Leveraging Endpoint Flexibility When Scheduling Coflows across Geo-distributed Datacenters

Wenxin Li*, Xu Yuan†, Keqiu Li*, Heng Qi*, Xiaobo Zhou‡
*Dalian University of Technology, P.R. China.
†University of Louisiana at Lafayette, USA.
‡Tianjin University, P.R. China.

***Abstract*—Coflow scheduling is crucial to improve the communication performance of data-parallel jobs, especially when these jobs running in the inter-datacenter networks with limited and heterogeneous link bandwidth. However, prior solutions on coflow scheduling assume the endpoints of flows in a coflow to be fixed, making them insufficient to optimize the coflow completion time (CCT). 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 linear programming (MILP) problem to characterize the intertwined relationship between endpoint placement and coflow scheduling, and reveal their impact on the average CCT. Then, we present *SmartCoflow*, a coflow-aware optimization framework, to solve the MILP problem without any prior knowledge of coflow arrivals. 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. To validate the efficiency and practical feasibility of *SmartCoflow*, we implement it as a real-world coflow scheduler based on the Varys open-source framework. Through experimental results from both a small-scale testbed implementation and large-scale simulations, we demonstrate that *SmartCoflow* can achieve significant improvement on the average CCT, when compared to the state-of-the-art scheduling-only method.**

I. Introduction

It is increasingly common for data-parallel jobs to run across geographically distributed datacenters to efficiently process large volumes of data generated all over the world [1–3]. These jobs typically involve a set of network flows to transfer the intermediate data between successive computation stages (e.g., map and reduce)—known as *coflows* [4]. 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 [1, 5, 6]. Meanwhile, the data volume of flows in a coflow could be enormous for a geo-distributed data-parallel job [1], yet a coflow's completion time (CCT) can account for more than 50% of job completion time [7, 8]. Therefore, optimizing the CCTs becomes extremely important to improve the performance of data-parallel jobs running across geo-distributed datacenters.

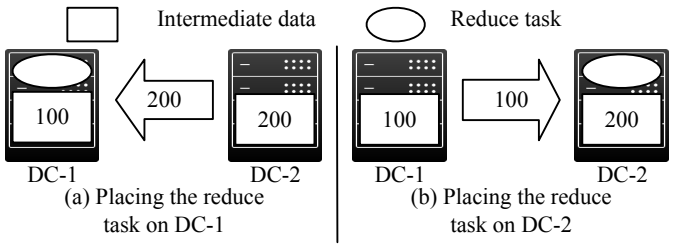


Fig. 1. An illustrating example, where a coflow has one network flow that transfers 200 units of data from DC-2 to DC-1 when placing the reduce task on DC-1. On the other hand, if placing the reduce task on DC-2, this coflow will only need to transfer a 100-unit flow from DC-1 to DC-2.

Many existing works [7–12] 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 on the CCT, i.e., where the flow is delivered to. As a result, coflow scheduling can have little space to take effect for optimizing the CCTs of coflows.

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 the 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 linear programming (MILP) problem to seamlessly integrate endpoint placement and coflow scheduling for optimizing the average CCT of coflows. In the MILP problem,

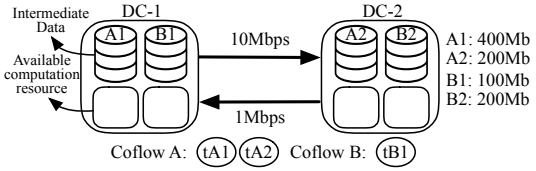


Fig. 2. An example with two coflows (i.e., A and B) and two datacenters (i.e., DC−1 and DC−2). For coflow A, there are two pieces of intermediate data (i.e., A1=400Mb and A2=200Mb) and two reduce tasks (i.e., tA1 and tA2). For coflow B, there is only one reduce task (tB1), and the intermediate data on 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.

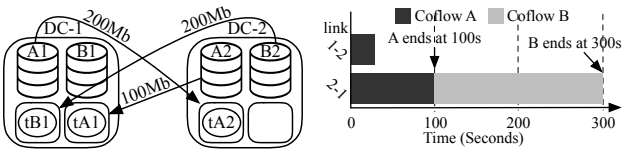
we take into account heterogeneous link bandwidth capacities, different coflow arrival times, and available computation resources of datacenters. Since the information about future coflows is usually unknown in advance, it is challenging to obtain the optimal solution for this problem. To solve this problem, 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 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 [9, 13]. 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%, compared to the state-of-the-art scheduling-only method [9].

The rest of this paper is organized as follows. In Section II, we describe our problem for this paper. In Section III, we develop the mathematical model and present our problem formulation. We show the design details of *SmartCoflow* in Section IV. The implementation details and experiment results are presented in Section V. Section VI discusses the related work and Section VII concludes this paper.

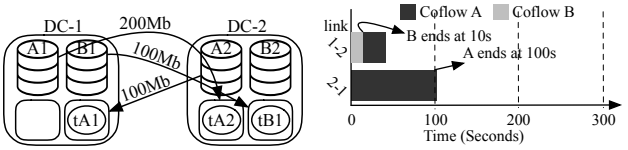
II. Problem Statement

In this paper, we consider a network with multiple geo-distributed datacenters denoted as $\mathcal{N} = \{1, \cdots, N\}$. 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



(a) Equal spread [16] (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.

arrives[1], 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)[2] to be launched on available computing slots in datacenters. We use U_i to denote the capacity of available computing slots in datacenter $i \in \mathcal{N}$.

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 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. 3(a) shows a possible case of reduce task placement with the naive equal spreading method [16], 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., 200Mb, from DC−1 to DC−2 while another one transfers half of the data A2, i.e., 100Mb, 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

[1]This assumption is reasonable because many recent studies (e.g., [7–9, 14]) 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 [6], and the number of reduce tasks can also be known through the TaskSet in Spark DAG scheduler [15].

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

meet at the inter-datacenter links, the optimal solution [9] is to schedule coflow B after coflow A, as shown in Fig. 3(b). 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. 4(a)), the data sizes of flows can be significantly reduced, i.e., the data size of the flow in coflow B is reduced to 100Mb. 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. 4(b)). The implies that both reduce task placement and scheduling must be jointly considered to minimize the average CCT.

The above example looks straightforward with the 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, yet is difficult to be accurately predicted. In most practical scenarios, we can only get the information of 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.

III. Modeling and Problem Formulation

In this section, we develop a mathematical model to study the problem of jointly considering reduce task placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters.

A. Mathematical model

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

$$I_{p,i}^k \in \{0,1\}, \forall i \in \mathcal{N}, \forall k \in \mathcal{K}, \forall p \in \{1,\dots,R_k\}. \quad (1)$$

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:

$$\sum_{i=1}^{N} I_{p,i}^k = 1, \forall k \in \mathcal{K}, \forall p \in \{1,\dots,R_k\}. \quad (2)$$

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:

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

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} [1]. 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:

$$\sum_{k=1}^{K} \sum_{p=1}^{R_k} I_{p,i}^k \leq U_i, i \in \mathcal{N}, \quad (4)$$

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

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 $C_{i,j}$.

B. 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**:

$$\underset{\{I_{p,i}^k\},\{B_{i,j}^k(x)\}}{\text{Minimize}} \quad \frac{1}{K}\sum_{k=1}^{K} T^k \quad (6)$$

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

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. (1) Problem **P1** is an MILP problem which is NP-hard in general [17], making it very hard to obtain the optimal solution. (2) 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.

IV. *SmartCoflow* Design

In this section, we propose an online coflow-aware optimization framework—*SmartCoflow*, to optimize the average CCT in the inter-datacenter networks by coordinating reduce task placement and coflow scheduling. In our design, once a coflow arrives at the network, we will formulate an integer linear programming (ILP) problem and apply a randomized

approximate algorithm to solve this ILP for deriving the reduce task placement and scheduling decisions for this coflow. This algorithm first relaxes the ILP into a linear programming (LP) and then uses 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, we rescale the bandwidth allocated to all existing coflows. We identify the following two goals when designing *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 a non-trivial competitive ratio when solving the original problem **P1**, such that the average CCT of coflows can be guaranteed with an upper bound.

In the rest of 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.

A. 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 III), 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).

Problem **P2** is a comprehensive ILP, which appears to be in the form of a Generalized Assignment Problem (GAP) that is typically NP-hard [17]. To solve this problem, we first relax $I_{p,i}$ into a continuous variable and accordingly obtain a LP, as shown in the following problem **P3**:

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

$$\text{Subject to: } 0 \le I_{p,i} \le 1, i \in \mathcal{N}, \forall p \in \{1,\dots,R\}, \tag{13}$$

$$\text{Eqs. (9), (10), (11), (12),}$$

which can be solved efficiently with standard linear programming solvers, such as MOSEK [18]. 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 3. 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** 1 (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** 2 (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

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

where $\mathbb{1}(I_{p,i}=1)$ is an indicator variable that is 1 if $I_{p,i}=1$ and 0 otherwise. Combining Theorem 1, the competitive ratio of Algorithm 1 can be calculated as $\rho = \max_{l_{ij}} \rho_{ij}$. Since $Pr(\rho > 2\ln 4M) \leq \sum_{l_{ij}\in\mathcal{E}} Pr(\rho_{ij} > 2\ln 4M)$, we fix l_{ij} and focus on the proof of $Pr(\rho_{ij} > 2\ln 4M) \leq \frac{1}{4M}$. Let $x_p := \frac{D_i}{RT}\mathbb{1}(I_{p,i}=1)$ 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) $E(x) = \frac{D_i\sum_{p=1}^{R} I_{p,i}}{RT} \leq C_{i,j}$, by Eqs. (11) and (12); 4) If $Pr(x_p{=}\frac{D_i}{RT}) > 0$ then $\frac{D_i}{RT} \leq C_{i,j}$, by Eqs. (10) (11) (12).

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

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

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

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

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

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

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

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

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

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

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

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

Substituting α to (15), we get

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

This implies the following inequality

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

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

B. 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). The following theorem demonstrates that our Algorithm 2 has a good competitive ratio for the original problem **P1**.

***Theorem** 3:* 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^k_{P2}, T^k_{P3} 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^k_{P2} when it occupies the network exclusively. Given the result of Theorem 1, we have $T_{P1} \geq \frac{1}{K}\sum_{k=1}^{K} T^k_{P2} \geq \frac{1}{K}\sum_{k=1}^{K} T^k_{P3}$. Therefore, we only need to compare the performance of our algorithm to $\frac{1}{K}\sum_{k=1}^{K} T^k_{P3}$. Specifically, let T_{alg} denote the average CCT achieved by Algorithm 2, and we focus on the proof of $T_{alg} \leq \rho\sum_{k=1}^{K} T^k_{P3} \leq K\rho \times T_{P1}$ in the rest proof process.

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

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

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**

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

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

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

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

Thus, proved. ■

V. 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.

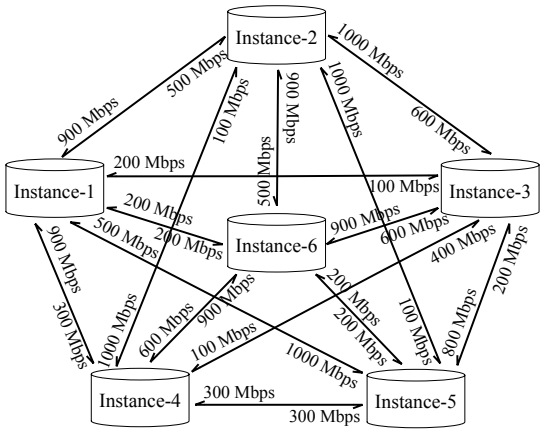


Fig. 5. The emulated testbed on Google's Cloud Compute Engine.

- **Varys-only**: schedules all coflows with the Shortest-Effective-Bottleneck-First (SEBF) coflow scheduler in Varys [9], 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 [1], and then schedules all flows using Varys SEBF scheduler [9]. 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.

A. Small-scale testbed implementation

We implement our *SmartCoflow* scheduler based on an open-source framework—Varys [9, 13]. 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 [19] 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 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.

TABLE I
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

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. 5 shows the detailed bandwidth constrained on each link which is randomly chosen from 100 to 1000 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.

TABLE II
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 III
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

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 1000 MB, as shown in Table I. The number of reduce tasks associated with each coflow is listed in Table II. Based on the 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 [15]. The calculated reduce task placement strategies under different schemes are illustrated in Table III.

Fig. 6 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

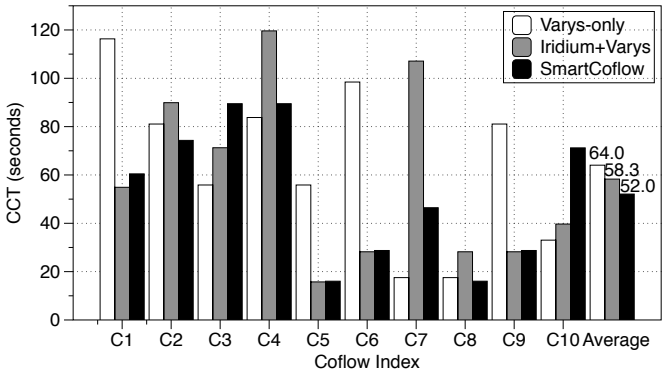
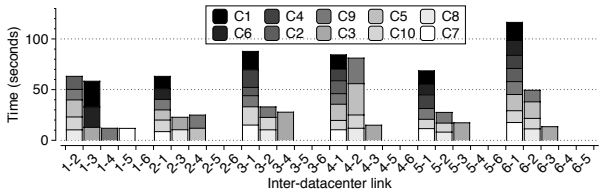
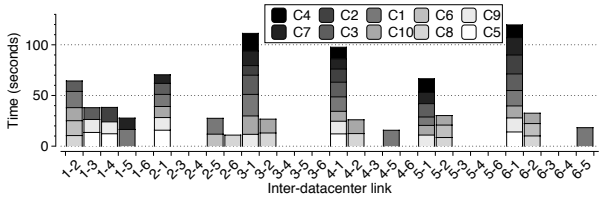


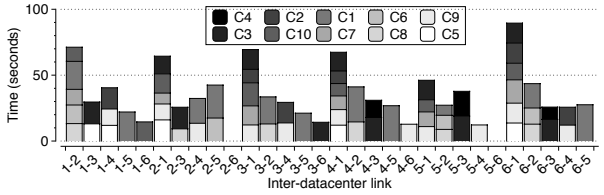
Fig. 6. The CCT of each coflow achieved by Varys-only, Iridium+Varys and *SmartCoflow* schemes, respectively.



(a) Varys-only scheme



(b) Iridium+Varys Scheme



(c) *SmartCoflow*

Fig. 7. The duration in which each coflow is scheduled on each link, under the (a) Varys-only, (b) Iridium+Varys, and (c) *SmartCoflow* schemes, respectively.

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 provide a detailed analysis on how the 10 coflows are scheduled under three different schemes, as shown in Fig. 7. The scheduling order under the Varys-only scheme is C7 $\rightarrow$ C8 $\rightarrow$ C10 $\rightarrow$ C5 $\rightarrow$ C3 $\rightarrow$ C9 $\rightarrow$ C2 $\rightarrow$ C4 $\rightarrow$ C6 $\rightarrow$ C1. While for Iridium+Varys scheme, the scheduling order is C5 $\rightarrow$ C9 $\rightarrow$ C8 $\rightarrow$ C6 $\rightarrow$ C10 $\rightarrow$ C1 $\rightarrow$ C3 $\rightarrow$ C2 $\rightarrow$ C7 $\rightarrow$ 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 [9]) of different coflows to appear at different links.

Though Iridium+Varys performs a little better than Varys-only scheme in terms of the average CCT (as shown in Fig. 6), it leaves most of the links idle and extends the tail CCT (i.e., the time when the last coflow finishes) from 116.32s to 119.61s. In contrast, *SmartCoflow* can obtain a better reduce task placement strategy, and schedule the 10 coflows with an order of C5 $\rightarrow$ C9 $\rightarrow$ C8 $\rightarrow$ C6 $\rightarrow$ C7 $\rightarrow$ C1 $\rightarrow$ C10 $\rightarrow$ C2 $\rightarrow$ C3 $\rightarrow$ C4. In this case, *SmartCoflow* reduces the average CCT to 52.0s. Moreover, the tail CCT is reduced to 89.48s. 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.

B. Large-scale trace-driven simulation

We develop a flow-level simulator based on an open-source framework `CoflowSim` [20], 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.

TABLE IV
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

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 [21]. Each datacenter has a uniform capacity of 100 computing slots. In our 40-datacenter setup, we vary the bandwidth between 100Mbps to 1Gbps, 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 [9, 10, 22]. The original trace is from a 3000-machine 150-rack cluster with 10:1 oversubscription 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 IV first presents the average, 95th percentile and maximum CCTs achieved by three different 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.

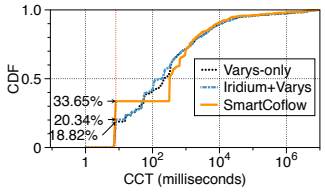


Fig. 8. The CDFs of CCTs under Varys-only, Iridium+Varys and *SmartCoflow* respectively. Note that the X-axes are in logarithmic scale.

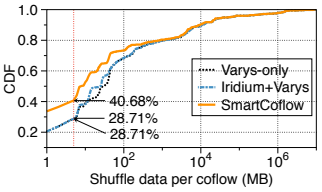


Fig. 9. The CDFs of coflow shuffle data under Varys-only, Iridium+Varys and *SmartCoflow* respectively. The X-axes are in logarithmic scale.

To understand on a microscopic level, we also plot the CDF of CCT for all coflows in Fig. 8. We can clearly find that 33.65% 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%. 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. This is because that 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. 9 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% 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%.

VI. 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 or endpoints of network flows closer to their data for optimizing the completion times of jobs inside a datacenter (e.g., [23–25]) or across

geo-distributed datacenters (e.g., [1, 6, 26]). However, the benefits of them are inherently limited because that they do not take into account the network flow scheduling after the reduce tasks or endpoints have been fixed. CLARINET [3] 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.

Coflow scheduling in datacenter networks: Existing efforts on coflow scheduling mainly fall into two categories. One of them focuses on achieving fairness among concurrent coflows (e.g., [14, 27]). Another category of work focuses on reducing the CCTs of coflows (e.g., Varys [9], Aalo [10], CODA [11], RAPIER [8], OMCoflow [12]). However, both categories of work consider the endpoints of network flow transfers to be fixed, making them insufficient to optimize the average CCT of coflows. *SmartCoflow*, on the other hand, leveraging the endpoint flexibility of network flows when scheduling coflows, can significantly improve the performance on the average CCT of coflows.

VII. Conclusions

In this paper, we jointly consider the endpoint placement and coflow scheduling to minimize the average CCT of coflows across geo-distributed datacenters. To characterize the combination of endpoint placement and coflow scheduling in the inter-datacenter networks, we develop a rigorous mathematical model and formulate an MILP problem. Then, we propose a coflow-aware optimization framework—*SmartCoflow*, to solve the MILP in an online manner. 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. Rigorous theoretical analysis has shown that without any prior knowledge of future coflows, *SmartCoflow* can achieve a good competitive ratio in minimizing the average CCT of the coflows. Finally, we have implemented *SmartCoflow* as a real-world coflow-aware scheduler and evaluated *SmartCoflow* through both small-scale testbed implementation and large-scale trace-driven simulations. From the experimental results, we conclude that *SmartCoflow* can offer significant performance improvement on average CCT when compared to the prevailing methods.

Acknowledgment

This work is supported by the National Key Research and Development Program of China No. 2016YFB1000205; the State Key Program of National Natural Science of China under Grant 61432002; the NSFC under Grant 61672379, Grant 61772112, Grant 61370199 and Grant 61702365; the Dalian High-level Talent Innovation Program under Grant 2015R049; the Natural Science Foundation of Tianjin under Grant 17JCQNJC00700. Xiaobo Zhou and Heng Qi are the corresponding authors: xiaobo.zhou@tju.edu.cn, hengqi@dlut.edu.cn.

References

[1] Q. Pu, G. Ananthanarayanan, P. Bodik, S. Kandula, A. Akella, P. Bahl, and I. Stoica, "Low latency geo-distributed data analytics," in *Proc. of ACM SIGCOMM*, 2015.

[2] C.-C. Hung, L. Golubchik, and M. Yu, "Scheduling jobs across geo-distributed datacenters," in *Proc. of ACM SoCC*, 2015.

[3] R. Viswanathan, G. Ananthanarayanan, and A. Akella, "Clarinet: Wan-aware optimization for analytics queries," in *Proc. of USENIX OSDI*, 2016.

[4] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. of ACM Workshop on Hot Topics in Networks*, 2012.

[5] K. Hsieh, A. Harlap, N. Vijaykumar, D. Konomis, G. R. Ganger, P. B. Gibbons, and O. Mutlu, "Gaia: Geo-distributed machine learning approaching lan speeds." in *Proc. of NSDI*, 2017.

[6] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *Proc. of IEEE INFOCOM*, 2016.

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

[8] Y. Zhao, K. Chen, W. Bai, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. of IEEE INFOCOM*, HK, 2015.

[9] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. of ACM SIGCOMM*, Chicago, IL, USA, 2014.

[10] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," in *Proc. of ACM SIGCOMM*, 2015.

[11] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proc. of ACM SIGCOMM*, 2016.

[12] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau, "Efficient online coflow routing and scheduling," in *Proc. of ACM MobiHoc*, 2016.

[13] "Varys: Efficient clairvoyant coflow scheduler." [Online]. Available: https://github.com/coflow/varys

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

[15] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. of USENIX NSDI*, 2012.

[16] S. Venkataraman, A. Panda, G. Ananthanarayanan, M. J. Franklin, and I. Stoica, "The power of choice in data-aware cluster scheduling." in *Proc. of USENIX OSDI*, 2014.

[17] J. K. Karlof, *Integer programming: theory and practice*. CRC Press, 2005.

[18] 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*. Springer, 2000, pp. 197–232.

[19] "Breeze: A numerical processing library for scala." [Online]. Available: http://www.scalanlp.org

[20] "Flow-level simulator for coflow scheduling used in varys and aalo." [Online]. Available: https://github.com/coflow/coflowsim

[21] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer, "Achieving high utilization with software-driven wan," in *Proc. of ACM SIGCOMM*, 2013.

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

[23] M. Chowdhury, S. Kandula, and I. Stoica, "Leveraging endpoint flexibility in data-intensive clusters," in *Proc. of ACM SIGCOMM*, 2013.

[24] F. Ahmad, S. T. Chakradhar, A. Raghunathan, and T. Vijaykumar, "Shufflewatcher: Shuffle-aware scheduling in multi-tenant mapreduce clusters," in *Proc. of USENIX ATC*, 2014.

[25] 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. of ACM SIGCOMM*, 2015.

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

[27] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *Proc. of IEEE INFOCOM*, 2016.