What Appears Suboptimal May Surprise You: A Fixed-Rate Scheduling Policy for Geo-Distributed CoFlows

Haoyu Liu
School of Information
Xiamen University
Xiamen, China
liuhaoyu@stu.xmu.edu.com

Feiyan Ding
School of Information
Xiamen University
Xiamen, China
feiyan.ding.xmu@gmail.com

Yao Wang
School of Information
Xiamen University
Xiamen, China
yaowong@stu.xmu.edu.com

Qiao Xiang*
School of Information
Xiamen University
Xiamen, China
xiangq27@gmail.com

Jiwu Shu
School of Information
Xiamen University
Xiamen, China
jwshu@xmu.edu.cn

Haizhou Du
School of CS and Technology
University of Electric Power of Shanghai
Shanghai, China
duhaizhou@gmail.com

Linghe Kong
Dept. of CS and Engineering
Shanghai Jiao Tong University
Shanghai, China
linghe.kong@sjtu.edu.cn

Xue Liu
School of Computer Science
McGill University
Montreal, Canada
xueliu@cs.mcgill.ca

***Abstract*—All existing coflow scheduling algorithms compute dynamic-rate schedules that change the rates of flows during transmission. In this paper, we make a crucial finding: although dynamically adjusting the rates of flows could lead to a better coflow completion time (CCT) in theory, it would introduce additional pressures on the congestion control mechanism in the underlying network, which result in poor CCT in practice. This difference between theoretical CCT and practical CCT is further exacerbated in wide-area networks, where the topology does not provide any bisection guarantee as data center networks do. To this end, we designed a fixed-rate coflow scheduling policy called FSCO. Although in theory, the best fixed-rate schedule is usually suboptimal, it keeps the in-flight traffic relatively steady, reducing the risk of triggering congestion control. The core of FSCO is an efficient scheduling algorithm based on the classic network utilization maximization (NUM) framework. We implement a prototype of FSCO and evaluate its performance extensively using real-world topologies and coflow traces. Experimental results show that the total CCT reduces up to 30% compared to baselines while yielding up to 12× speedups compared to the solver.**

***Index Terms*—Coflow Scheduling, Network Utility Maximization, Dual decomposition**

I. Introduction

With the rapid improvement of computer and information infrastructures, many geo-distributed applications and frameworks are developed to provide data analytics and large model training services, such as MapReduce [1], Spark [2], Bulk Synchronous Parallel (BSP) [3]. These jobs contain many individual flows between consecutive stages, indicating that their performance is determined by the collective behavior of all these flows. We usually regard these flows as a coflow [4]. Since the results of jobs are significant to subsequent processing, it is critical to schedule geo-distributed coflows to achieve high throughput and short job completion time in a low bandwidth scene or bandwidth-scarce scene such as Wide-Area Network (WAN).

In recent years, to improve the performance of data-parallel jobs in various scenes, many schemes [5]–[16] have focused on managing coflows to avoid the bottleneck of networks. One effort of them puts attention on coflow scheduling in data centers. Varys [5] and Sincornia [6] focus on data center coflow scheduling with prior knowledge, while some other schemes focus on scheduling in data centers without prior knowledge, such as [7], [8], [11]. All these above works abstract the network as a single big switch connected to all the machines, regardless of the link bandwidth. Unlike the non-blocking scene in data centers, the bandwidth on the WAN is scarce, leading to performance degradation when applying these scheduling schemes to the transmission of geo-distributed coflows. Thus, many works [9], [10], [12], [13] propose solutions for coflow transmission over the WAN which takes bandwidth into account. To achieve better performance, these scheduling schemes change the flow rate during transmission, which dynamically sends data to the network stack.

Emerging trends [9], [10], [17]–[19] in how jobs are scheduled only consider the optimal solution of application layer scheduling in mathematics.Unfortunately, based on our findings in many simulations, there are gaps between the practical results and theoretical results due to in-network conditions. Many previous scheduling schemes assume the low network layer as a stable system, which is impractical in modern network scenes, especially in WAN. We believe that this performance gap is caused by the changes in network states, which constantly trigger congestion control, reducing

* Corresponding author

the transmission rate. Thus, it is crucial to design a scheduling framework whose performance is little affected by the underlying network, and improve the practical CCT in real-world transmission.

To tackle these issues, we present a fixed-rate geo-distributed coflows scheduling framework. It can significantly improve performance and alleviate the gap between theoretical and practical results. We argue that rapid changes in flow rate would result in queuing, buffer overflow, and other issues that could influence the in-network conditions. Thus, congestion control and retransmission would be triggered when in-network conditions change largely. Dynamic rate scheduling may achieve optimal solutions in theory, while frequent rate change has the risk of performance degradation. Thus, we propose a fixed rate scheduling framework, which is suboptimal in theory but performs better in practice because this scheduling framework can put data into the network stack at a stable rate. We formulate the model as a convex problem and develop a fast and efficient algorithm based on dual decomposition using modern CPU and GPU to process the large-scale data. An analytical solution is also introduced to accelerate the problem-solving.

The **main contributions** of this paper are as follows:

- We find that the common dynamic rate scheduling schemes perform badly when applied to real network systems. Thus, we propose a fixed-rate scheduling framework to address geo-distributed coflow scheduling in WANs. It can avoid additional pressures on the congestion control mechanism. To the best of our knowledge, it is the first time that scheduling all coflows at a fixed rate simultaneously.
- We introduce dual decomposition to solve the subproblems in parallel, which can use CPU and GPU resources to speed up the coflow scheduling. An analytic solution to the subproblem is also proposed to greatly reduce the time of subproblem solving.
- Experimental results show that FSCO reduces the CCT up to 30% compared to baselines and yields up to 12x speedups compared to the Gurobi [20] solver.

II. Background

In modern computer communication schemes, many applications are sensitive to network congestion, where the bad allocation of jobs would hurt the performance of the network. Thus, many network service providers introduce flow scheduling, with the hope of alleviating the bottleneck of networks to achieve high performance and satisfy the needs of users.

Many existing programming frameworks [1]–[3] have job-specific communication requirements, generating a lot of data-parallel jobs. In these frameworks, one job cannot begin until all the preceding communication flows have finished. As defined by Chowdhury [5], we regard the set of flows sharing a common performance goal as a coflow.

For details, we assume that a coflow has many independent flows: the input of each flow is independent of the output of another flow in this coflow. The scheduling of coflow, just like the scheduling of network flow, is a crucial block to the network and has attracted attention in the network area. Many works [6], [10], [18] try to schedule the coflow to acquire better performance. Chowdhury proposes Varys [5] to schedule coflows in the data center. Varys models the scheduling problem as a concurrent open shop problem and solves it using a heuristics method. To make the abstraction more useful in the network area, Chowdhury proposes a novel formulation [10], which could adapt to various scenes, especially in WAN, whose bandwidth is a scarce resource.

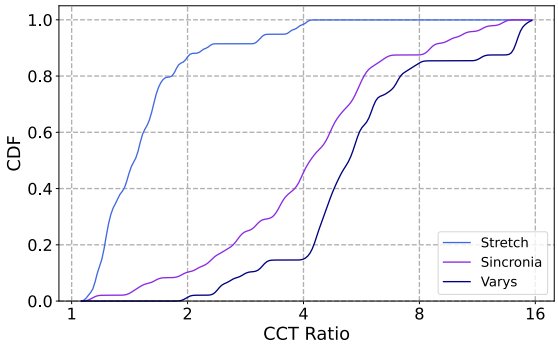


Fig. 1. The ratio of the practical CCT and the theoretical CCT of different algorithms in simulations.

All these works allocate the bandwidth to flows with dynamic rates. However, we find the performance of these decisions is different from the theoretical results through simulations on NS3 [21]. We tested three different algorithms *Stretch* [10], *Varys* [5] and *Sincronia* [6]. Fig. 1 shows the ratio of the practical CCT and the theoretical CCT of different algorithms in simulations. We can see that the practical CCT of most traffic is much larger than the theoretical CCT.

Moreover, existing works model the coflow problem as an optimization problem and solve it using a solver like Gurobi [20]. Linear Programming (LP) and Mixed Integer Linear Programming (MILP) are the most common algorithms to solve these problems. Due to the large computing time of MILP, SmartCoflow [12] formulates the coflow scheduling problem as a MILP and solves it by an approximate algorithm. Unfortunately, even though the approximation and the parallelizing LP solvers [22] are efficient, it has poor performance with multiple CPUs and GPUs.

To tackle these problems, we first propose the fixed-rate scheduling framework to alleviate the impact of in-network conditions on scheduling. Parallel computing and GPU assistance are also introduced to accelerate the optimization.

III. Design of FSCO

In this section, we introduce the fixed-rate scheduling framework of geo-distributed coflows. In Subsection III-A, we first describe the overview of FSCO design. We develop a model of FSCO in Subsection III-C to minimize the total CCT of coflows across geo-distributed networks. After that, we

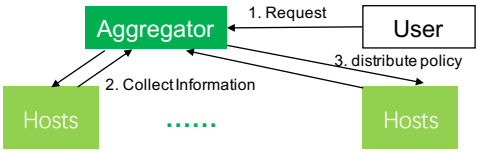


Fig. 2. The architecture of FSCO

propose a method based on dual decomposition to accelerate the solution of the problem in Subsection III-D.

A. Overview

The design of FSCO is motivated by the performance degrading shown in Section II, whose goal is to optimize the total CCT in the geo-distributed network. FSCO could determine the rate of flows in this scheduling period, bringing a good performance on CCT with prior knowledge like the setting in [5], [10].

Considering the actual environment, we use a master agent to manage coflow information and give the scheduling policy. This scheduling policy is a relatively stable scheduling decision due to the fixed rate flow setting and will be transmitted to the networks to decide the rate of flows. Besides, we transform the problem into a dual problem and solve the part of it with a proper analytic solution. We also decompose the dual problem into many independent subproblems to accelerate the speed from the perspective of parallel execution. We put the simple computational logic process into GPUs, and the other remains in CPUs to utilize multi-threads of modern CPU and GPU. The architecture of the FSCO is shown in Fig. 2, where the scheduling policy is computed by the aggregator and distributed to the hosts.

We design the fixed rate coflow scheduling policy with Algorithm 1. The FSCO assumes all the information of coflows are known before and can be accessed from the input $\mathbf{\Omega}$ as many other works [10], [12]. Moreover, the link capacity is also needed to calculate optimal scheduling.

Algorithm 1 The FSCO Workflow

Input: $\mathbf{\Omega}$ = Coflow information set for current coflows which could be scheduled. $\boldsymbol{c}$ = Capacity vector of links.
Output: $\boldsymbol{x}$ = the rate vector of all the flows in all coflows.

1: **Procedure** NetworkBandwidthAllocation($\mathbf{\Omega}, \boldsymbol{c}$)
2: Parse the coflow and network information ($\mathbf{\Omega}, \boldsymbol{c}$)
3: $\boldsymbol{x}$ = MiniTCCT($\mathbf{\Omega}, \boldsymbol{c}$)
4: Allocate the bandwidth $\boldsymbol{x}$ to the flows
5: Transmit the flows at the rate of $\boldsymbol{x}$
6: **end procedure**

B. Network Representation

Coflow is an abstract for many flows that have the same goals. Just like mentioned above, the tasks of parallel-data applications can be abstracted into coflows, where the $\mathbb{C}$ is a set of coflows, defined as $\mathbb{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_N\}$, and the $|\mathbb{C}|$ represents the number of coflows. For each coflow j, there are many flows i within this coflow j. We denote this set of flows as $\mathcal{C}_j = \{f_j^1, ..., f_j^i, ...f_j^k\}$. Each flow f_j^i has associate rate x_j^i and demand σ_j^i, which means the flow f_j^i transmits σ_j^i of data at a rate of x_j^i.

The networks have many properties, such as links and paths. We denote the set of links as $\mathbb{E} = \{e_1, ..., e_l, ...e_L\}$ and the capacity of link l as c_l. Moreover, one flow f_j^i will transverse one path consisting of many links to the target node, where the path of this flow can be defined as $\mathcal{P}(i,j) = \{e_u, ..., e_v\}$. We also denote the set of flows that traverse link l as $\mathcal{S}_l = \{f_j^i, ..., f_J^I\}$. The notation $\mathbb{P} = \{\mathcal{P}(i,j)_1, ..., \mathcal{P}(i,j)_N\}$ means the path set and $\mathbb{S} = \{\mathcal{S}_1, ..., \mathcal{S}_l, ..., \mathcal{S}_N\}$ means the set of flow set.

To represent the state of the link, we denote μ_l as the shadow price of link l, and we define $p_j^i = \sum_{l \in \mathcal{P}(i,j)} \mu_l$ as the flow price for flow i in coflow j. The main notations are shown in Table I.

C. Minimize Total Coflow Completion Time

We begin this part by thinking about the importance of coflow scheduling. To exploit the bandwidth of this network and finish all the tasks in the shortest time, we should adjust the flow rate for better performance. Thus, the formulation of coflow scheduling can be formulated as follows:

Input: The FSCO should get all the information before making scheduling decisions like previous works [6], [10]. The set of coflows $\mathbb{C}$ is the information abstract of network flow, and the set of links $\mathbb{E}$ is the information abstract of network topology. Moreover, we could get the flow volume σ_j^i set, the path of one flow $\mathcal{P}(i,j)$, and the network resource information such as the link capacity c_l set.

Output: We hope to get the flow scheduling policy to make the data-parallel applications better. Thus, for each network flow information and network topology information, the FSCO will return a vector of rate for all flows, and the x_j^i denotes the rate of flow f_j^i.

Now, we know what we have and want, so we can easily formulate the problem of fixed rate coflow scheduling to minimize the total CCT across geo-distributed data centers. We can formulate the total completion time of all the coflows at the current time as follows:

$$Minimize \sum_j \max_{f_j^i \in \mathcal{C}_j} \frac{\sigma_j^i}{x_j^i}, \tag{1}$$

Subject to:

$$\sum_{f_j^i \in S_l} x_j^i <= c_l, \forall \mathcal{C}_j \in \mathbb{C}, \forall f_j^i \in \mathcal{C}_j, \forall l \in \mathbb{E}, \tag{1a}$$

$$x_j^i >= 0, \forall \mathcal{C}_j \in \mathbb{C}, \forall f_j^i \in \mathcal{C}_j. \tag{1b}$$

Note that in our formulation, the objective is to minimize the total completion time of coflows in coflow set $\mathbb{C}$. We denote the completion time of a flow f_j^i as $t_j^i = \frac{\sigma_j^i}{x_j^i}$. And the completion time of a single coflow j is denoted as a maximize function $\max_{f_j^i \in \mathcal{C}_j} \frac{\sigma_j^i}{x_j^i}$ because the completion time of one

coflow $\mathcal{C}_j$ is defined by the slowest one $f_j^i \in \mathcal{C}_j$. Moreover, the σ_j^i denotes the traffic demand of the flow f_j^i and the x_j^i denotes the rate of flow f_j^i.

The capacity of links in the network is an inherent attribute, which cannot be violated in real scenes. Constraint (1a) is used to ensure that the summation of bandwidth allocated to all flows on a link l should not exceed the link bandwidth capacity, denoted by c_l. Constraints (1b) states that the rate of all flows should not be negative because the negative rate is impossible for flow transmission.

Unfortunately, although the formulation is simple, it is time-consuming to achieve the optimal solutions using solver [20] when the scale of problems becomes large.

TABLE I
SUMMARY OF MAIN NOTATIONS

Notation	Description
i	flow index
j	coflow index
l	link index
f_j^i	the i_{th} flow in j_{th} coflow
x_j^i	rate of flow i in coflow j
e_l	the l_{th} links in the network
σ_j^i	volume of flow i in coflow j
$\mathbb{C}$	the coflow set
$\mathcal{C}_j$	the flow set of coflow j
$\mathbb{E}$	the link set
$\mathcal{P}(i,j)$	the path of flow i in coflow j
$\mathcal{S}_l$	the flow set which traverses link l
$\mathbb{P}$	the path set of flows contains P(i,j)
$\mathbb{S}$	the aggregate set which contains S(l)
c_l	the capacity of link l
μ_l	the shadow price of link l
p_j^i	the price of flow f_j^i

D. Dual Decomposition and Parallel Solution

A parallel solution is a common way to speed up the solution. To solve the problem with better performance, we introduce the dual decomposition to enhance parallelism. It can transform the primal problem into a dual problem, which eliminates the coupling of many tasks. Thus, a dual problem can be divided into many independent subproblems and these subproblems can be solved in parallel.

It is crucial to analyze the optimal value of the primal problem and the dual problem. Actually, from formulation (1), it is obvious that the primal problem is a convex problem, which means the duality gap is zero and we can get the optimal solution to the primal problem if we solve the dual problem.

As mentioned above, one of the features of the dual problem is that it can be decoupled into multiple independent subproblems, which could be solved in parallel, meeting the need for fast transmission.

Usually, we can get the Lagrangian function based on Lagrangian multipliers, which is the most basic step to get the dual problem, just like what NUM [23] does. In order to observe the relationship between different variables, we have made some transformations, so as to solve the Lagrangian function more simply. Thus, the Lagrangian function can be formatted as Equation (2).

$$\begin{aligned} L(\boldsymbol{x}, \boldsymbol{\mu}) &= \sum_j \max_i \frac{\sigma_j^i}{x_j^i} + \sum_j \sum_i x_j^i \sum_{l \in P(i,j)} \mu_l - \sum_l \mu_l c_l \\ &= \sum_j (\max_i \frac{\sigma_j^i}{x_j^i} + \sum_i x_j^i \sum_{l \in P(i,j)} \mu_l) - \sum_l \mu_l c_l. \end{aligned} \tag{2}$$

In Equation (2), μ_l is a dual variable, which we use to solve the coupling issues due to the link capacity guarantee.

After we get the Lagrangian function, the dual function can be obtained by minimizing the Lagrangian function. The decision variables of this function are $\boldsymbol{x}$ and the dual function is thus:

$$\begin{aligned} D(\boldsymbol{\mu}) &= \min_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\mu}) \\ &= \sum_j \min_{\boldsymbol{x}} (\max_i \frac{\sigma_j^i}{x_j^i} + \sum_i x_j^i \sum_{l \in P(i,j)} \mu_l) - \sum_l \mu_l c_l. \end{aligned} \tag{3}$$

The first right term of the dual function (3) can be separated into j subproblems $\min_{\boldsymbol{x}}(\max_i \frac{\sigma_j^i}{x_j^i} + \sum_i x_j^i \sum_{l \in P(i,j)} \mu_l)$, which could be solved independently.

To get the optimal solution of the primal problem in another method, we formulate the dual problem (4). The formulation (4) is also a convex optimization problem with the decision variables (dual variables or shadow prices) $\boldsymbol{\mu}$.

$$\max_{\boldsymbol{\mu}} (\sum_j \min_{\boldsymbol{x}} (\max_i \frac{\sigma_j^i}{x_j^i} + \sum_i x_j^i \sum_{l \in P(i,j)} \mu_l) - \sum_l \mu_l c_l), \tag{4}$$

Subject to:

$$\boldsymbol{x}, \boldsymbol{\mu} \geq \mathbf{0}. \tag{4a}$$

All we need is to solve the dual problem to obtain the proper rate allocation decision. Generally, the solution to the dual problem requires multiple iterations, including the optimization of the Lagrangian function and dual problem. For example, we initialize the $\boldsymbol{\mu}$ and solve the function (3). After that, we freeze $\boldsymbol{x}$ and solve the problem (4). The final result will be obtained after multiple iterations to reach convergence. However, the multiple iterations require much time to achieve the optimal solutions, which is another challenge for us.

We take some methods to accelerate the speed of the computation. Considering the problem (3), it is time-consuming to solve the independent subproblem $\min_x(\max_i \frac{\sigma_j^i}{x_j^i} + \sum_i x_j^i \sum_{l \in P(i,j)} \mu_l)$ again and again. To alleviate the stress which this part brings, we could get the analytic solution for

this subproblem, and the solution is $x_j^{i^*} = \frac{\sigma_j^i}{\sqrt{\sum_i (p_j^i \sigma_j^i)}}$ where $p_j^i = \sum_{l \in P(i,j)} (\mu_l)$.

Theorem 1: The optimal solution of the subproblem is $x_j^{i^*} = \sigma_j^i / \sqrt{\sum_i (p_j^i \sigma_j^i)}$.

Proof of Theorem 1: Consider a coflow $\mathcal{C}_j$, we could reformulate the subproblem as:

$$\min_{\boldsymbol{x}'} (\max_i \frac{\sigma_j^i}{x_j^i} + \sum_i x_j^i \sum_{l \in P(i,j)} \mu_l), \tag{5}$$

where $x^{'} = (x_j^1, x_j^2, \cdots)$ is the vector of the rate of flows f_j^i in coflow $\mathcal{C}_j$.

We denote $t_j^i = \frac{\sigma_j^i}{x_j^i}$ as the flow completion time of flow f_j^i and $t_j^{max} = \max_{f_j^i}(t_j^i)$ as the completion time for coflow $\mathcal{C}_j$. Also, we get $p_j^i = \sum_{l \in P(i,j)} (\mu_l)$. Thus, the Equation (5) can be transformed as:

$$\min_{\boldsymbol{t}} (t_j^{max} + \sum_i \frac{\sigma_j^i}{t_j^i} p_j^i). \tag{6}$$

Consider the second term of the Equation (6), we could get $\frac{p_j^i \sigma_j^i}{t_j^i} \geq \frac{p_j^i \sigma_j^i}{t_j^{max}}$ because of $t_j^i \leq t_j^{max}$ for any i in $\boldsymbol{C_j}$.

Thus, the second term of the Equation (6) can be relaxed as $\sum_i \frac{p_j^i \sigma_j^i}{t_j^i} \geq \sum_i \frac{p_j^i \sigma_j^i}{t_j^{max}}$, and the equality obtains *iff* $t_j^1 = t_j^2 = \cdots = t_j^{max}$.

Further, the relaxed problem $(t_j^{max} + \sum_i \frac{p_j^i \sigma_j^i}{t_j^{max}}) \geq 2\sqrt{\sum_i p_j^i \sigma_j^i}$ *iff* $t_j^{max} = \sqrt{\sum_i p_j^i \sigma_j^i}$.

Therefore, the subproblem can be written as $t_j^{max} + \sum_i \frac{p_j^i \sigma_j^i}{t_j^i} \geq t_j^{max} + \sum_i \frac{p_j^i \sigma_j^i}{t_j^{max}} \geq 2\sqrt{\sum_i p_j^i \sigma_j^i}$. And the optimal solution $2\sqrt{\sum_i p_j^i \sigma_j^i}$ can be obtained *iff* $t_j^{max} = \sqrt{\sum_i p_j^i \sigma_j^i}$ and $t_j^1 = t_j^2 = \cdots = t_j^{max}$.

In a nutshell, we can get the optimal solution $x_j^i = \frac{\sigma_j^i}{\sqrt{\sum_i (p_j^i \sigma_j^i)}}$ of this subproblem. And the optimal value is $2\sqrt{\sum_i (p_j^i \sigma_j^i)}$.

■

We should calculate the optimal solutions of the problem (4) to acquire the new dual variables μ for the next step of problem (3). And the object function of problem (4) can be written as:

$$\max_{\boldsymbol{\mu}} (\sum_{\mathcal{C}_j} (2\sqrt{\sum_{f_j^i} (p_j^i \sigma_j^i)}) - \sum_l \mu_l c_l). \tag{7}$$

For fast iteration, we just update the value using gradient descent rather than the optimal value. The workflow of this algorithm is introduced briefly in the Algorithm 2 in a serialized form.

To fully utilize the parallel capability of modern CPUs and GPUs, FSCO introduces unified memory and arranges the execution flow as Fig. 3. The CPU will calculate the x_j^i for f_j^i. Lines 18 - 25 in Algorithm 2 introduce the details of the Rate Calculator in Fig. 3. The Step 1A calculates the dual gap to determine whether the algorithm converges. At the same time, the CPU will calculate the gradients for new dual variables. Due to the synchronism of Step 2, Step 3A and Step 3 in Fig. 3, we design a staleness method, where the Step 3 in iteration k uses the results of Step 3A in iteration $k-1$. This design masks the data replication overhead under the collaborative computing of CPU and GPU. The multi-threads of CPU and GPU can accelerate all the processes above. The rest details of these processes are presented in Algorithm 3. And we will show the results in Section IV.

Algorithm 2 The Minimize Algorithm

1: **procedure** MiniTCCT($CoflowInfo\ \boldsymbol{\Omega}, Linkstate\ \boldsymbol{L}$)
2: Get $\boldsymbol{\sigma}, \boldsymbol{C}$ from $\boldsymbol{\Omega}$, $\boldsymbol{c}, \boldsymbol{S}, \boldsymbol{P}$ from $\boldsymbol{L}$
3: Initialize $\boldsymbol{\mu}$
4: **while** $True$ **do**
5: **for** $j = 1$ to $|\boldsymbol{C}|$ **do**
6: $\boldsymbol{p_j} = \{p_j^i = \sum_{l \in P(i,j)} \mu_l\}_i$
7: $\boldsymbol{x_j}, v_j = MinimizeL(\boldsymbol{\sigma_j}, \boldsymbol{p_j})$
8: **end for**
9: $TCCT = (\sum_j v_j - \sum_l \mu_l c_l)$;
10: $\boldsymbol{if}\ converged\ \boldsymbol{then}\ break$
11: **for** $r = 1$ to $|\boldsymbol{c}|$ **do**
12: $x_r = \sum_{f_j^i \in S(r)} x_j^i$
13: $\mu_r = MaxmizeDual(\mu_r, c_r, x_r)$
14: **end for**
15: **end while**
16: return $\boldsymbol{x}$
17: **end procedure**

18: **procedure** MinimizeL($Demand\ \boldsymbol{\sigma_j}, Shadowprice\ \boldsymbol{p_j}$)
19: $t_j^{max} = \sqrt{\sum_i p_j^i \sigma_j^i}$
20: $v_j = 2\sqrt{\sum_i p_j^i \sigma_j^i}$
21: **for** $i = 1$ to $|C_j|$ **do**
22: $x_j^i = \sigma_j^i / t_j^{max}$
23: **end for**
24: return $\boldsymbol{x_j}, v_j$
25: **end procedure**

26: **procedure** MaxmizeDual($Linkprice\ \mu_l, Linkcapacity\ c_l$, $Linkrate\ x_l$)
27: Receives the sum of the rate of the flows traveling through link l and the corresponding link price and capacity
28: Computes a new price using gradient descent:
29: $\mu_l = [\mu_l + \gamma(x_l - c_l)]^+$
30: return μ_l
31: **end procedure**

IV. EVALUATION

In this section, we present the results of FSCO through simulations on NS3 [21] with different network and data scales to demonstrate the effectiveness of our fixed-rate setting and

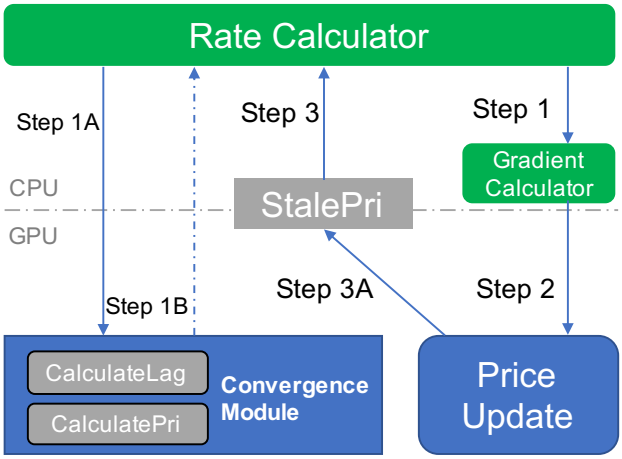


Fig. 3. The Execution Flow of FSCO

Algorithm 3 The Solution Process

procedure CalculateGrad ($Link\ e_l, Flowrate\ \boldsymbol{x}$)
$g_l = -c_l$
for $f_j^i \in \mathcal{S}_l$ **do**
  $g_l = g_l + x_j^i$
end for
return g_l

procedure UpdatePrice ($Gradient\ g_l, Shadowprice\ \mu_l$)
$\mu_l = [\mu_l + \gamma(g_l)]^+$
return μ_l

procedure CalculaltePri ($Coflow\ \mathbb{C}, Demand\ \boldsymbol{\sigma}$)
$tt_j^i = \sigma_j^i / x_j^i$
$CCT_j = \sum_i tt_j^i$
$value = \sum_j CCT_j$
return $value$

procedure CalculalteLag ($Link\ \mathbb{E}, LinkPrice\ \boldsymbol{\mu}, V\ v$)
$v_l = e_l * \mu$
$vsum = \sum_l v_l$
$value = v - vsum$
return $value$

parallel speedup. In Subsection IV-A, we will introduce our experiment settings, including the experiment environment and methodology. We evaluate and analyze the results of solving time and coflow completion time in Subsection IV-B.

A. Experiment Setting

To mimic the transmission in the network with different network stack conditions and scarce bandwidth resources, we implement the experiments on different network and data settings, reflecting the performance of networks with various settings.

The Network Topology. We simulate the geo-distributed networks by Topology Zoo [24] which is widely used to generate the networks in related works [25]–[27]. We select some network topologies with different scales, whose nodes are connected with 100Mbps links. We also set the propagation delay to 2μs or 2ms. The switch nodes in these topologies are modeled as standard output-queued switches, with a shallow buffer of size 8KB per port. We constructed some networks based on the above information, and the topology setting is shown in Table II.

TABLE II
THE SIZE OF TOPOLOGY WE SELECTED IN THE EXPERIMENTS

Name	Ai3	Airtel	Attmpls	Ernet	Globenet
Nodes	10	16	25	30	67
Links	9	37	57	32	113

Congestion Control Algorithms. This is a crucial factor influencing the performance of transmission in a network. We choose three common congestion control algorithms to simulate the real network.

- *DCTCP* [28]: DCTCP is a typical congestion control algorithm that uses ECN as a congestion signal.
- *TCPLinuxReno* [29]: TCPLinuxReno is the most mature congestion control algorithm that is applied widely.
- *TCPCubic* [30]: TCPCubic is a default congestion control algorithm on Linux with high performance.

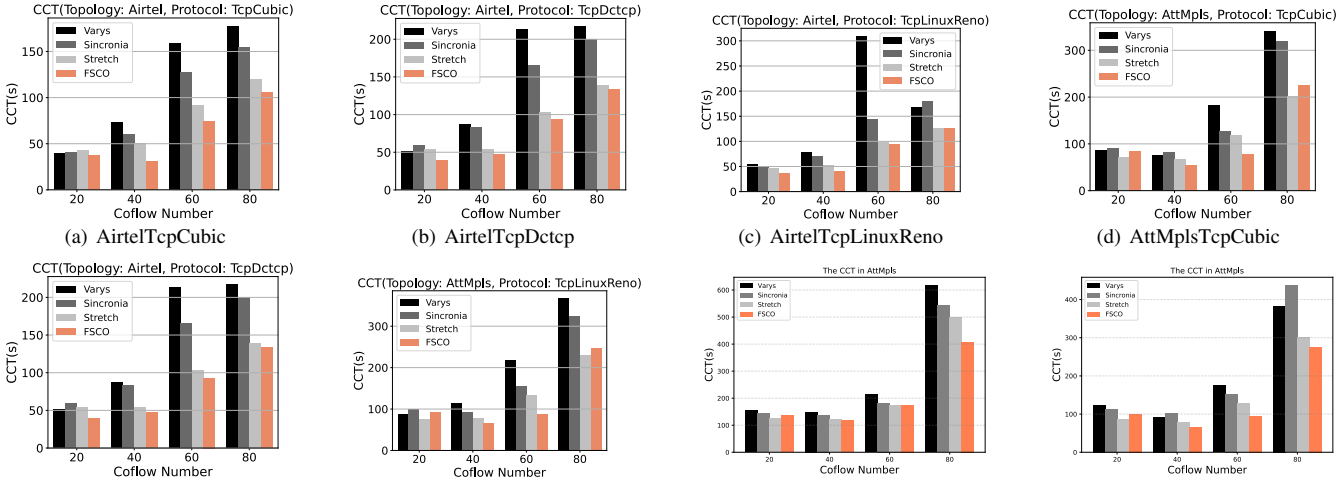
Coflow Benchmark. To analyze the data-parallel jobs in WAN, we get the coflow benchmark from Facebook [5] which is widely used in coflow scheduling research. To facilitate use and adapt to different topology sizes, we select part of the coflows from these data as our experimental data with different coflow widths, coflow numbers, and coflow volumes. We generate new coflows for the

Baseline. We compare the following schemes with FSCO.

- *Stretch* [10]: A Dynamic rate scheduling algorithm that used a time-indexed LP formulation on WAN.
- *Varys* [5]: An algorithm that used the smallest-effective-bottleneck-first heuristic to address coflow scheduling.
- *Sincronia* [6]: An algorithm that achieves average coflow completion time within 4× of the optimal by giving a "right" ordering of coflows.

Performance Metrics. In this evaluation, we define the performance improvement as $ratio_{imp} = \frac{CCT_1 - CCT_2}{CCT_1}$, where the CCT_1 is the total CCT of one algorithm while CCT_2 is the CCT of another algorithm. The large $ratio_{imp}$ implies the performance of the second algorithm is better than the first algorithm, while $0 \leq ratio_{imp} < 1$. We also use the ratio of solution time $ratio_{time} = \frac{SolveTime_1}{SolveTime_2}$ as the time metric, where the $SolveTime_1$ is the solving time of the first algorithm while the $SolveTime_2$ represents the solving time of the second algorithm.

Hardware. In our experiment, we use Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz and NVIDIA GeForce RTX 3090 for problem-solving and simulation.



(a) AirtelTcpCubic (b) AirtelTcpDctcp (c) AirtelTcpLinuxReno (d) AttMplsTcpCubic

(e) AttMplsTcpDctcp (f) AttMplsTcpLinuxReno (g) AttMplsCubic with 2ms delay (h) AttMplsDctcp with 2ms delay

Fig. 4. The Total Completion Time of Coflows in Different Topology and Congestion Control Algorithms with Different Number of Coflows.

Simulation Methodology. We embed the scheduling algorithms computed by our scheme into NS3 simulator to get the evaluation results. In the simulations, we use several variables as test metrics. Besides the topology and the traffic mentioned above, we also introduce different congestion control algorithms to ensure the universality of the results. On the other hand, we compare the solution time of the dual problem with the solver [20] to analyze the time consumption of our algorithms.

B. Experiment Performance

We simulate our experiments on the most widely used packet-level simulator NS3, to further clarify the advantages of our design on different scale networks with various settings of coflow and congestion control algorithms mentioned above.

The Scheduling Results. We do many experiments on NS3 with different traffic, topology, and congestion control algorithms. We first put our attention on the performance in some typical network settings and traffic settings. Fig. 4 presents the CCT in *Airtel* and *AttMpls* with various coflow numbers and delays, in which Fig. 4(g) and 4(h) are results with longer latency, and we can find the performance of FSCO is better in many scenes, especially with fewer coflows and the results are correction to congestion control algorithms. And the FSCO overperforms the Sincronia up to 20% running time and the Stretch up to 30% running time from Fig. 4(a).

Compared to the different algorithms in Fig. 4, we find the results of FSCO are relatively more stable. From Fig. 4(c), Fig. 4(d) and Fig. 4(f), we find FSCO performs worse than Stercth when the number of coflow is too large or too small, and the result is very relevant to the topology.

We think our framework can improve the performance by alleviating the pressure of networks so that the performance is improved in many cases as shown in Fig. 5, where close to half of our results are better than others.

The Computing Time. The decision computing time is vital in the scheduling system because we should make the correct

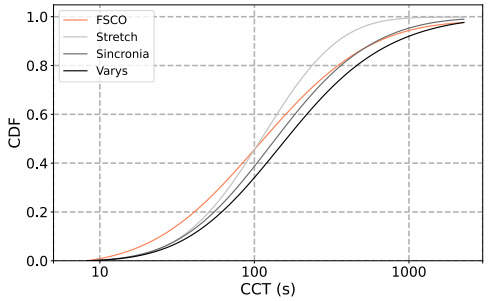


Fig. 5. The CCT Results with Different Number of Coflows, Coflow Volumes, Congestion Control Algorithms and Topology.

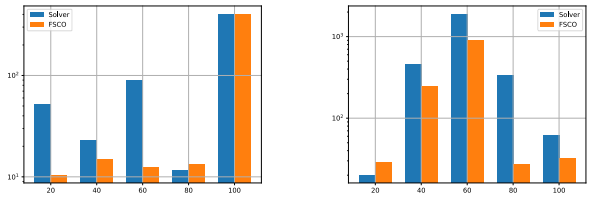
decisions in time to achieve better performance. We simulate the computing time motivated by the article [31]. As aforementioned, we analyze the computation time in many different scenes compared to the common solver Gurobi [20]. In each run of experiments, we adjust the parameters mentioned above and analyze the computation time to evaluate the performance.

We present partial results about the solving time of FSCO compared to the solver in Fig. 6. From the two figures, we could find FSCO performs better in many scenes. Moreover, the solving time and coflow number are not positively correlated.

We conduct the experiments and present the results as Fig. 7. From the above, we could observe that the computing time of the solver is larger than our algorithm in most scenes and the speed can be accelerated up to 12x while almost 50% of the cases achieve more than 2 times faster.

V. Conclusion

In this work, we proposed a novel framework in that we should schedule coflows at a fixed rate and formulate the problem. Then, we use dual decomposition and parallel solutions with analytic solutions to accelerate the speed of decision calculation. At the end of this paper, we show performance improvement and illustrate the advantages of



(a) AttMpls (b) Globenet

Fig. 6. The solving time of solver and FSCO in different Topology.

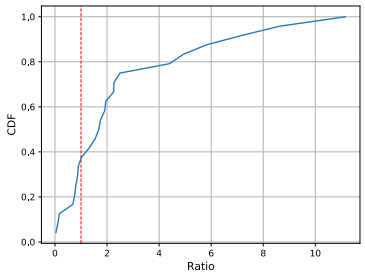


Fig. 7. The Improvement Ratio (SolverTime/FSCOTime) of Parallel Solution.

fixed-rate scheduling. The experiments show the importance of in-network conditions in scheduling. In future work, we will further qualitatively analyze the reasons for this situation.

ACKNOWLEDGMENT

We are extremely grateful for the anonymous reviewers for their wonderful feedback. Haoyu Liu, Feiyan Ding, Yao Wang, Qiao Xiang are supported in part by the National Key R&D Program of China 2022YFB2901502, NSFC Award 62172345, Open Research Projects of Zhejiang Lab 2022QA0AB05, MOE China 2021FNA02008, and NSF-Fujian-China 2022J01004.

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[2] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Presented as part of the 9th Symposium on Networked Systems Design and Implementation*, 2012, pp. 15–28.

[3] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 135–146.

[4] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, 2012, pp. 31–36.

[5] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014, pp. 443–454.

[6] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, 2018, pp. 16–29.

[7] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4, pp. 393–406, 2015.

[8] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, pp. 160–173.

[9] H. Jahanjou, E. Kantor, and R. Rajaraman, "Asymptotically optimal approximation algorithms for coflow scheduling," in *Proceedings of the 29th ACM SPAA*, 2017.

[10] M. Chowdhury, S. Khuller, M. Purohit, S. Yang, and J. You, "Near optimal coflow scheduling in networks," in *The 31st ACM Symposium on Parallelism in Algorithms and Architectures*, 2019, pp. 123–134.

[11] Z. Wang, H. Zhang, X. Shi, X. Yin, Y. Li, H. Geng, Q. Wu, and J. Liu, "Efficient scheduling of weighted coflows in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2003–2017, 2019.

[12] W. Li, X. Yuan, K. Li, H. Qi, X. Zhou, and R. Xu, "Endpoint-flexible coflow scheduling across geo-distributed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 10, pp. 2466–2481, 2020.

[13] S. Liu, L. Chen, and B. Li, "Siphon: Expediting {Inter-Datacenter} coflows in {Wide-Area} data analytics," in *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, 2018, pp. 507–518.

[14] B. Guo, Z. Zhang, Y. Yan, and H. Li, "Optimal job scheduling and bandwidth augmentation in hybrid data center networks," in *2022 IEEE GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM 2022)*, 2022, pp. 5686–5691.

[15] K. Lei, K. Li, J. Xing, B. Jin, and Y. Wang, "Distributed information-agnostic flow scheduling in data centers based on wait-time," in *2018 IEEE GLOBAL COMMUNICATIONS CONFERENCE (GLOBECOM)*, 2018.

[16] S. Zhang, S. Zhang, X. Zhang, Z. Qian, M. Xiao, J. Wu, J. Ge, and X. Wang, "Far-sighted multi-stage awasre coflow scheduling," in *2018 IEEE Global Communications Conference (GLOBECOM)*, 2019.

[17] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 424–432.

[18] L. Shi, Y. Liu, J. Zhang, and T. Robertazzi, "Coflow scheduling in data centers: routing and bandwidth allocation," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2661–2675, 2021.

[19] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. C. Lau, "Efficient online coflow routing and scheduling," in *Proceedings of the 17th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2016, pp. 161–170.

[20] L. Gurobi Optimization, "Gurobi optimizer reference manual," http://www.gurobi.com.

[21] "Network simulator 3 (ns-3)," https://www.nsnam.org/.

[22] Z. Xu, F. Y. Yan, R. Singh, J. T. Chiu, A. M. Rush, and M. Yu, "Teal: Learning-accelerated optimization of wan traffic engineering," 2023.

[23] F. P. Kelly, A. K. Maulloo, and D. K. H. Tan, "Rate control for communication networks: shadow prices, proportional fairness and stability," *Journal of the Operational Research Society*, vol. 49, no. 3, pp. 237–252, 1998.

[24] S. Knight, H. X. Nguyen, and Falkner, "The internet topology zoo," *IEEE Journal on Selected Areas in Communications*, 2011.

[25] S. Chiu and C. A. Papachristou, "A design for testability scheme with applications to data path synthesis," in *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 1991, pp. 271–277.

[26] L. Zhao, J. Wang, J. Liu, and N. Kato, "Optimal edge resource allocation in IoT-based smart cities," *IEEE Network*, 2019.

[27] F. Naeem, M. Tariq, and H. V. Poor, "SDN-enabled energy-efficient routing optimization framework for industrial internet of things," *IEEE Transactions on Industrial Informatics*, 2021.

[28] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data Center TCP (DCTCP)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: Association for Computing Machinery.

[29] V. Jacobson, "Modified TCP congestion avoidance algorithm," *Email to the end2end-interest mailing list*, 1990.

[30] S. Ha, I. Rhee, and L. Xu, "Cubic: A new TCP-friendly high-speed TCP Variant," *SIGOPS Oper. Syst. Rev.*, 2008.

[31] A. Agrawal, S. Boyd, and Narayanan, "Allocation of fungible resources via a fast, scalable price discovery method," *Mathematical Programming Computation*, 2022.