} = \max_{\{i|l_{i,j}\in\mathcal{E}\}} - D_i/(RC_{i,j})$, the profit of task p can then be computed as $\max_j \Re_{p,j}$. It is required to find an assignment in which all agents do not exceed their budget, and the total profit of all tasks is maximized. Since the GAP) has proven to be NP-hard [34], our problem *P2* is also NP-hard. □

To solve problem *P2*, we first relax $I_{p,i}$ into a continuous variable and accordingly obtain an LP, as shown in the following problem *P3*:

$$\begin{aligned} &\underset{\{I_{p,i}\}}{\text{Minimize}} \quad T \\ &\text{Subject to: } 0 \le I_{p,i} \le 1, i \in \mathcal{N}, \forall p \in \{1,\dots,R\}, \\ &\qquad \text{Eqs. } (9),(10),(11),(12), \end{aligned} \tag{13}$$

which can be solved efficiently with standard linear programming solvers, such as `Breeze` [35]. Since $I_{p,i}$ is relaxed into continuous variable, the solution of *P3* may not give a feasible solution for the ILP *P2*. To find a feasible solution, we then propose to use another technique—*rounding*.

Algorithm 1. Minimize Single Coflow CCT

Input: Coflow information: $\{\{D_i\}, R\}$
Output: A feasible solution for this coflow
1: Calculate the optimal solution (I, B, T) of problem **P3**.
2: **for** $p = 1, 2, \dots, R$ **do**
3: Sample an i with probability $I_{p,i}$ from $\{1, 2, \dots N\}$.
4: If $U_i = 0$, repeat step 1. Otherwise, set $I_{p,i} = 1$ and update $U_i = U_i - 1$.
5: **end for**
6: Find a smallest real number $\lambda \ge 1$ to make $\{\{I_{p,i}\}, \{\frac{B_{i,j}}{\lambda}\}\}$ become a feasible solution to problem **P2**.
7: **return** $\{\{I_{p,i}\}, \{\frac{B_{i,j}}{\lambda}\}\}$

The whole procedure to minimize the single coflow CCT is summarized in Algorithm 1. Our Algorithm starts from the optimal solution of LP *P3* (Step 1). Then, in the *for* loop (Step 2-5), it chooses a datacenter for each reduce task independently. Specifically, it samples a datacenter i for each reduce task p with probability $I_{p,i}$. Finally, it rescales the bandwidth to make sure the solution is feasible (Step 6). To verify that Algorithm 1 can approach a CCT that is near to the optimal one of the ILP *P2*, we first give *a lower bound* of the minimum CCT of the coflow. Then, we provide *an upper bound* of the CCT achieved by Algorithm 1.

Theorem 3 (Lower bound of the optimal CCT of *P2*). *Define T_{P2} and T_{P3} as the optimal CCTs for problems* P2 *and* P3, *respectively. Then, we have $T_{P3} \le T_{P2}$.*

Proof. We mainly focus on proving that *P3* is a relaxation of *P2* because such relaxation directly leads to $T_{P3} \le T_{P2}$. The relaxation means that: for any feasible solution $S := \{\{I_{p,i}\}, \{B_{i,j}\}\}$ of *P2*, there always exists a feasible solution of *P3* to make the objective value in *P3* equal to T_S (the CCT under solution *S*). To prove it, we define a solution of *P3* as follows: (1) set $T = T_S$; (2) for each p, set $I_{p,i} = 1$, and $I_{p,i'} = 0$ for all $i' \ne i$; (3) for each l_{ij}, set $B_{i,j} = \frac{D_i \sum_{p=1}^{R} I_{p,j}}{RT}$.

Since S is a feasible solution of *P2*, it is obvious that $\sum_{p=1}^{R} I_{p,i} \le U_i$. Therefore, it remains only one constraint (11) to be verified. With the definition of completion time, we have $\sum_0^T B_{i,j}(x) \ge D_i \sum_{p=1}^{R} I_{p,j}/R$, which implies that $B_{i,j} = D_i \sum_{p=1}^{R} I_{p,j}/RT \le \sum_0^T B_{i,j}(x)/T$. Again, by the fact that S is a feasible solution, we have $\sum_0^T B_{i,j}(x) \le TC_{i,j}$. Then, we get $B_{i,j} \le \sum_0^T B_{i,j}(x)/T \le C_{i,j}$. This implies the relaxation, and thus the theorem is proved. □

Theorem 4 (Upper bound of the CCT achieved by Algorithm 1). *Algorithm 1 achieves a CCT that is at most $2\ln 4M$ times the optimal CCT with probability at least $\frac{3}{4}$, where $M = |\mathcal{E}|$*

is the total number of inter-datacenter links in $\mathcal{E}$. More specifically, Algorithm 1 guarantees its competitive ratio ρ for problem P2 *to satisfy that $Pr(\rho > 2\ln 4M) \le 1/4$.*

Proof. For each $l_{ij} \in \mathcal{E}$, define

$$\rho_{ij} = \frac{B_{i,j}}{C_{i,j}} = \frac{\frac{D_i}{RT}\sum_{p=1}^{R} 1(I_{p,i}=1)}{C_{i,j}},$$

where $1(I_{p,i}=1)$ is an indicator variable that is 1 if $I_{p,i}=1$ and 0 otherwise. Combining Theorem 3, the competitive ratio of Algorithm 1 can be calculated as $\rho = \max_{l_{ij}} \rho_{ij}$. Since $Pr(\rho > 2\ln 4M) \le \sum_{l_{ij}\in\mathcal{E}} Pr(\rho_{ij} > 2\ln 4M)$, we fix l_{ij} and focus on the proof of $Pr(\rho_{ij} > 2\ln 4M) \le \frac{1}{4M}$. Let $x_p := \frac{D_i}{RT} 1(I_{p,i}=1)$ be a random variable, and define $x := \sum_{p=1}^{R} x_p$. These random variables have the following properties: 1) all x_p's are independent; 2) x_p is either 0 or $\frac{D_i}{RT}$, and $Pr(x_p=\frac{D_i}{RT})=I_{p,i}$; 3) $E(x) = \frac{D_i\sum_{p=1}^{R} I_{p,i}}{RT} \le C_{i,j}$, by Eqs. (11) and (12); 4) If $Pr(x_p=\frac{D_i}{RT}) > 0$ then $\frac{D_i}{RT} \le C_{i,j}$, by Eqs. (10), (11), and (12).

Let $\theta \ge 2e$ be a real number, and we now focus on the proof of $Pr(x > C_{i,j}\theta) \le \exp(-\frac{\theta}{2})$. To this end, let $\alpha > 0$ be a fixed parameter. By using Markov's inequality, we have

$$\begin{aligned} Pr(x > C_{i,j}\theta) &= Pr(\alpha x > \alpha C_{i,j}\theta) \\ &= Pr(\exp(\alpha x) > \exp(\alpha C_{i,j}\theta)) \\ &\le \exp(-\alpha C_{i,j}\theta)\cdot E(\exp(\alpha x)) \\ &= \exp(-\alpha C_{i,j}\theta)\prod_{p=1}^{R} E(\exp(\alpha x_p)). \end{aligned} \tag{14}$$

Now, we analyze $E(\exp(\alpha x_p))$. Using the definition of mathematical expectation, we have

$$\begin{aligned} E(\exp(\alpha x_p)) &= (1-I_{p,i}) + I_{p,i}\exp(\alpha D_i/RT) \\ &= 1 + I_{p,i}(\exp(\alpha D_i/RT) - 1) \\ &\le \exp(I_{p,i}(\exp(\alpha D_i/RT) - 1)), \end{aligned}$$

where the last inequality is derived from the fact that $1 + a \le \exp(a)$ for $a > 0$. With the above formula, we have

$$\prod_{p=1}^{R} E(\exp(\alpha x_p)) \le \exp\left(\sum_{p=1}^{R} I_{p,i}(\exp(\alpha D_i/RT) - 1)\right).$$

For all p, we choose α to satisfy $\exp(\alpha D_i/RT) \le 1 + \frac{1}{2}\alpha\theta D_i/RT$. We will show the existence and give the value of such an α later. Combining the property of α and the fact that $E(x) \le C_{i,j}$, we have

$$\prod_{p=1}^{R} E(\exp(\alpha x_p)) \le \exp\left(\sum_{p=1}^{R}\frac{1}{2} I_{p,i}\alpha\theta D_i/RT\right) \le \exp\left(\frac{1}{2}C_{i,j}\alpha\theta\right).$$

Substituting the above inequality to Eq. (14), we obtain

$$Pr(x > C_{i,j}\theta) \le \exp\left(\frac{1}{2}C_{i,j}\alpha\theta - \alpha C_{i,j}\theta\right) = \exp\left(\alpha(-\frac{1}{2}C_{i,j}\theta\right). \tag{15}$$

We now define $\alpha := \frac{\ln\frac{\theta}{2}}{C_{i,j}}$, to verify that $\exp(\alpha D_i/RT) \le 1 + \frac{1}{2}\alpha\theta D_i/RT$. Using the fact that for $a \ge 1$, $0 \le b \le 1$, $a^b \le 1 + ab$. Then, we have

$$\begin{aligned} \exp(\alpha D_i/RT) &= \exp\left(\frac{\frac{D_i}{RT}}{C_{i,j}}\ln\frac{\theta}{2}\right) = \left(\frac{\theta}{2}\right)^{\frac{\frac{D_i}{RT}}{C_{i,j}}} \\ &\le 1 + \frac{\frac{D_i}{RT}}{C_{i,j}}\cdot\frac{\theta}{2} \le 1 + \frac{1}{2}\alpha\theta\frac{D_i}{RT}. \end{aligned}$$

Substituting α to (15), we get

$$Pr(x > C_{i,j}\theta) \le \exp\left(-\frac{\theta}{2}\ln\frac{\theta}{2}\right) \le \exp\left(-\frac{\theta}{2}\right).$$

This implies the following inequality

$$\begin{aligned} Pr(\rho > \theta) &\le \sum_{l_{i,j}\in\mathcal{E}} Pr(\rho_{ij} > \theta) = \sum_{l_{i,j}\in\mathcal{E}} Pr(x > C_{i,j}\theta) \\ &\le \sum_{l_{i,j}\in\mathcal{E}} \exp\left(-\frac{\theta}{2}\right) = M\exp\left(-\frac{\theta}{2}\right). \end{aligned}$$

Choosing $\theta = 2\ln 4M$, the theorem can then be proved.□

5.2 Handling Multiple Coflows

Taking advantage of Algorithm 1, we treat the single-coflow solution as a black box, and design a competitive algorithm to minimize the average CCT of multiple coflows. The key idea is that when a new coflow arrives, we first invoke the Algorithm 1 to calculate the reduce task placement and the bandwidth allocation for this new coflow. Then, we rescale the bandwidths of all existing flows including the flows in this new coflow, with the purpose of deriving a feasible solution for each coflow and fully utilizing the link capacity. The algorithm is shown in Algorithm 2, which is competitive with a non-trivial ratio for problem *P1*.

Algorithm 2 works in a *laissez-fair* manner, i.e., it will be invoked whenever a new coflow arrives or an existing coflow finishes (Step 1). To avoid frequent reduce task placement, it invokes Algorithm 1 only for the newly arrived coflows, and stores the computed solution $\{\{I^k_{p,i}\},\{B^k_{i,j}\}\}$ for each coflow k (Step 2). These solutions may be infeasible due to the bandwidth contention of concurrent coflows. Hence, it scales down each coflow's bandwidth with a weight factor λ_k (Step 3-6). Such weighted sharing policy inherently guarantees *fairness* among concurrent coflows: large coflows will get more bandwidth, while small coflows will get relatively less bandwidth. Finally, it scales all flow's bandwidth by a same largest possible factor, to utilize the residual bandwidth (Step 7).

Our algorithm requires very little overhead since it only makes new scheduling decisions on coflow arrival or departure. Moreover, our algorithm has a low time complexity. Its dominant operation lies in scaling all coflows' bandwidths from their relevant single-coflow solutions derived by Algorithm 1. Algorithm 1's dominate overhead is LP calculation and hence its the time complexity is $O(LP(O(R_{max}N), O(N^2)))$, where $R_{max} = \max_k R_k$ and $LP(x,y)$ is the time complexity of solving an LP with x variables and y constraints. In conclusion, Algorithm 2 can run in $O(K)\cdot O(LP(O(R_{max}N), O(N^2)))$. The following theorem further demonstrates the efficiency of Algorithm 2 in solving the original problem *P1*.

Algorithm 2. Minimize Average CCT of Multiple Coflows

Input: Coflow information $\{\{D_i^k\}, R_k\}, \forall k \in \mathcal{K}$
Output: Feasible solutions for all $k \in \mathcal{K}$
1: **while** receiving a new coflow or a feedback indicating the completion of an existing coflow **do**
2: Add this new coflow to J_Ω or remove the completed coflow from J_Ω. Here, J_Ω stores the set of coflows that are not completed till current time.
3: **for** each coflow k in J_Ω **do**
4: Define $\lambda_k := \frac{\sqrt{T_{P3}^k}}{\sum_{k' \in J_\Omega} \sqrt{T_{P3}^{k'}}}$, where T_{P3}^k is the optimal objective of the LP **P3** for coflow k.
5: Update the solution of k as S_k:=$\{\{I_{p,i}^k\}, \{\lambda_k B_{i,j}^k\}\}$, where $\{\{I_{p,i}^k\}, \{B_{i,j}^k\}\}$ is the solution computed by Algorithm 1 when coflow k arrived.
6: **end for**
7: Find a largest factor to scale the bandwidths of all flows in $\mathcal{J}_\Omega$ to pursue work conversing property.
8: **end while**

Theorem 5. *Algorithm 2 is $K\rho$-competitive for the original problem* P1*, where ρ is the competitive ratio of Algorithm 1.*

Proof. Define T_{P1} as the optimal value of problem *P1*, and let T_{P2}^k, T_{P3}^k denote the optimal CCTs of coflow k for problem *P2* and *P3*, respectively. It is clear that each coflow k contributes to the average CCT with no less than its minimum completion time T_{P2}^k when it occupies the network exclusively. Given the result of Theorem 3, we have $T_{P1} \geq \frac{1}{K}\sum_{k=1}^K T_{P2}^k \geq \frac{1}{K}\sum_{k=1}^K T_{P3}^k$. Therefore, we only need to compare the performance of our algorithm to $\frac{1}{K}\sum_{k=1}^K T_{P3}^k$. Specifically, let T_{alg} denote the average CCT achieved by Algorithm 2, and we focus on the proof of $T_{alg} \leq \rho\sum_{k=1}^K T_{P3}^k \leq K\rho \times T_{P1}$ in the rest proof process.

Suppose that there exists an optimization problem for some subset J_Ω of $\{1, \ldots, K\}$

$$\text{Minimize} \sum_{k\in J_\Omega} \frac{T_{P3}^k}{x_k} \text{ Subject to: } \sum_{k\in J_\Omega} x_k \leq 1,$$

where x_k is non-negative value. By leveraging the Cauchy-Schwarz inequality, we have

$$\sum_{k\in J_\Omega} \frac{T_{P3}^k}{x_k} \geq \left(\sum_{k\in J_\Omega} \frac{T_{P3}^k}{x_k}\right) \cdot \left(\sum_{k\in J_\Omega} x_k\right) \geq \left(\sum_{k\in J_\Omega} \sqrt{T_{P3}^k}\right)^2.$$

When $x_k = \sqrt{T_{P3}^k}/\sum_{k\in J_\Omega}\sqrt{T_{P3}^k}, \forall k \in J_\Omega$, the above optimization problem can be optimally solved. This implies that for each k in each iteration of Algorithm 2, the weighted factors λ_k's ($\forall k \in J_\Omega$) are optimally picked. In other words, the λ_k's have least impact on the average CCT when rescaling the bandwidth of each coflow. Define $\lambda_k^{(K)} := \sqrt{T_{P3}^k}/\sum_{k=1}^K \sqrt{T_{P3}^k}$ and let T_{alg}^k denote the CCT of coflow k achieved by the Algorithm 1. Since J_Ω might be a subset of $\{1, \ldots, K\}$, we have the following inequality for any k

$$\lambda_k \geq \frac{\sqrt{T_{P3}^k}}{\sum_{k=1}^K \sqrt{T_{P3}^k}} = \lambda_k^{(K)}.$$

Combining Algorithm 1, the CCT of each coflow is at most

$$\frac{T_{alg}^k}{\lambda_k} \leq \frac{T_{alg}^k}{\lambda_k^{(K)}} \leq \rho\frac{T_{P3}^k}{\lambda_k^{(K)}} = \rho\sqrt{T_{P3}^k}\sum_{k=1}^K \sqrt{T_{P3}^k}.$$

Again applying the Cauchy-Schwarz inequality, we have

$$T_{alg} = \frac{1}{K}\sum_{k=1}^K \frac{T_{alg}^k}{\lambda^k} \leq \frac{1}{K}\sum_{k=1}^K \frac{T_{alg}^k}{\lambda_k^{(K)}}$$
$$\leq \frac{\rho}{K}\left(\sum_{k=1}^K \sqrt{T_{P3}^k}\right)^2 \leq \rho\sum_{k=1}^K T_{P3}^k \leq K\rho T_{P1}.$$

Thus, proved. □

Remarks. Such good theoretical performance lies in the scaling factor λ_k chosen for each coflow. First, λ_k is proportional to the square root of T_{P3}^k, ensuring fairness among coflows: the coflows with large minimum possible completion time will get more bandwidth, while those with small completion time will get relatively less bandwidth. Second, as indicated in the proof process of the above theorem, λ_k is computed with respect to the optimal average CCT of all concurrent coflows. In other words, there exists no coflow that can unilaterally improve its scaling factor without incurring a negative impact on the performance of other coflows.

One may further notice that our algorithm computes endpoint placements for each coflow only once when the coflow arrives at the network. The reasons are two-fold. First, computing endpoint placement decisions frequently can bring substantial overheads to our scheduler. Second, changing the endpoint (reduce task) placement in the runtime is a hard task and remains largely unexplored in existing data-parallel computing frameworks.

Algorithm 3. Guaranteeing Coflow Deadline

Input: Coflow information $\{\{D_i^k\}, R_k, t_k, \Gamma_k\}, \forall k \in \mathcal{K}$
Output: Feasible solutions for all $k \in \mathcal{K}$
1: **while** receiving a new coflow or a feedback indicating the completion of an existing coflow **do**
2: If this is a new coflow, invoke Algorithm 1 to compute $\{\{I_{p,i}^k\}, T_{alg1}^k\}\}$ for it and add this coflow to J_Ω if it can be admitted; Otherwise, remove the completed coflow from J_Ω.
3: Sort coflows in J_Ω with shortest-$(\Gamma_k - t_k)$-first policy.
4: For each coflow k in J_Ω, allocate $(D_i^k \frac{\sum_{p=1}^{R_k} I_{p,j}^k}{R_k})/(\Gamma_k - t_k)$ amount of bandwidth to its flow on link l_{ij}.
5: Find a largest factor to scale the bandwidths of all flows in $\mathcal{J}_\Omega$ to pursue work conversing property.
6: **end while**

6 Extended Models

Our optimization framework can be flexibly extended to incorporate various design choices and practical requirements for applications or datacenter operators. In particular, we mainly consider two extended models: one is done by enforcing a bandwidth usage budget and the other one by considering coflow deadline.

6.1 Enforcing Bandwidth Usage Budget

It has been widely accepted that inter-datacenter WAN bandwidth is an expensive and scarce resource. It not only brings the operational cost for datacenter providers [5], [6], [19], but also incurs WAN usage cost for the third-party tenants that run their services on public clouds such as AWS and Azure [36]. Purely reducing WAN usage can arbitrarily increase the CCT, while solely minimizing the CCT could result in increased WAN bandwidth usage. Actually, most datacenter operators and third-party tenants want to improve their services' or jobs' performance as much as possible, with a given budget [5]. So, we are motivated to enforce a WAN bandwidth usage budget in our *SmartCoflow* framework, so as to enable a flexible balance choice between the speedup of coflows and WAN usage.

As a baseline for the budget, we start with the bandwidth consumption $\{C_{i,j}, \forall l_{ij}\}$ of the original *SmartCoflow* framework that optimizes for average CCT of coflows. We set the WAN bandwidth usage budget to be $\{\sigma \cdot C_{i,j}\}$, $0 < \sigma \leq 1$. $\sigma = 1$ implies no WAN usage budget, while lower values of σ imply tighter budget and slower completion of coflows. To be more specific, by enforcing the budget within the original problem *P1*, we yield an extended optimization problem *P4*:

$$\underset{\{I^k_{p,i}\},\{B^k_{i,j}(x)\}}{\text{Minimize}} \quad \frac{1}{K}\sum_{k=1}^{K} T^k \tag{16}$$

$$\text{S.t.:} \sum_{k=1}^{K} B^k_{i,j}(x) \leq \sigma C_{i,j}, \forall l_{ij} \in \mathcal{E}, \forall x \geq 0. \quad \text{Eqs. } (1), (2), (3), (4). \tag{17}$$

We can see that the only difference between *P4* and *P1* lies in the bandwidth constraint. To solve *P4*, the only thing we need to do is to scale down the bandwidth capacity of all links with an identical ratio σ and then use the unmodified Algorithms 1 and 2.

6.2 Considering Deadlines for Coflows

Driven by mission-critical analytics jobs [37], [38] and the fact that inter-datacenter transfers typically require to be completed within certain time periods [39], [40], [41], inter-datacenter coflows may be generated with deadlines [9], [15], e.g., all individual flows in a coflow should be completed within a common deadline. It is important to guarantee a coflow's completion within the deadline; otherwise, the coflow will become useless, thus hurting the user experience. To guarantee coflow deadline, the *SmartCoflow* algorithms need some changes. Specifically, we first design an admission control algorithm based on Algorithm 1, and then replace Algorithm 2 with a new one to allocate the minimum bandwidth to each coflow to make all flows finish exactly at the deadline.

To ease the presentation, let Γ_k denote the deadline of coflow $k \in \mathcal{K}$. To guarantee deadlines in online fashion, we introduce admission control. Specifically, once a coflow k arrives, we formulate an ILP (*P2*) for it and solve this ILP with Algorithm 1. Then, we get the reduce task placement decisions and the hypothetical CCT T^k_{alg1} for this coflow k. Though Algorithm 1 is not optimal with respect to the ILP *P2*, it is carried out by assuming that each arriving coflow monopolizes the network. Hence, T^k_{alg1} could be viewed as the minimum time for coflow k required to finish its data transfers. As such, we admit a coflow k if, and only if its minimum CCT $T^k_{alg1} \leq \Gamma_k - t_k$ (t_k is the arrival time of k).

We now design a new Algorithm 3 to perform bandwidth allocation. It is different from Algorithm 2 in Steps 2-5. Specifically, Algorithm 3 starts by admitting a new coflow or removing existing completed coflow to get the set of coflows that have arrived but not completed until now. Then, it sorts coflows in ascending order of $\Gamma_k - t_k$ and seeks to allocate the minimum bandwidth to every coflow to guarantee its deadline. Finally, it scales all flow's bandwidth as Algorithm 2 did to pursue work conserving property.

7 Performance Evaluation

In this section, we evaluate *SmartCoflow* using both a small-scale testbed implementation and large-scale simulations.

Comparing Solutions. We compare the following schemes with *SmartCoflow* throughout our experiments.

- *Varys-only*: schedules all coflows with the Shortest-Effective-Bottleneck-First (SEBF) coflow scheduler in Varys [15], with already-fixed endpoints of flows for each coflow. This scheme corresponds to a scheduling-only scheme that ignores the reduce task placement.
- *Iridium+Varys*: assigns the reduce tasks for each coflow with the reduce task placement method in Iridium [5], and then schedules all flows using Varys SEBF scheduler [15]. This scheme considers the reduce task placement and coflow scheduling independently, rather than jointly.

Performance Metrics. We define $\frac{CCT_2 - CCT_1}{CCT_2}$ as the performance improvement of scheme 1 compared to scheme 2, where CCT_1 and CCT_2 are the average CCTs achieved by scheme 1 and scheme 2, respectively.

7.1 Small-Scale Testbed Implementation

We implement our *SmartCoflow* scheduler based on an open-source framework—Varys [15], [20]. The Varys framework can not only provide a simple API to data-parallel jobs for coflow submission, but also provide a global view of the network and coflow information. Upon receiving a new coflow or an update indicating the completion of an existing coflow, *SmartCoflow* invokes Algorithm 2 to calculate the reduce task placement and scheduling decisions, based on the information of new coflow and the updated information of existing coflows. We use the simplex method implemented in the `Breeze` optimization library [35] to solve the relaxed LP problem *P3* involved in *SmartCoflow* algorithms. To enforce the calculated reduce task placement decisions, we have implemented two new `Scala` classes: `CoflowSender` and `CoflowReceiver`. We launch a `CoflowSender` thread at each sender node of a coflow, by

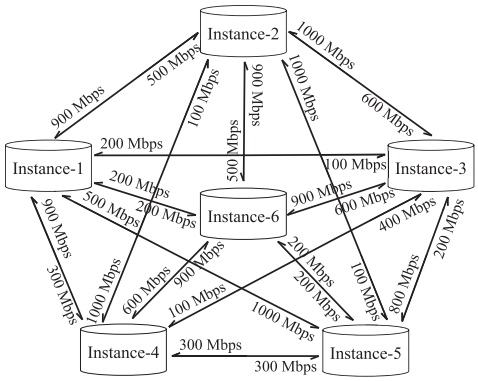


Fig. 6. The emulated testbed on Google's cloud compute engine.

TABLE 2
Intermediate Data Associated With Each Coflow

Instances	Intermediate data associated with each coflow (MB)									
	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Instance-1	900	1000	100	900	100	700	800	300	300	900
Instance-2	100	1000	400	1000	700	300	200	700	600	1000
Instance-3	100	100	1000	700	0	1000	500	700	200	600
Instance-4	600	1000	800	800	300	0	400	100	800	100
Instance-5	100	1000	1000	800	0	400	700	100	200	100
Instance-6	300	500	700	400	100	400	700	500	500	200

specifying the coflow id and the number of flows. Then, for each source-destination pair of a coflow, we launch a `CoflowReceiver` thread at the destination node to fetch the corresponding intermediate data. On the other hand, we enforce the calculated scheduling decisions (i.e., $B_{i,j}^k(t)$) to the application-layer bandwidth allocation through updating the rate limit to the `ThrottledInputStream` (java I/O objective) provided by the Varys framework.

We build a testbed with 6 compute instances in the `us-central1-c` zone on the Google's Cloud Compute Engine to emulate an inter-datacenter network, where each instance is an `n1-standard-2` with 2 vCPUs and 7.5 GB memory. We use each instance to emulate a datacenter. To control the link bandwidth that would exist in the inter-datacenter networks, we leverage Linux Traffic Control to limit the link bandwidth between any two compute instances. Fig. 6 shows the detailed bandwidth constrained on each link which is randomly chosen from 100 to 1,000 Mbps. Even though each compute instance we launched is not as large as a commodity datacenter, we believe that this testbed can faithfully emulate the bandwidth bottlenecks in the inter-datacenter networks.

In our experiment, we inject 10 coflows into the network to evaluate the performance of *SmartCoflow*. For each coflow, the amount of associated intermediate data stored on each datacenter is randomly chosen from 100 to 1,000 MB, as shown in Table 2. The number of reduce tasks associated with each coflow is listed in Table 3. Based on the

TABLE 3
The Number of Reduce Tasks Associated With Each Coflow

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Num. of reduce tasks	11	4	10	3	12	5	3	4	8	6

TABLE 4
Reduce Task Placement for Three Different Schemes

Coflow	Varys-only						Iridium+Varys						*SmartCoflow*					
	Instance Indexes						Instance Indexes						Instance Indexes					
	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
C1	4	0	4	3	0	0	2	2	0	0	5	2	0	2	0	1	8	0
C2	1	1	0	1	1	0	2	0	0	1	0	1	1	0	0	3	0	0
C3	0	0	3	2	3	2	2	1	2	0	3	2	1	0	5	0	0	4
C4	1	1	0	1	0	0	1	0	0	1	0	1	0	0	1	1	0	1
C5	1	7	0	3	0	1	4	0	3	4	0	1	4	0	3	4	0	1
C6	2	0	2	0	1	0	0	3	0	0	1	1	0	3	0	0	2	0
C7	1	0	0	0	1	1	1	0	0	0	1	1	1	1	0	0	1	0
C8	0	2	2	0	0	0	0	2	0	0	0	2	0	3	1	0	0	0
C9	1	2	0	3	0	2	2	0	1	2	0	3	2	0	0	3	0	3
C10	2	3	1	0	0	0	2	3	0	0	0	1	3	1	0	0	0	2

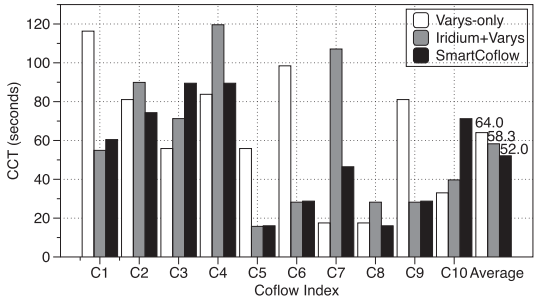
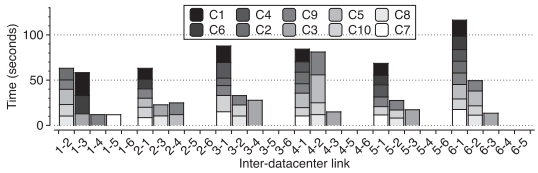


Fig. 7. The CCT of each coflow achieved by Varys-only, Iridium+Varys and *SmartCoflow* schemes, respectively.

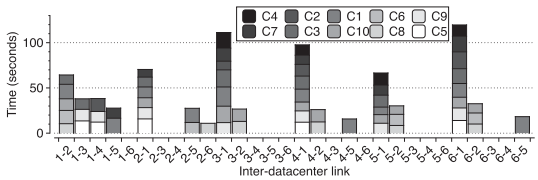
above-mentioned experimental setup, we calculate the decisions on reduce task placements. Note that for the Varys-only scheme, we use the locality policy to place the reduce tasks of each coflow, which means that the number of reduce tasks placed on each datacenter is proportional to the intermediate data on it. In fact, such locality policy is commonly adopted in data-parallel frameworks such as Spark [23]. The calculated reduce task placement strategies under different schemes are illustrated in Table 4.

Fig. 7 first presents the CCTs achieved by Varys-only, Iridium+Varys and *SmartCoflow* schemes, respectively. It is clear that *SmartCoflow* can save $\frac{64.0-52.0}{64.0} = 18.75\%$ of the average CCT compared to the Varys-only scheme, and it also can reduce the average CCT by $\frac{58.3-52.0}{58.3} = 10.81\%$ compared to the Iridium+Varys scheme. We further observe that *SmartCoflow* can reduce the tail CCT, compared to both Varys-only and Iridium+Varys schemes. The reason for these results is that *SmartCoflow* can smartly avoid the bottleneck links when transferring the flows, reducing the overall CCTs and improving the link bandwidth utilization as well.

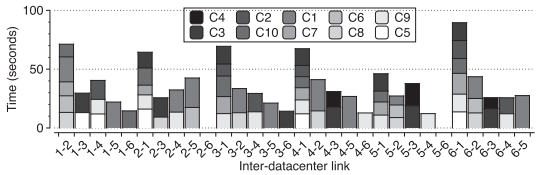
To clearly illustrate the underlying reason for such improvement, we next plot the schedule orders achieved by three different schemes in Fig. 8. The scheduling order under the Varys-only scheme is C7 → C8 → C10 → C5 → C3 → C9 → C2 → C4 → C6 → C1. While for Iridium+Varys scheme, the scheduling order is C5 → C9 → C8 → C6 → C10 → C1 → C3 → C2 → C7 → C4. The reason for such different scheduling order is that different schemes use different reduce task placement strategies, making the bottleneck flows (i.e., the largest flow in a coflow [15]) of different



(a) Varys-only scheme



(b) Iridium+Varys Scheme



(c) *SmartCoflow*

Fig. 8. The duration in which each coflow is scheduled on each link, under the (a) Varys-only, (b) Iridium+Varys, and (c) *SmartCoflow* schemes, respectively.

coflows to appear at different links. In contrast, *SmartCoflow* can obtain a better reduce task placement strategy, and schedule the 10 coflows with an order of C5 → C9 → C8 → C6 → C7 → C1 → C10 → C2 → C3 → C4, resulting in significant reduction in the average CCT (as shown in Fig. 7). An interesting observation is that almost all links (except the links 2-6 and 5-6) have been utilized for coflow transmission under *SmartCoflow*, indicating that *SmartCoflow* is capable of improving the link bandwidth utilization. To show this point more clearly, we plot the average link bandwidth utilization over time for different methods in Fig. 9. As we can see, *SmartCoflow* can improve link bandwidth utilization substantially in the beginning 40s, and maintain comparable bandwidth utilization after 40s, compared to Varys-only and Iridium+Varys solutions.

To demonstrate the efficiency of *SmartCoflow*, we further evaluate its scalability. In particular, we record the time it takes to solve the relevant LP problem *P3*. Fig. 10 plots the computation time *SmartCoflow* taken to resolve the relevant

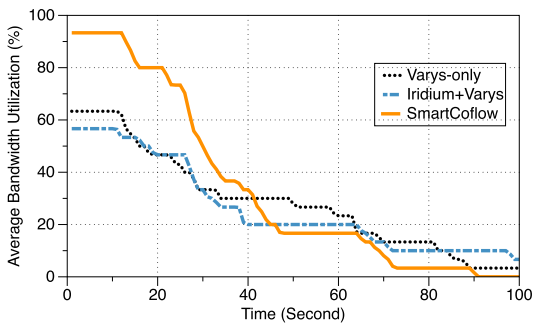


Fig. 9. Average link bandwidth utilization achieved by Varys-only, Iridium +Varys and *SmartCoflow* schemes, respectively.

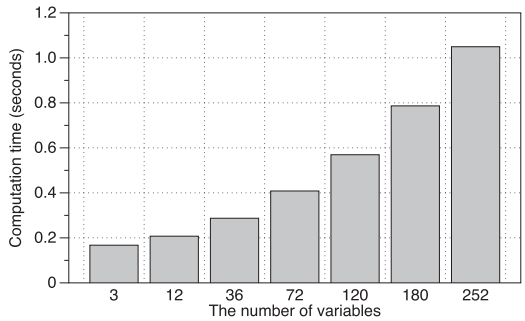


Fig. 10. The computation time of *SmartCoflow*'s linear program.

LP under different number of variables. In this figure, each point is averaged over ten runs. Though the computation time grows as the number of variables increases, we can observe that *SmartCoflow* is still efficient: it takes about 0.4 seconds to return the LP result for 72 variables and 1 second for 252 variables. This computation time is relatively small, as the CCT of a coflow can reach hundreds of seconds across the geo-distributed datacenters. This implies that our *SmartCoflow* is scalable as its dominant overhead lies in the LP, which can be solved with standard LP solvers. One can further reduce the overhead of solving the LP by using commercial solvers (e.g., MOSEK [42] and CPLEX [43]) that take less than one second to return results for problems with thousands of variables.

7.2 Large-Scale Trace-Driven Simulation

We develop a flow-level simulator based on an open-source framework `CoflowSim` [44], to further exploit the advantages of our proposed solution when applying to large-scale network with a large number of concurrent coflows. To reduce the simulation complexity, `CoflowSim` accounts for both the flow arrival and departure events, rather than packet sending and receiving events. Also, it updates the rate and remaining volume of each flow when an event initiates. To solve the LP problem *P3* in *SmartCoflow*, we embed the API provided by `Breeze` into our simulator.

Simulation Setup and Data Trace. We simulate a production inter-datacenter network with 40 datacenters, which is a typical size in today's inter-datacenter networks [45]. Each datacenter has a uniform capacity of 100 computing slots. In our 40-datacenter setup, we vary the bandwidth between 100 Mbps to 1 Gbps, hoping to mimic the heterogeneous bandwidths between different datacenters. Our simulations are conducted on Hive/MapReduce trace provided by Facebook, which is a widely adopted trace in coflow issues [15], [16], [46]. The original trace is from a 3000-machine 150-rack cluster with 10:1 over-subscription ratio, and contains 526 coflows. We scale down all coflows to the 40-datacenter inter-datacenter network in our deployment, with preserving the original coflow's communication characteristics. Note that the original trace only provides the whole data size of a coflow to be transferred to reducers. Therefore, we distribute the data to each flow in a uniform manner, and accordingly obtain the intermediate data placed on each datacenter.

Simulation Results. Table 5 first presents the average, 95th percentile and maximum CCTs achieved by three different

TABLE 5
The Average CCT, 95th Percentile CCT, and Maximum CCT Achieved by Three Different Schemes

Metrics	Varys-only	Iridium+Varys	*SmartCoflow*
Average CCT (ms)	111684	109936	79742
95th percentile CCT (ms)	47760	46158	29258
Maximum CCT (ms)	8784344	8827218	6733016

schemes. We observe that *SmartCoflow* reduces the average, 95th percentile and maximum CCTs by $\frac{111684-79742}{111684}=28.6\%$, $\frac{47760-29258}{47760}=38.7\%$, and $\frac{8784344-6733016}{8784344}=23.4\%$, respectively, compared to the Varys-only scheme. Moreover, compared to the Iridium+Varys scheme, the average, 95th percentile and maximum CCTs can also be reduced by $\frac{109936-79742}{109936}=27.5\%$, $\frac{46158-29258}{46158}=36.7\%$, and $\frac{8827128-6733016}{8827128}=23.7\%$, respectively, under our *SmartCoflow* scheme. One may wonder at this point that the 95th percentile CCT is even smaller than the average CCT for each scheme. This is because that some coflows experience extremely high CCT, while the CCTs of other coflows are relatively low. The above results directly confirm that combining reduce task placement and scheduling can significantly reduce the average CCT of coflows.

To understand on a microscopic level, we also plot the CDF of CCT for all coflows in Fig. 11. We can clearly find that 33.65 percent of coflows experience a CCT smaller than 8 ms under *SmartCoflow* scheduler, while the fractions for Varys-only and Iridium+Varys are only 18.82 and 20.34 percent. We further observe that Varys-only and Iridium+Varys schemes perform a little better than *SmartCoflow* — only for coflows whose CCT are in the range [56,1008] ms. As coflows become larger (> 1008 ms), *SmartCoflow* always performs better, as the curve of *SmartCoflow* is higher than that of Varys-only and Iridium+Varys. Note that the curve of *SmartCoflow* maintains a stable value within [8,1008]ms because no coflows experience a CCT within [8,1008]ms.

After we see the improvements of CCTs, we also evaluate the performance of diferent schemes in terms of the amount of shuffle data across geo-distributed datacenters for each coflow. Even though reducing the coflow shuffle data is not the main goal of our optimization, we find out that it is in the line with the goal of reducing CCT. Fig. 12 shows the CDF of coflow shuffle data under three schemes. We can easily check that *SmartCoflow* always achieve a smaller size of coflow shuffle data, compared to both the Varys-only and Iridium+Varys schemes. Specifically, under our *SmartCoflow* scheme, 40.68 percent of coflows become short (< 5 MB). On the other hand, the fractions of short coflows under the Varys-only and Iridium+Varys schemes are both 28.71 percent.

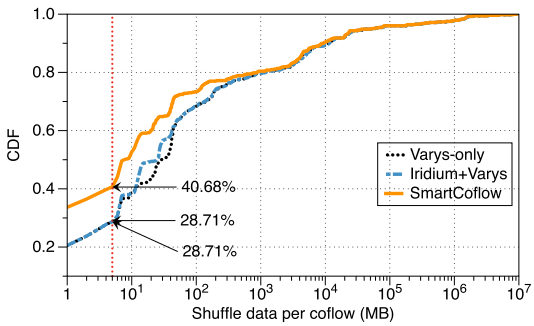


Fig. 12. The CDFs of coflow shuffle data under Varys-only, Iridium+Varys and *SmartCoflow* respectively. The X-axes are in logarithmic scale.

The Effectiveness of WAN Usage Budget Enforcement. Reducing inter-datacenter WAN bandwidth usage is meaningful to both providers and thirty-party cloud tenants. To evaluate the impact of different bandwidth usage budgets, we vary the knob σ from 0.1 to 1. Then, we record the average CCT of coflows achieved by our *SmartCoflow* solution under various knobs, as shown in Fig. 13. For comparison, we fix the knob σ to 1 (implying no budget) and plot the average CCT achieved by Varys-only and Iridium+Varys solutions under this knob. From Fig. 13, the first observation is that the average CCT under *SmartCoflow* increases as σ decreases. This is reasonable because the smaller is the σ, the less bandwidth can be used for coflow scheduling. The second obvious observation is that with the same knob $\sigma = 1$, *SmartCoflow* achieves lower average CCT than Varys and Iridium+Varys. The last implicated observation is that even when *SmartCoflow* uses 20 percent less bandwidth, it still achieves a little bit smaller average CCT than Varys and Iridium+Varys. This verifies that *SmartCoflow* can smartly use the bandwidth to ensure fast coflow transmission and reasonable bandwidth usage.

The Effectiveness of Considering Coflow Deadline. The trace we used has no coflow-specific deadlines. We introduce them as existing literature [15] did, by using the minimum CCT of a coflow in an empty network. Specifically, we let the deadline of a coflow to be the multiplier of its minimum CCT and $(1 + U(0, \eta))$, where $U(0, \eta)$ is a uniform random number ranging from 0 to η. By varying η from 0.1 to 25, we

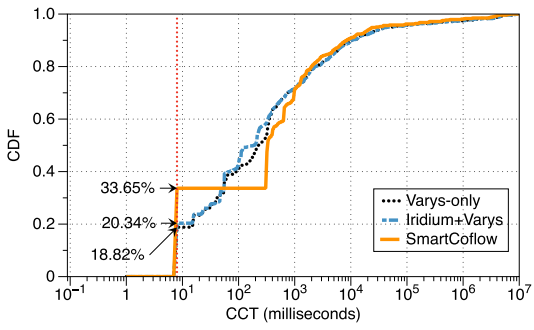


Fig. 11. The CDFs of CCTs under Varys-only, Iridium+Varys and *SmartCoflow*, respectively. Note that the X-axes are in logarithmic scale.

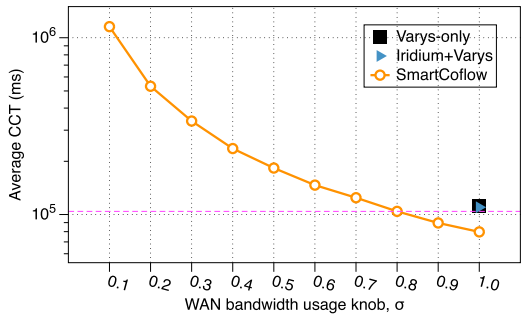


Fig. 13. The average CCT under different WAN bandwidth usage knobs. Note that the pink line is an auxiliary line for illustrating the experimental results more clearly.

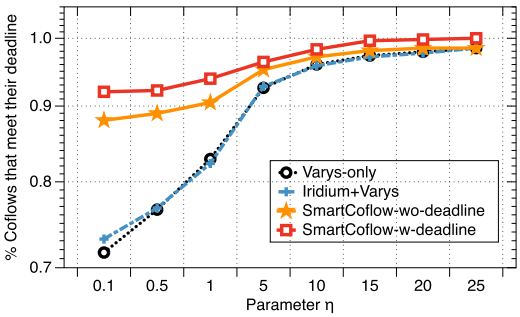


Fig. 14. The ratio of coflows that meet their deadline when using different schemes under varying ϵ.

plot the deadline meet rate achieved by different schemes in Fig. 14. It is obvious that under all the schemes, the deadline meet rate increases as η increases. Among all the schemes, *SmartCoflow-w-deadline* performs best in meeting coflow deadline. The reasons are two-fold. On the one hand, it can reduce the coflow shuffle data significantly by smartly placing the endpoints of coflows. On the other hand, it uses Algorithm 1 as an efficient admission control rule, and combines the shortest-deadline-first prioritization and deadline-associated minimum bandwidth allocation policy. More precisely, our *SmartCoflow-w-deadline* can improve the deadline meet rate by up to 20.3 and 18.8 percent, compared to Varys and Iridium+Varys, respectively. One may wonder that compared to both Varys and Iridium+Varys, the *SmartCoflow* algorithms achieve a decreasing benefit in meeting the coflow deadline as η increases. The reason is that an increasing η leads to a longer deadline for each coflow. In such a case, Varys and Iridium+Varys can meet coflows' deadlines, leaving less room to *SmartCoflow* algorithms to improve coflow deadline meet rate.

8 Discussion

Estimating Intermediate Data Size. Our work assumes prior information of intermediate data sizes as we believe that to be reasonable. However, to obtain such information, considerable efforts on modifying applications are required. Such modification is doable, but not trivial. Without modifying applications, one possible approach to estimate the intermediate data size is to apply the machine learning technique to learn flow sizes from past system traces [47]. For example, one can trace information like the data received, read from disk or memory, CPU usage, and then build a machine learning model to interpret the traced dataset, and finally deploy the learned model in either user or kernel space to estimate flow size. The rationale for such learning is that the traced activities may have causality, e.g., most datacenter jobs are repetitive [38].

Handling Coflow Dependencies. So far, we only consider isolated single-stage coflows. However, GDA jobs with pipeline dataflows [16], [48] and GDML jobs with numerous iterations can create multiple coflows with dependencies, where a coflow cannot start until its dependent one has finished. A simple approach for considering coflow dependencies is to order coflows first by ancestry and then break ties using our Algorithm 2. Specifically, one can keep collecting the set of independent coflows and feeding it to Algorithm 2 for scheduling until all coflows have finished. Instead of ordering by ancestry, one can also use the Critical-Path method [49] to acquire a better order, so as to achieve better performance. We leave it as one direction of future work.

Handling Tiny Coflows. While coflows in production environments are mostly large [15], there do exist some tiny coflows. Due to the inherent scheduling overhead in our scheduler, our work fails to handle the tiny coflows. One possible way is to give tiny coflows the highest priorities if the geo-distributed WAN routers support priority queuing. Alternatively, one can directly transfer tiny coflows during the period of calculating scheduling decisions, and the newly calculated schedules can not be enforced until the tiny coflows have finished. We leave it as future work.

Coflow Routing. Our scheduler performs well even when routing is not considered. Existing studies have demonstrated that integrating coflow routing and scheduling can achieve better performance [12], [18]. A possible way to realize such integration could do the following. We add a new set of decision variables (i.e., $\omega_{i,j}^{k}$) along with relevant link constraints in the formulation to indicate the path of the $i \rightarrow j$ flow in coflow k, and redesign new algorithms to derive all the decisions including task placement, routing, scheduling.

Regulatory and Privacy Constraints. So far, our work considers that traffic or data can be freely moved between datacenters. However, regulatory and privacy concerns might forbid the data movement between certain pairs of datacenters [50]. One possible approach is to translate such regulatory and privacy concerns into the constraints on the placement of reduce tasks. Then, one can design efficient heuristic to avoid placing reduce tasks on invalid datacenters.

Incorporating With Data-Parallel Frameworks. Our scheduler was implemented based on Varys [15], which provides an API to abstract away the underlying scheduling and communication mechanisms. So, once the data-parallel frameworks (e.g., Spark and MapReduce) create `VarysClient` objects, our scheduler can be incorporated with those frameworks, and user jobs can take advantages of our scheduler for their coflow transmissions without any modifications.

9 Related Work

SmartCoflow contains two parts: reduce task placement and coflow scheduling. There is a large spectrum of related work in datacenter networks, along either reduce task placement or scheduling. We only review some closely related ones here.

Reduce Task Placement in Datacenter Networks. Existing work mainly focuses on placing reduce tasks close to their *intermediate data* for optimizing the completion times of jobs inside a data center or across geo-distributed datacenters. ShuffleWatcher, improves the locality of shuffle by placing both map and reduce tasks on the same set of racks in a single datacenter [51]. Iridium [5] and Flutter [10] reduce the job completion time by placing reduce tasks on or close to datacenters that have a large amount of intermediate data and relatively high link bandwidth. Chen *et al.* propose to achieve max-min fairness among multiple jobs across geo-distributed data centers, by using the task scheduling method [32]. Regarding the endpoints of network transfers, Sinbad leverages the flexibility in

the placement of output data of big data jobs, without considering other stages (e.g., shuffle stage) in a job [52]. Corral focuses on joint optimization of input data and task placement, so as to reduce the network contention [53]. However, the benefits of these techniques are inherently limited because they do not take into account the network flow scheduling after the reduce tasks or endpoints have been fixed. CLARINET [8] is the most related recent work that considers network flow scheduling after task placement. Nevertheless, it focuses on optimizing network flows at the flow-level instead of the coflow-level, and it considers task placement and flow scheduling independently rather than jointly.

Network Scheduling in Datacenter Networks. Existing work can be categorized into two categories: flow-level scheduling and coflow-level scheduling. Regarding the flow-level scheduling, there are many existing solutions that aim to finish individual network flows faster or improve network utilization, such as pFabric [54], PDQ [55], PASE [56], Aemon [57]. There are also solutions on providing performance guarantee for individual flows or even the upper-layer jobs [58], [59], [60], [61], [62]. Unfortunately, since coflows generalize traditional point-to-point flows by capturing the multipoint-to-multipoint aspect of data-parallel communications, flow-level resource schedulers do not consider the collective behaviors of flows in a coflow and thus are coflow-agnostic. Regarding the coflow-level scheduling, Orchestra takes the first step to consider the collective behaviors of flows when optimizing flow transfers in data center networks [11]. After that, Chowdhury *et al.* explicitly present the concept of coflow to describe such collective behaviors of flows [9]. Then, many solutions on coflow-level scheduling have been proposed with the aim of reducing CCT (e.g., Baraat [63], Varys [15], Aalo [16], CODA [17], RAPIER [12], OMCoflow [18], Siphon [64], Sincronia [26]), or achieving fairness (e.g., Li. *et al.* [65], Coflex [31], Utopia [66], and NC-DRF [67]). However, since these coflow-level schedulers consider the endpoints of coflow transfers to be fixed, there is little space for scheduling to take effect for optimizing the average CCT. *SmartCoflow*, on the other hand, uses reduce task placement technique to place the endpoints of coflows smartly before scheduling.

10 Conclusion

In this paper, we have presented *SmartCoflow*, a coflow-aware optimization framework that seeks to integrate endpoint placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters. Starting from an approximate algorithm for minimizing the CCT of single coflow, *SmartCoflow* develops a fast and efficient online algorithm to minimize the average CCT of multiple coflows. We have conducted rigorous theoretical analysis to show that *SmartCoflow* can achieve a good competitive ratio in minimizing the average CCT of the coflows, without prior knowledge of future coflows. Furthermore, we have also extended *SmartCoflow* to make our solution be able to support various design choices and practical requirements, such as enforcing an inter-datacenter bandwidth usage budget and guaranteeing coflow deadline. To the best of our knowledge, *SmartCoflow* is the first work that proposes and proves the position that endpoint placement and coflow scheduling must be jointly considered for various optimization goals, i.e., average CCT, meeting deadlines and saving bandwidth usage. Extensive testbed experiments and trace-driven simulations have shown convincing evidence that *SmartCoflow* can reduce the average CCT, lower inter-datacenter bandwidth usage, and improve coflow deadline meet rate when compared to the prevailing methods.

Acknowledgments

This work was supported in part by the NSFC General Technology Basic Research Joint Funds under Grant U1836214; in part by the State Key Program of National Natural Science of China under Grant 61832013; in part by the Artificial Intelligence Science and Technology Major Project of Tianjin under Grant 18ZXZNGX00190; in part by the National Key R&D Program of China under Grant 2019QY1302; in part by the NSFC under Grant 61672379; in part by the National Key R&D Program of China under Grant 2019YFB2102404; in part by the NSFC-Guangdong Joint Funds under Grant U1701263; in part by the Natural Science Foundation of Tianjin under Grant 18ZXZNGX00040; in part by the National Key R&D Program of China under Grant 2018YFB1004700; in part by the NSFC under Grants 61872265 and 61672131; in part by the Key research and Development Program for Guangdong Province 2019B010136001; in part by the NSFC under Grant 61772112; and in part by the Science Innovation Foundation of Dalian under Grant 2019J12GX037. A preliminary version of this article was presented in IEEE INFOCOM 2018 [1].

References

[1] 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.
[2] Google datacenter locations, Accessed: 2020. [Online]. Available: https://www.google.com/about/datacenters/inside/locations/
[3] Amazon datacenter locations, Accessed: 2020. [Online]. Available: https://aws.amazon.com/cn/about-aws/global-infrastructure/
[4] Microsoft datacenters, Accessed: 2020. [Online]. Available: http://www.microsoft.com/en-us/server-cloud/cloud-os/global-datacenters.aspx
[5] Q. Pu *et al.*, "Low latency geo-distributed data analytics," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 421–434.
[6] K. Hsieh *et al.*, "Gaia: Geo-distributed machine learning approaching LAN speeds," in *Proc. 14th USENIX Conf. Netw. Syst. Des. Implementation*, 2017, pp. 629–647.
[7] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proc. 6th ACM Symp. Cloud Comput.*, 2015, pp. 111–124.
[8] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "CLARINET: WAN-aware optimization for analytics queries," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 435–450.
[9] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 31–36.
[10] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.
[11] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 98–109.
[12] Y. Zhao *et al.*, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE INFOCOM*, 2015, pp. 424–432.
[13] L. Luo, J. Nelson, L. Ceze, A. Phanishayee, and A. Krishnamurthy, "Parameter hub: A rack-scale parameter server for distributed deep neural network training," in *Proc. ACM Symp. Cloud Comput.*, 2018, pp. 41–54.

[14] G. Wangt, A. Phanishayee, S. Venkataraman, and I. Stoicat, "Blink: A fast NVLink-based collective communication library," in *Proc. Conf. Syst. Mach. Learn.*, 2018, Accessed: 2020. [Online]. Available: https://rise.cs.berkeley.edu/wp-content/uploads/2018/01/blink-2-page-11_50.pdf

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

[16] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 393–406.

[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 Conf.*, 2016, pp. 160–173.

[18] Y. Li *et al.*, "Efficient online coflow routing and scheduling," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2016, pp. 161–170.

[19] A. Vulimiri, C. Curino, P. B. Godfrey, T. Jungblut, J. Padhye, and G. Varghese, "Global analytics in the face of bandwidth and regulatory constraints," in *Proc. 12th USENIX Conf. Netw. Syst. Des. Implementation*, 2015, pp. 323–336.

[20] Varys: Efficient clairvoyant coflow scheduler, Accessed: 2020. [Online]. Available: https://github.com/coflow/varys

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

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. 2nd USENIX Conf. Hot Topics Cloud Comput.*, 2010, Art. no. 10.

[23] M. Zaharia *et al.*, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, Art. no. 2.

[24] I. Cano, M. Weimer, D. Mahajan, C. Curino, G. M. Fumarola, and A. Krishnamurthy, "Towards geo-distributed machine learning," *IEEE Data Eng. Bulletin*, vol. 40, no. 4, pp. 41–59, 2017.

[25] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, "The power of choice in data-aware cluster scheduling," in *Proc. 11th USENIX Conf. Operating Syst. Des. Implementation*, 2014, pp. 301–316.

[26] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 16–29.

[27] Q. Ho *et al.*, "More effective distributed ML via a stale synchronous parallel parameter server," in *Proc. 26th Int. Conf. Neural Inf. Process. Syst.*, 2013, pp. 1223–1231.

[28] B. Recht, C. Re, S. Wright, and F. Niu, "HOGWILD: A lock-free approach to parallelizing stochastic gradient descent," in *Proc. 24th Int. Conf. Neural Inf. Process. Syst.*, 2011, pp. 693–701.

[29] M. Jeon, S. Venkataraman, J. Qian, A. Phanishayee, W. Xiao, and F. Yang, "Multi-tenant GPU clusters for deep learning workloads: Analysis and implications," Microsoft Research, Redmond, WA, USA, Tech. Rep. MSR-TR-2018–13, 2018.

[30] Y. Yu, P. K. Gunda, and M. Isard, "Distributed aggregation for data-parallel computing: Interfaces and implementations," in *Proc. ACM SIGOPS 22nd Symp. Operating Syst. Princ.*, 2009, pp. 247–260.

[31] 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.

[32] L. Chen, S. Liu, B. Li, and B. Li, "Scheduling jobs across geo-distributed datacenters with max-min fairness," in *Proc. IEEE INFOCOM*, 2017, pp. 1–9.

[33] J. K. Lenstra, A. R. Kan, and P. Brucker, "Complexity of machine scheduling problems," *Stud. Integer Program.*, vol. 1, pp. 343–362, 1977.

[34] J. K. Karlof, *Integer Programming: Theory and Practice*. Boca Raton, FL, USA: CRC Press, 2005.

[35] Breeze: A numerical processing library for scala, Accessed: 2020. [Online]. Available: http://www.scalanlp.org

[36] EC2 pricing, Accessed: 2020. [Online]. Available: https://aws.amazon.com/ec2/pricing/

[37] Z. Hu, B. Li, C. Chen, and X. Ke, "FlowTime: Dynamic scheduling of deadline-aware workflows and ad-hoc jobs," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 929–938.

[38] S. A. Jyothi *et al.*, "Morpheus: Towards automated SLOs for enterprise clusters," in *Proc. 12th USENIX Conf. Operating Syst. Des. Implementation*, 2016, pp. 117–134.

[39] V. Jalaparti, I. Bliznets, S. Kandula, B. Lucier, and I. Menache, "Dynamic pricing and traffic engineering for timely inter-datacenter transfers," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 73–86.

[40] H. Zhang *et al.*, "Guaranteeing deadlines for inter-data center transfers," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 579–595, Feb. 2017.

[41] S. Kandula, I. Menache, R. Schwartz, and S. R. Babbula, "Calendaring for wide area networks," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 515–526.

[42] 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*. Berlin, Germany: Springer, 2000, pp. 197–232.

[43] IBM ILOG CPLEX optimizer, Accessed: 2020. [Online]. Available: https://goo.gl/jyvDuV

[44] Flow-level simulator for coflow scheduling used in Varys and Aalo, Accessed: 2020. [Online]. Available: https://github.com/coflow/coflowsim

[45] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 15–26.

[46] Synthesized data from real-world traces of data-intensive applications for coflow benchmarking, Accessed: 2020. [Online]. Available: https://github.com/coflow/coflow-benchmark

[47] V. T1ukić, S. A. Jyothi, B. Karlaš, M. Owaida, C. Zhang, and A. Singla, "Is advance knowledge of flow sizes a plausible assumption?" in *Proc. 16th USENIX Conf. Netw. Syst. Des. Implementation*, 2019, pp. 565–580.

[48] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *Proc. IEEE INFOCOM*, 2018, pp. 864–872.

[49] J. E. Kelley Jr, "Critical-path planning and scheduling: Mathematical basis," *Operations Res.*, vol. 9, no. 3, pp. 296–320, 1961.

[50] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. Culler, "GUPT: Privacy preserving data analysis made easy," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 349–360.

[51] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "ShuffleWatcher: Shuffle-aware scheduling in multi-tenant MapReduce clusters," in *Proc. USENIX Conf. USENIX Annu. Tech. Conf.*, 2014, pp. 1–12.

[52] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 231–242.

[53] V. Jalaparti, P. Bodik, I. Menache, S. Rao, K. Makarychev, and M. Caesar, "Network-aware scheduling for data-parallel jobs: Plan when you can," in *Proc. ACM Conf. Special Interest Group Data Commun.*, 2015, pp. 407–420.

[54] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," in *Proc. ACM SIGCOMM Conf.*, 2013, pp. 435–446.

[55] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM Conf. Appl. Technol. Archit. Protocols Comput. Commun.*, 2012, pp. 127–138.

[56] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," in *Proc. ACM Conf. SIGCOMM*, 2014, pp. 491–502.

[57] T. Wang, H. Xu, and F. Liu, "Aemon: Information-agnostic mix-flow scheduling in data center networks," in *Proc. ACM 1st Asia-Pacific Workshop Netw.*, 2017, pp. 106–112.

[58] 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.

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

[60] 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, Oct. 2017.

[61] 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 36th Int. Conf. Distrib. Comput. Syst.*, 2016, pp. 415–424.

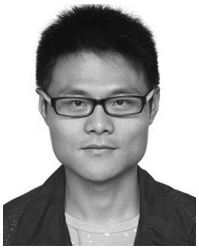
[62] F. Liu, J. Guo, X. Huang, and J. C. Lui, "eBA: Efficient bandwidth guarantee under traffic variability in datacenters," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 506–519, Feb. 2017.

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

[64] S. Liu, L. Chen, and B. Li, "Siphon: Expediting inter-datacenter coflows in wide-area data analytics," in *Proc. USENIX Conf. Usenix Annu. Tech. Conf.*, 2018, pp. 507–518.

[65] 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.

[66] L. Wang, W. Wang, and B. Li, "Utopia: Near-optimal coflow scheduling with isolation guarantee," in *Proc. IEEE INFOCOM*, 2018, pp. 891–899.
[67] L. Wang and W. Wang, "Fair coflow scheduling without prior knowledge," in *Proc. IEEE 38th Int. Conf. Distrib. Comput. Syst.*, 2018, pp. 22–32.



Wenxin Li received the BE and PhD degrees from the School of Computer Science and Technology, Dalian University of Technology, Dalian, China, in 2012 and 2018, respectively. Currently, he is a postdoc researcher with the Hong Kong University of Science and Technology. From 2014–2015, 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.

Xu Yuan (Member, IEEE) received the BS degree from Nankai University, Tianjin, China, in 2009, and the PhD degree from Virginia Tech, Blacksburg, Virginia, in 2016. From 2016 to 2017, he was a postdoctoral fellow with the University of Toronto, Canada. He is currently an assistant professor with the University of Louisiana at Lafayette, Louisiana. His research interest focuses on cloud computing security, algorithm design and optimization for spectrum sharing, coexistence, and cognitive radio networks.

Keqiu Li (Senior Member, IEEE) received the bachelor's and master's degrees from the Department of Applied Mathematics, Dalian University of Technology, Dalian, China, in 1994 and 1997, respectively, and the PhD degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Japan, in 2005. He also has two-year postdoctoral experience with the University of Tokyo, Japan. He was a professor with the School of Computer Science and Technology, Dalian University of Technology, from 2007 to 2016. After that, he joined Tianjin University, where he is currently the director of the College of Intelligence and Computing. He has published more than 100 technical papers, such as the *IEEE Transactions on Parallel and Distributed Systems*, *ACM Transactions on Internet Technology*, and *ACM Transactions on Multimedia Computing, Communications, and Applications*. He was an associate editor of the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. His research interests include internet technology, data center networks, cloud computing, and wireless networks.

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

Xiaobo Zhou recieved the BSc degree in electronic information science and technology from the University of Science and Technology of China (USTC), Hefei, China, in 2007, the ME degree in computer application technology from the Graduate University of Chinese Academy of Science (GUCAS), Beijing, China, in 2010, and the PhD degree from the School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), Ishikawa, Japan, in 2013. He is currently an associate professor with the School of Computer Science and Technology, Tianjin University. Prior to that, he was a researcher with Centre for Wireless Communications, University of Oulu, Finland from 2014 to 2015. His research interests include joint source-channel coding, cooperative wireless communications, network information theory, cloud computing, and software defined networking.

Renhai Xu received the BE and MS degrees from the School of Computer Science and Technology, Dalian University of Technology, Dalian, China, in 2014 and 2017, respectively. Currently, he is working toward the PhD degree in the School of Computer Science and Technology, Tianjin University, Tianjin, China. His research interests include datacenter networks and cloud computing.