# Weighted Scheduling of Time-Sensitive Coflows

Olivier Brun, Rachid El-Azouzi, Quang-Trung Luu, Francesco De Pellegrini, Balakrishna Prabhu, Cédric Richier

# Weighted Scheduling of Time-Sensitive Coflows

Olivier Brun, Rachid El-Azouzi, Quang-Trung Luu, Francesco De Pellegrini,
Balakrishna J. Prabhu, and Cédric Richier

***Abstract*—Datacenter networks commonly facilitate the transmission of data in distributed computing frameworks through coflows, which are collections of parallel flows associated with a common task. Most of the existing research has concentrated on scheduling coflows to minimize the time required for their completion, i.e., to optimize the average dispatch rate of coflows in the network fabric. Nevertheless, modern applications often produce coflows that are specifically intended for online services and mission-crucial computational tasks, necessitating adherence to specific deadlines for their completion. In this paper, we introduce `WDCoflow`, a new algorithm to maximize the weighted number of coflows that complete before their deadline. By combining a dynamic programming algorithm along with parallel inequalities, our heuristic solution performs at once coflow admission control and coflow prioritization, imposing a $\sigma$-order on the set of coflows. With extensive simulation, we demonstrate the effectiveness of our algorithm in improving up to $3\times$ more coflows that meet their deadline in comparison the best SoA solution, namely `CS-MHA`. Furthermore, when weights are used to differentiate coflow classes, `WDCoflow` is able to improve the admission per class up to $4\times$, while increasing the average weighted coflow admission rate.**

***Index Terms*—Time-sensitive coflow scheduling, weighted coflow admission control, $\sigma$-order, deadline, datacenter networking, task scheduling, resource allocation.**

## I. Introduction

THE concept of coflow, firstly introduced in [1], forms the foundation of modern traffic engineering in datacenter networks. This abstraction of traffic was initially developed to capture the patterns of data exchange within distributed computing frameworks like MapReduce or Spark [2, 3]. These frameworks employ the *dataflow* computing model for processing large-scale data, which involves distributing intermediate computation stages across multiple nodes and transferring outputs to nodes responsible for the subsequent stages. During the transitions between computation stages, dataflows generate a set of network flows that traverse the datacenter fabric. These flows are abstracted as a *coflow*. A prominent example of a dataflow occurs in the *shuffle phase* of the Hadoop MapReduce framework [2]. However, it has been investigated in real traces [4] that coflow scheduling has a significant impact on the completion time of applications and the *shuffle phase* accounts for 33% of the running time in observed coflows. Hence, the reference objective function to measure acceleration at network layer is the makespan or Weighted Coflow Completion Time (WCCT). Minimizing the average WCCT or CCT is an appropriate objective for maximizing the number of computing jobs dispatched per hour in a datacenter fabric. Numerous works, such as [1, 5–10], have addressed the minimization of WCCT and proposed algorithmic solutions. Over the past decade, extensive research has illuminated the complexity of this problem. It has been proven to be *NP*-hard and inapproximable below a factor of 2 through reduction to the job scheduling problem on multiple correlated machines. Near-optimal methods have also been proposed in the literature, with performance bounds approximating a factor of 4 [6, 7, 11]. However, the context radically changes when dealing with time-critical jobs that impose strict deadlines on the coflow's data transfer phase.

In such scenario, the scheduling of coflows is commonly combined with admission control to minimize the number of deadline violations, i.e., the number of coflows that are unable to be completely transferred before their deadlines. This gives rise to the Coflow Deadline Satisfaction (CDS) problem, first introduced in [12]. Each coflow is assigned a specific deadline, and the objective is to perform joint *coflow admission control and scheduling* to maximize the number of admitted coflows that can meet their respective deadlines. This problem is also proven to be *NP*-hard, and it has been shown to be inapproximable within any constant factor of the optimal solution [12].

While the issue of time-sensitive coflows has been acknowledged in the early literature [13], most works on coflow scheduling have not focused on addressing this problem, with a few exceptions [12]. However, our performance analysis has revealed that even near-optimal algorithms designed for minimizing CCT may fail to meet coflow deadlines. In reality, the concept of time-sensitive coflows has become increasingly prevalent in modern distributed datacenters. It is not only computing frameworks that deal with time-sensitive tasks; modern web and mobile applications are built using microservice architectures, where user requests can trigger numerous services across multiple servers to retrieve data. The completion time of a batch of flows, i.e., the time instant at which the last bit of data arrives, determines the lag to the response time of these services, and significant delays can lead to a degraded user experience. In the realm of cloud computing and data centers, there is a rise in more time-sensitive applications, such as web search [14] and machine learning [15], which impose stricter deadline constraints. With this performance objective in mind, a coflow is only considered beneficial when all of its individual flows have completed their data transfer within the required deadline.

To reduce the number of coflows missing their deadlines,

Parts of this work have been presented at IFIP Networking 2022.

Olivier Brun and Balakrishna J. Prabhu are with LAAS-CNRS, University of Toulouse, CNRS, 31400 Toulouse, France (e-mails: {brun, bala}@laas.fr).

Rachid El-Azouzi, Francesco De Pellegrini, and Cédric Richier are with CERI/LIA, University of Avignon, 84029 Avignon, France (e-mails: {rachid.elazouzi, francesco.de-pellegrini, cedric.richier}@univ-avignon.fr).

Quang-Trung Luu is with the School of Electrical and Electronic Engineering, Hanoi University of Science and Technology, 100000 Hanoi, Vietnam (e-mail: trung.luuquang@hust.edu.vn).

i.e., the number of violations, existing solutions perform admission control. On the other hand, in solving the CDS problem one has to operate simultaneously both *coflow admission control and scheduling*. This allows to maximize the number of admitted flows while respecting their deadlines, i.e., the Coflow Acceptance Rate (CAR).

In this paper, we generalize the CDS problem to the case when coflows have a priority in the form of a nonnegative weight. The performance metric to maximize is the Weighted Coflow Acceptance Rate (WCAR). Since maximizing CAR is *NP*-hard [12], the same is true for maximizing WCAR and exact solution methods are of little practical use. In principle, it is possible to address the problem of time-sensitive coflows by formulating a suitable Mixed Integer Linear Program (MILP). However, when dealing with datacenters that handle tens of thousands of coflows [5], techniques relying on MILPs or their relaxations may not be practical or feasible. The computational complexity and scalability challenges associated with solving MILPs in such large-scale environments make them less viable for real-time implementation.

To achieve scalability in coflow scheduling for datacenters, the use of scalable algorithms is crucial. Many research works propose the concept of scheduling coflows using a priority order, known as the $\sigma$-order. Once the $\sigma$-order is determined, a work-conserving transmission policy can be adopted. The focus on $\sigma$-order schedulers is driven by their implementation advantages. Specifically, in terms of rate control, any work-conserving preemptive dynamic rate allocation can be used as long as it is compatible with the assigned coflow priorities. It has been shown that the maximum performance loss within such rate allocation policies is bounded by a factor of 2 [7]. For example, using fixed coflow priorities under DiffServ satisfies the definition of a $\sigma$-order scheduler. Additionally, commercial switches often have built-in priority queues and support per-flow tagging, which can be utilized to prioritize active coflows without requiring per-flow rate control. This allows for a greedy rate allocation that aligns with the desired $\sigma$-order. The exact mapping between a coflow's $\sigma$-order and the switch's priority queuing mechanism, as well as the limitations imposed by legacy hardware, are interesting subjects but beyond the scope of this paper.

*Contributions.* In this paper, we introduce lightweight algorithms for coflow scheduling with deadlines. The proposed algorithms surpass existing solutions in the literature and do not rely on solving linear programs. The proposed approach consists of an offline admission control policy combined with a scheduler belonging to the class of $\sigma$-order coflow schedulers. The output of the algorithm is a priority order restricted to the set of admitted coflows. Our heuristic solutions, named `WDCoflow`, leverage techniques such as dynamic programming [16] (known for optimality in the single link case) and parallel inequalities for completion times [17]. By employing these techniques, our heuristics effectively capture the inter-coflow impacts and determine the coflows that should be admitted. The algorithms are further extended to handle joint admission control and scheduling in online scenarios where coflows are generated at runtime with unknown release times. Through extensive numerical experiments on various scenarios, including both synthetic and real traces obtained from the Facebook data [13], we demonstrate that our algorithm consistently outperforms existing solutions in the literature. The simulations encompass offline and online settings, and the `WDCoflow` algorithm consistently achieves near-optimal Weighted Coflow Arrival Rate (WCAR) for smaller fabrics. Moreover, it outperforms state-of-the-art solutions across all evaluated workloads, achieving significant improvements of up to 4 times in certain cases, especially for overloaded fabrics.

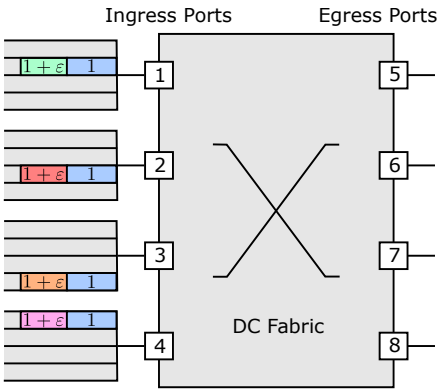



Fig. 1. Example of a Big-Switch fabric having 4 ingress/egress ports connecting to 4 machines. Flows in ingress ports are organized by destinations and are color-coded by coflows. The example has 5 coflows. Coflow $k_1$ (blue) has 4 flows, with each ingress port sending 1 units of data to one egress port: its deadline is 1; coflows $k_2$ (green), $k_3$ (red), $k_4$ (orange) and $k_5$ (purple) have a single flow, each sending $(1+\varepsilon)$ unit of data. The deadline of these coflows is 2.

The remaining sections of the paper are structured as follows. Sec. II presents an overview of the general problem addressed in the paper, including the description of coflow ordering models. Sec. III introduces the proposed algorithms, detailing their design and methodology. Numerical results are then presented in Sec. IV. Sec. V discusses the related work in the field of scheduling time-sensitive coflows. Finally, Sec. VI presents concluding remarks and outlines potential directions for future research.

## II. Problem Statement

In this section, we formally define the deadline scheduling problems for coflows with weights as an MILP.

The datacenter network is modeled as a non-blocking switch, as in Fig. 1. This is usually referred to as the *Big-Switch* model, used for the first time for datacenter coflow scheduling in [13]. In that model, two disjoint sets of ports, namely the *ingress* ports and the *egress* ports, represent all the ports of the Top of Rack (ToR) switches connecting machines hosted in racks to the network fabric. The mathematical model for a switch is set of ports (or links) $\mathcal{L} = \{1, 2, \ldots, 2M\}$ where $\ell \in \{1, \ldots, M\}$ are ingress ports and $\ell \in \{M+1, \ldots, 2M\}$ are egress ports. We assume that each port $\ell \in \mathcal{L}$ has a maximum rate of $B_\ell$.

A coflow is a set of flows, where each flow is a volume of data to be transferred between an ingress port and an egress port. In the example of Fig. 1, at each ingress port, flows are organized in virtual output queues indicating the output port.

TABLE I
MAIN NOTATIONS.

| *Symbol* | *Description* |
|---|---|
| $M$ | number of machines |
| $\mathcal{L}$ | set of fabric ports, $\|\mathcal{L}\| = L$ |
| $B_\ell$ | available bandwidth of port $\ell \in \mathcal{L}$ |
| $\mathcal{C}$ | set of coflows, $\|\mathcal{C}\| = N$ |
| $v_k$ | volume of coflow $k$ |
| $w_k$ | weight of coflow $k$ |
| $T_k$ | deadline of coflow $k$ |
| $z_k$ | binary indicator for coflow $k$ |
| $c_k$ | completion time of coflow $k$ |
| $c_{\ell,k}$ | completion time of coflow $k$ on port $\ell$ |
| $p_{\ell,k}$ | processing time of coflow $k$ on port $\ell$ |
| $\hat{v}_{\ell,k}$ | total volume sent by coflow $k$ on port $\ell$ |
| $\mathcal{F}_k$ | set of flows of coflow $k$ |
| $\mathcal{F}_{\ell,k}$ | set of flows of coflow $k$ that use port $\ell$ |
| $v_{k,j}$ | volume of flow $j \in \mathcal{F}_k$ |
| $\mathcal{T}$ | time horizon for coflow scheduling |
| $r_{k,j}(t)$ | rate allocated to flow $j \in \mathcal{F}_k$ at time $t$ |
| $\sigma$ | scheduling order of coflows, $\sigma = \{\sigma_1, \cdots, \sigma_{N-1}, \sigma_N\}$ |
| $\alpha$ | deadline threshold (coflow $k$ has max deadline of $\alpha \text{CCT}_k^0$) |
| $\mathbb{1}_{k,i}$ | binary indicating whether coflow $k$ is of class or not |
| $\delta_{k',k}$ | binary indicating coflow $k'$ has a higher priority that $k$ |
| $P^{(j)}(w)$ | minimum total processing time for any feasible subset of coflows $\{1, \cdots, j\}$ that has total weight $w$ |

For clarity, the scheduling problem is formulated in the offline setting. Hence, all the coflows are available at time 0, i.e., when the scheduling decision is taken. Later on, the algorithms will be adapted for the online setting where scheduling decision are taken over the course of a given time horizon, and coflows arrive over time. In turn, the characteristics of the future coflows are unknown.

Consider a batch of $N$ coflows $\mathcal{C} = \{1, 2, ..., N\}$. We denote by $w_k$ the weight, i.e., the importance, of coflow $k$, so that the acceptance rate can be optimized with regard to its weight. Each coflow $k$ is subject to a completion deadline $T_k$. The set of flows of coflow $k$ is denoted by $\mathcal{F}_k$. A flow is defined by its volume and the pair of ports that it uses. Let $v_{k,j}$ be the volume of flow $j$ of coflow $k$, and let $\mathcal{F}_{\ell,k}$ be the set of flows in $\mathcal{F}_k$ that uses port $\ell \in \mathcal{L}$ either as ingress port or as egress port. Table I summarizes the main notations used throughout the paper.

### A. MILP Formulation

Let $z_k \in \{0, 1\}$ be an indicator of whether coflow $k$ finishes before its deadline $T_k$, and let $r_{k,j}(t) \in \mathbb{R}_+$ be the rate allocated to flow $j \in \mathcal{F}_k$ at time $t$. The target coflow scheduling problem prescribes to identify the set of coflows to be scheduled in order to maximize the corresponding cumulative weight. We will refer to this scheduling problem as *Weighted Coflow Acceptance Rate* (WCAR) problem, which is formulated as

$$\max_{r} \sum_{k \in \mathcal{C}} w_k z_k \qquad \text{(WCAR)}$$

$$\text{s.t.} \sum_{k \in \mathcal{C}} \sum_{j \in \mathcal{F}_{\ell,k}} r_{k,j}(t) \leq B_\ell, \quad \forall \ell \in \mathcal{L}, \forall t \in \mathcal{T}, \qquad (1)$$

$$\int_0^{T_k} r_{k,j}(t)\, \mathrm{d}t \geq v_{k,j} z_k, \quad \forall j \in \mathcal{F}_k, \forall k \in \mathcal{C}, \qquad (2)$$

where $\mathcal{T}$ is the time interval over which scheduling is performed and can be set to $[0, \max_k T_k]$. Constraint (1) ensures that the total rate allocated on port $\ell$ at every time instant in $\mathcal{T}$ does not exceed its capacity $B_\ell$. Constraint (2) ensures that all flows of every *accepted* coflow are completely processed before the deadline of that coflow. Note that solving the WCAR problem requires to define optimal rate allocations $r_{k,j}(t)$, $\forall j \in \mathcal{F}_k$, $\forall k \in \mathcal{C}$, and $\forall t \in \mathcal{T}$.

Assume without loss of generality that coflows are numbered in the Earliest Due Date (EDD) order. It is then clear that, given a feasible schedule, only coflows $\{k+1, \ldots, N\}$ are present in the system in time interval $[T_k, T_{k+1}]$. Assuming that the rate allocations $r_{k,j}(t)$ are constant in the time intervals $[0, T_1], [T_1, T_2], \ldots, [T_{N-1}, T_N]$, we obtain a MILP formulation of Problem (WCAR), which generalizes the formulation proposed in [12] for unweighted coflows.

When all the coflows have the same weight, it was shown in [12] that Problem (WCAR) is *NP*-hard.

**Lemma 1** (Proposition 1 in [12])**.** *When the weights are equal, there exists a polynomial time reduction of Problem* (WCAR) *to the problem of minimizing the number of late jobs in a concurrent open shop [18]. Hence, Problem* (WCAR) *is NP-hard.*

For completeness, we restate the result for unequal weights as well although it is direct consequence of the problem with equal weights.

### B. Upper Bound ILP for WCAR

Problem (WCAR) solves for the rate allocation and determines which coflows satisfy their deadline. It thus allows rate allocations that share ports' capacity possibly among several coflows. An alternative approach is to determine an ordering $\sigma$ of coflows first and then assign full port rates to coflows that have higher priority according to $\sigma$. Hence, flow $j \in \mathcal{F}_{\sigma_k}$ is blocked if and only if either its ingress or egress port is busy serving a flow $j' \in \mathcal{F}_{\sigma_{k'}}$ for some $k' < k$ in the $\sigma$-order. The order thus implies a strict priority on the ports utilization. A flow scheduling that follows this priority rule is called *$\sigma$-order-preserving*.

The coflow ordering approach was first taken in [7] for the minimization of Coflow Completion Times (CCT). It was then applied to deadline scheduling but without weights in [19]. The advantage of this approach is that it does not require rate computations. Once an order is determined, the rates can be deduced directly from there. On the other hand, it has the disadvantage of being an upper bound for deadline scheduling as shown in [19].

Here, we give a short summary of those arguments. The problem of finding the optimal $\sigma$-order is in fact an ILP. To see this, we will need to define a couple of terms. The processing time in isolation of coflow $k$ at port $\ell$ is defined as $p_{\ell,k} = \hat{v}_{\ell,k}/B_\ell$, where $\hat{v}_{\ell,k} = \sum_{j \in \mathcal{F}_{\ell,k}} v_{k,j}$ is the total volume sent by coflow $k$ on port $\ell$. That is, $p_{\ell,k}$ is the time to transfer all the data of coflow $k$ on port $\ell$ in the absence of other coflows. Further, for $k' \neq k$, define the binary variable $\delta_{k',k}$ which is 1 if coflow $k'$ has a higher priority than $k$.

and 0 otherwise. An ordering $\sigma$ can then be derived from the variables $\{\delta_{k,k'}\}_{k,k' \in \mathcal{C}}$ by subjecting them to the standard disjunctive constraints

$$\delta_{k,k'} + \delta_{k',k} = 1, \quad \forall k, k' \in \mathcal{C}, \tag{3}$$

$$\delta_{k,k'} + \delta_{k',k"} + \delta_{k",k} \leq 2, \quad \forall k, k', k" \in \mathcal{C}. \tag{4}$$

The only step remaining now is to express the constraint that accepted coflows should have a CCT smaller than their deadline in a linear form. Unfortunately, there are no known linear inequalities to express the region of schedulability of coflows in a switch. The difficult arises from the blocking nature of the switch: a flow may be blocked because either its ingress or egress port is being used by another flows. Therefore, transmission times on a port depend on what happens on the other ports.

Nevertheless, the following lower bound on the completion time of coflow $k$ on port $\ell$, $c_{\ell,k}$ can be obtained by assuming the ports are independent,

$$c_{\ell,k} \geq \sum_{k' \neq k} p_{\ell,k'} \delta_{k',k} z_{k'} + p_{\ell,k} z_k, \quad \forall \ell \in \mathcal{L}, k \in \mathcal{C}. \tag{5}$$

Here, only accepted coflows, i.e. those for which $z_k = 1$, are accounted for in the bound (5) (hence the term $p_{\ell,k} z_k$). The lower bound on $c_{\ell,k}$ is then just the time it takes to transmit all the coflows with priority higher than $k$ on port $\ell$. The product $\delta_{k',k} z_{k'}$ can easily be linearized by introducing binary variables $y_{k',k}$ satisfying the constraints

$$y_{k',k} \leq z_{k'}; \quad y_{k',k} \leq \delta_{k',k}; \quad y_{k',k} \geq z_{k'} + \delta_{k',k} - 1. \tag{6}$$

The lower bound (5) can now be rewritten as the following linear inequality:

$$c_{\ell,k} \geq \sum_{k' \neq k} p_{\ell,k'} y_{k',k} + p_{\ell,k} z_k, \quad \forall \ell \in \mathcal{L}, k \in \mathcal{C}. \tag{7}$$

Since the CCT of coflow $k$ is given by $c_k = \max_{\ell \in \mathcal{L}} c_{\ell,k}$, the constraint that the CCT of this coflow is smaller that its deadline can be expressed as

$$c_{\ell,k} \leq T_k z_k, \quad \forall \ell \in \mathcal{L}, k \in \mathcal{C}. \tag{8}$$

Finally, the optimal $\sigma$-order coflow scheduling problem can be formulated as the following ILP,

$$\max \sum_{k \in \mathcal{C}} w_k z_k, \quad \text{s.t. } (3, 4, 6, 7, 8). \tag{$\sigma$-WCAR}$$

Recall that solutions of Problem ($\sigma$-WCAR) provide an upper bound on the number of accepted coflows to that of Problem (WCAR).

### C. Motivating Example

We now illustrate, with an example, some of the shortcomings of CS-MHA [20], an algorithm for maximizing the acceptance ratio of coflows *without* weights (i.e., maximizing the CAR). CS-MHA introduces a novel approach to solve the scheduling problem by employing a static coflow prioritization. This prioritization is utilized to approximate the solution for the coflow scheduling problem that maximizes the CAR. First, CS-MHA computes the scheduling order and the set of admitted coflows at each port using the Moore-Hodgson algorithm [21]. Since different ports may have different sets of admitted coflows, a coflow is admitted only if it is admitted by all ports simultaneously. Then, for the coflows that are rejected, a second round is conducted to reassess if some of them can actually meet their deadlines. In this case, CS-MHA selects the coflow with the minimum bandwidth requirement at the bottleneck port. This choice is based on the reasoning that coflows with lower bandwidth requirements are more likely to catch up with their deadlines.

Fig. 1 shows a simple example to illustrate the shortcomings of CS-MHA. This will be used as a running example throughout the paper. The example consists of five coflows: $k_1$ with four flows, and $k_2, k_3, k_4$, and $k_5$ with one flow each. To facilitate the presentation, the flows are organized in virtual output queues at the ingress ports, where the virtual queue index represents the flow output port modulo the number of machines. The numbers on the flows' representations indicate their *normalized* volumes. All fabric ports have the same *normalized* bandwidth of 1.

In the first iteration, CS-MHA uses the Moore-Hodgson algorithm to compute the scheduling order at each port, as mentioned earlier. This algorithm is based on the EDD rule with objective to minimize the number of missed deadlines on a single machine (or port in the coflow context). In this example, since coflow $k_1$ uses all ports and has the smallest deadline ($T_1 = 1$), it will be scheduled first at each port. Consequently, all other coflows are rejected because they cannot meet their deadlines when scheduled after $k_1$. This results in a coflow scheduling with a CAR of $\frac{1}{5}$. However, an optimal scheduling solution would be $k_2, k_3, k_4, k_5, k_1$ or any combination where coflow $k_1$ is scheduled last. This scheduling achieves a CAR of $\frac{4}{5}$.

To further illustrate the limitations of the CS-MHA, consider now the case where there are $M$ machines, coflow $k_1$ utilizes all ports, and coflows $k_2, \ldots, k_M$ have one flow each. The other parameters remain unchanged. In this setting, we shall demonstrate that the CAR obtained using CS-MHA and DCoflow[1] are respectively $\frac{1}{M}$ and $\frac{M-1}{M}$. With this setting, when the $M$ increases, CS-MHA yields a CAR close to zero, while with DCoflow, it is close to one.

The key observation in this example is that how CS-MHA neglects the impact that a coflow may have on other coflows across multiple ports. Specifically, a coflow that leads to the missing of multiple deadlines should have a lower priority, even if its own deadline is the earliest. Neglecting this consideration leads to a misjudgment in the coflow ordering, resulting in a final schedule that significantly degrades the CAR compared to an optimal solution. Building upon this observation, in what follows, we propose a new class of $\sigma$-order schedulers called WDCoflow to address the joint coflow admission control and scheduling problem.

[1] DCoflow [19] is the variant of WDCoflow that deals with unweighted coflows. Detailed differences between DCoflow and WDCoflow shall be given in Sec. III-B.

## III. $\sigma$-Order Scheduling with WDCoflow

In this section, we present WDCoflow, an algorithm to solve the problem of joint coflow admission control and scheduling. Given a list of $N$ coflows and their respective weights, it provides a permutation $\sigma = (\sigma_1, \sigma_2, .., \sigma_N)$ of these coflows, with the aim of maximizing the coflow acceptance rate. A key ingredient of our algorithm is a simple rule for deciding which coflow to reject when there is no feasible schedule. This rule is based on a necessary schedulability condition which is established in Sec. III-A. We describe our algorithm for solving offline instances in Sec. III-B and Sec. III-C, and analyze their complexity in Sec. III-D. Finally, the online implementation of WDCoflow is described in Sec. III-E.

### A. A Necessary Schedulability Condition

Given a subset $\mathcal{S} \subseteq \mathcal{C}$ of *admitted* coflows, a feasible schedule of $\mathcal{S}$ is a processing order of coflows such that $c_k \leq T_k$, $\forall k \in \mathcal{S}$, where $c_k$ represents the completion time of coflow $k$. We establish below a necessary condition for such a schedule to exist and show how it can be used to decide which coflows should be admitted.

Given $\mathcal{S} \subseteq \mathcal{C}$ and a coflow $k \in \mathcal{S}$, let $\mathcal{S}_k^-$ be the set of coflows in $\mathcal{S}$ which are scheduled before $k$ (i.e., coflows of higher priority). By assuming that the transmission of coflow $k$ on port $\ell$ can start as soon as all flows of all coflows $j \in \mathcal{S}_k^-$ have been transmitted on port $\ell$, we can obtain a lower bound on the completion time of coflow $k$

$$c_k \geq p_{\ell,k} + \sum_{j \in \mathcal{S}_k^-} p_{\ell,j}, \quad \forall \ell \in \mathcal{L}. \tag{9}$$

Multiplying (9) on both sides by $p_{\ell,k}$ and summing over all coflows $k \in \mathcal{S}$ yields

$$\begin{aligned}\sum_{k\in\mathcal{S}} p_{\ell,k}c_k &\geq \sum_{k\in\mathcal{S}} (p_{\ell,k})^2 + \sum_{k\in\mathcal{S}} p_{\ell,k} \sum_{j\in\mathcal{S}_k^-} p_{\ell,j} \\ &= \frac{1}{2}\sum_{k\in\mathcal{S}} (p_{\ell,k})^2 + \frac{1}{2}\left[\sum_{k\in\mathcal{S}} (p_{\ell,k})^2 + 2\sum_{k\in\mathcal{S}} p_{\ell,k} \sum_{j\in\mathcal{S}_k^-} p_{\ell,j}\right] \\ &= f_\ell(\mathcal{S}), \end{aligned} \tag{10}$$

where

$$f_\ell(\mathcal{S}) = \frac{1}{2}\sum_{k\in\mathcal{S}} (p_{\ell,k})^2 + \frac{1}{2}\left(\sum_{k\in\mathcal{S}} p_{\ell,k}\right)^2. \tag{11}$$

From (10), we can conclude that the CCTs $\{c_k\}_{k\in\mathcal{S}}$ necessarily satisfiy the condition $\sum_{k\in\mathcal{S}} p_{\ell,k}c_k \geq f_\ell(\mathcal{S})$ for any port $\ell \in \mathcal{L}$ and for any subset $\mathcal{S} \subseteq \mathcal{C}$ of *admitted* coflows. These conditions are referred to as the *parallel inequalities*, and they serve as valid inequalities for the concurrent open shop problem [22]. It is important to note that these inequalities are independent of the coflow ordering and solely depend on the set of admitted coflows.

We now use the parallel inequalities to determine the coflows that should be rejected, if any. More precisely, given a set $\mathcal{S}$ of coflows, we define for each port $\ell \in \mathcal{L}$ the quantity

$$I_\ell(\mathcal{S}) \triangleq \sum_{k\in\mathcal{S}} p_{\ell,k}T_k - f_\ell(\mathcal{S}) \geq 0, \tag{12}$$

and use it as a measure of the schedulability of the set $\mathcal{S}$ of flows. Indeed, if $I_\ell(\mathcal{S}) < 0$, it follows from (10) and (12) that $\sum_{k\in\mathcal{S}} p_{\ell,k}T_k < f_\ell(\mathcal{S}) \leq \sum_{k\in\mathcal{S}} p_{\ell,k}c_k$, which implies that at least one coflow in $\mathcal{S}$ is late, whatever the order in which these coflows are scheduled. In other words, $I_\ell(\mathcal{S}) \geq 0$ for all $\ell \in \mathcal{L}$ is a necessary condition for a feasible schedule of $\mathcal{S}$ to exist.

The set $\mathcal{L}^\star = \{\ell \in \mathcal{L} \ : I_\ell(\mathcal{S}) < 0\}$ then represents the set of ports on which at least one coflow is late, whatever the order in which the coflows are (locally) processed. Hence, if $\mathcal{L}^\star \neq \varnothing$, at least one coflow $k^\star$ using one or more ports in $\mathcal{L}^\star$ should be removed from $\mathcal{S}$ so that the remaining coflows can meet their deadlines.

If there is only one port $\ell$ in $\mathcal{L}^\star$, a natural choice is to choose $k^\star$ so as to maximize the quantity $I_\ell(\mathcal{S} \setminus \{k^\star\})$ in the hope that it becomes positive. Observe that for any $j \in \mathcal{S}$,

$$\begin{aligned} f_\ell(\mathcal{S}) &= \frac{1}{2}\left[p_{\ell,j}^2 + \sum_{k\in\mathcal{S}\setminus\{j\}} p_{\ell,k}^2 + \left(p_{\ell j} + \sum_{k\in\mathcal{S}\setminus\{j\}} p_{\ell k}\right)^2\right] \\ &= f_\ell(\mathcal{S}\setminus\{j\}) + p_{\ell,j}\sum_{k\in\mathcal{S}} p_{\ell,k}, \end{aligned} \tag{13}$$

from which it follows that $I_\ell(\mathcal{S}\setminus\{j\}) = I_\ell(\mathcal{S}) + \Psi_{\ell,j}$, where

$$\Psi_{\ell,j} = p_{\ell,j}\left(\sum_{k\in\mathcal{S}} p_{\ell,k} - T_j\right). \tag{14}$$

Hence, maximizing $I_\ell(\mathcal{S}\setminus\{j\})$ is equivalent to maximizing $\Psi_{\ell,j}$. As coflows with small weights should be rejected in priority, we choose $k^\star \in \operatorname{argmax}_j \frac{1}{w_j}\Psi_{\ell,j}$. In words, this rule dictates to reject a coflow $k^\star$ with a small weight $w_{k^\star}$ and which has either a large processing time $p_{\ell,k^\star}$ or a large deadline violation $\sum_{k\in\mathcal{S}} p_{\ell,k} - T_{k^\star}$ when scheduled as the last one, or both.

When there is more than one port in $\mathcal{L}^\star$, a straightforward extension of the previous rule is to choose a coflow $k^\star$ with a small weight so as to maximize $\sum_{\ell\in\mathcal{L}^\star} I_\ell(\mathcal{S}\setminus\{k^\star\})$. In this case, we choose the coflow $k^\star$ with the largest value of the index $\frac{1}{w_k^\star}\sum_{\ell\in\mathcal{L}^\star}\Psi_{\ell,k^\star}$. An obvious advantage of this simple rule is that it allows to account for the impact of the removal of coflow $k^\star$ on all ports $\ell \in \mathcal{L}^\star$ used by this coflow.

### B. Offline Algorithm

The proposed offline algorithm, namely WDCoflow, is inspired from the DCoflow algorithm proposed in [19], which was devised for the unweighted setting. It takes as input a set $\mathcal{C} = \{1, 2, \ldots, N\}$ of coflows, which are all available at time 0, and computes as output the scheduling order of accepted coflows. The pseudocode of WDCoflow is described in Algorithm 1. We have omitted the RemoveLateCoflows subroutine in the pseudocode since it is the same as in [19].

**Algorithm 1:** `WDCoflow`

Set $\mathcal{S} = \{1, 2, \ldots, N\}$ and $n = N$; ▷ initialization
**while** $\mathcal{S} \neq \varnothing$ **do**
  Compute $t_\ell = \sum_{k \in \mathcal{S}} p_{\ell,k}\ \forall\, \ell \in \mathcal{L}$ and $\ell_b = \arg\max_{\ell \in \mathcal{L}} t_\ell$;
  Set $\mathcal{S}_b = \{k \in \mathcal{S} \,:\, p_{\ell_b,k} > 0\}$; ▷ coflows in $\mathcal{S}$ using $\ell_b$
  Set $k' = \arg\max_{k \in \mathcal{S}_b} T_k$ ▷ max-deadline coflow on $\ell_b$
  **if** $t_{\ell_b} \leq T_{k'}$ **then**
    Set $\sigma_n = k'$ and $\sigma^\star_n = 0$ ▷ admit coflow $k'$
  **else**
    Set $k^\star = \texttt{RejectCoflow}(\mathcal{S}_b)$; ▷ select a coflow to reject
    Set $\sigma_n = k^\star$ and $\sigma^\star_n = k^\star$; ▷ pre-reject coflow $k^\star$;
  $\mathcal{S} = \mathcal{S} \backslash \{\sigma_n\}$; ▷ remove coflow $\sigma_n$ from $\mathcal{S}$
  $n = n - 1$; ▷ update the iteration index
$\sigma = \texttt{RemoveLateCoflows}\,(\sigma, \sigma^\star)$;
**return** $\sigma$; ▷ final scheduling order
**Function** `RejectCoflow`$(\mathcal{S}_b)$**:**
  Set $\mathcal{R} = \texttt{Filter}(\mathcal{S}_b)$ ▷ candidate coflows for rejection
  Set $k^\star = \arg\max_{j \in \mathcal{R}} \frac{1}{w_j} \sum_{\ell \in \mathcal{L}^\star} \Psi_{\ell,j}$ ▷ coflow to reject
  **return** $k^\star$

In what follows, we highlight the main steps of `WDCoflow` and its main differences to `DCoflow`.

There are two main phases in `WDCoflow`. In the first phase, the algorithm works in iterations and in each iteration, it either accepts or rejects one coflow. The selected coflow is then removed from the current set of coflows $\mathcal{S}$, which is initialized to $\mathcal{C}$. In each iteration, `WDCoflow` updates two vectors $\sigma$ and $\sigma^\star$ to keep track of candidate coflows: $\sigma_{N-n+1}$ is set to the identity of the coflow selected in iteration $n = 1, 2, \ldots, N$, and $\sigma^\star_{N-n+1}$ is set to the identity of the coflow rejected in that iteration, if any (otherwise, we set $\sigma^\star_{N-n+1} = 0$ and accept coflow $\sigma_{N-n+1}$).

In iteration $n$, `WDCoflow` sweeps through the set of coflows $\mathcal{S}$ to compute the total completion time $t_\ell = \sum_{k \in \mathcal{S}} p_{\ell,k}$ of coflows in each port $\ell$. It then determines the bottleneck port $\ell_b$, i.e., the port $\ell$ with the largest completion time $t_\ell$. Let $k'$ be the coflow using port $\ell_b$ with the largest deadline. If $t_{\ell_b} \leq T_{k'}$, then coflow $k'$ can be scheduled as the last one on port $\ell_b$ and still satisfies its deadline. This coflow is therefore accepted by the algorithm and we set $\sigma_{N-n+1} = k'$ and $\sigma^\star_{N-n+1} = 0$. If on the contrary $t_{\ell_b} > T_{k'}$, this implies that at least one coflow among those using the bottleneck port will be late and therefore one of these coflows has to be rejected. `WDCoflow-DP`, a variant of `WDCoflow`, then uses a filtering algorithm described in Section III-C to compute a set $\mathcal{R} \subseteq \mathcal{S}_b$ of candidate coflows for rejection among those using the bottleneck port. For `WDCoflow`, this filter is desactivated, so that $\mathcal{R} = \mathcal{S}_b$. The coflow $k^\star \in \mathcal{R}$ to be rejected is then chosen as $k^\star = \arg\max_{k \in \mathcal{R}} \frac{1}{w_k} \sum_{\ell \in \mathcal{L}^\star} \Psi_{\ell,k}$, as explained in Section III-A. `WDCoflow` then sets $\sigma_{N-n+1} = \sigma^\star_{N-n+1} = k^\star$.

The second phase of `WDCoflow` is a post-processing phase intended at accepting unduly rejected coflows. Indeed, some coflows in $\sigma^\star$ could have been accepted if certain coflows that were rejected later would have been rejected earlier. To handle such cases, we use the function `RemoveLateCoflows` proposed in [19]. At the end of the second phase of `WDCoflow`, the estimated CCT of all coflows appearing in the order $\sigma$ is at most their deadline.

TABLE II
EXECUTION OF `WDCoflow` ON THE EXAMPLE OF FIG. 1.

| Unscheduled coflows (set $\mathcal{S}$) | $\ell_b$ | $\{\overline{\Psi}_1, \overline{\Psi}_2, \overline{\Psi}_3, \overline{\Psi}_4, \overline{\Psi}_5\}$ |
|---|---|---|
| $\mathcal{S} = \{\boldsymbol{k_1}, \boldsymbol{k_2}, k_3, k_4, k_5\}$ | 1 | $\{-4\,(1+\varepsilon)\,, -\varepsilon, \cdot, \cdot, \cdot\}$ |
| $\mathcal{S} = \{\boldsymbol{k_2}, k_3, k_4, k_5\}$ | 1 | $\{\cdot, 0, \cdot, \cdot, \cdot\}$ |
| $\mathcal{S} = \{\boldsymbol{k_3}, k_4, k_5\}$ | 2 | $\{\cdot, \cdot, 0, \cdot, \cdot\}$ |
| $\mathcal{S} = \{\boldsymbol{k_4}, k_5\}$ | 3 | $\{\cdot, \cdot, \cdot, 0, \cdot\}$ |
| $\mathcal{S} = \{\boldsymbol{k_5}\}$ | 4 | $\{\cdot, \cdot, \cdot, \cdot, 0\}$ |

We revisit the example depicted in Fig. 1 to demonstrate the difference between `WDCoflow` and `CS-MHA`. The execution of `WDCoflow` on this example is presented in Table II. In the initial step, `WDCoflow` selects bottleneck ingress port 1, which is used by coflows $k_1$ and $k_2$. It then calculates $\overline{\Psi}_k = \sum_{\ell:\Psi_{\ell,k}<0} \Psi_{\ell,k}$ for both coflows and chooses the coflow that yields the largest $\bar{\Psi}_k$ (in this case, $k_1$) to be scheduled last. Since the remaining unscheduled coflows do not share any ports in the fabric, the specific ordering of these coflows does not impact the average CAR. Given the final scheduling order, `WDCoflow` yields a CAR of $\frac{4}{5}$, which is the optimal result, and is better than the average CAR of $\frac{1}{5}$ yielded by `CS-MHA`. In a general setting with $M$ machines, the CAR obtained using `CS-MHA` and `WDCoflow` are respectively $\frac{1}{M}$ and $\frac{M-1}{M}$.

In the following, we consider three variants of `WDCoflow`, namely `DCoflow` for unweighted coflows and `WDCoflow` and `WDCoflow-DP` for weighted coflows. The first variant, `DCoflow` corresponds to Algorithm 1 in [19] and therefore assumes that all coflow weights are equal. The second variant, `WDCoflow`, is similar to `DCoflow` but uses coflow weights in the coflow rejection rule, as described in Section III-A. Finally, the third variant, `WDCoflow-DP`, works as `WDCoflow` but uses a Dynamic Programming (DP) algorithm which plays the role of a filter that restricts the choice of coflows that can be rejected. `WDCoflow-DP` is described in Section III-C below.

### C. Filtering Algorithm in `WDCoflow-DP`

The coflow rejection rule discussed in Section III-A is not necessarily optimal, even in the simple case of a single port[2]. It turns out that, in this simple case, finding a maximum-weight feasible set of coflows is equivalent to the well-known scheduling problem of minimizing the weighted number of late jobs on a single machine, a problem usually referred to as[3] $1\|\sum w_j U_j$.

As it includes the ordinary knapsack problem as a special case, this problem is *NP*-hard. Nevertheless, it can be solved by a dynamic programming algorithm within a pseudo-polynomial time bound of $\mathcal{O}(N\,W)$, where $W = \sum_j w_j$, as we now explain [16].

Without loss of generality, we assume that coflows are numbered in the EDD order, i.e., $T_1 \leq T_2 \leq \ldots, T_N$. As we assume that there is a single port $\ell$, we denote the processing time of coflow $k$ by $p_k$ instead of $p_{\ell,k}$. Let $P^{(j)}(w)$ denote the minimum total processing time for any feasible subset of

[2]If there is only one input port and one output port, the problem reduces to scheduling coflows on the minimum-capacity port.

[3]This follows the notable triple $\alpha|\beta|\gamma$ notation proposed in [23], where $\alpha$ is the number of machines, $\beta$ is an optional list of job characteristics (not present in this case), and $\gamma$ is the objective function.

coflows $1, \ldots, j$ that has total weight $w$. Initially, $P^{(0)}(0) = 0$ and $P^{(0)}(w) = +\infty$ for all $w \in \{1, 2, \ldots, W\}$. In the subsequent $n$ iterations $j = 1, 2, \ldots, n$, the variables $P^{(j)}(w)$ are computed as follows

$$P^{(j)}(w) = \begin{cases} \min\left\{P^{(j-1)}(w),\ P^{(j-1)}(w - w_j) + p_j\right\}, \\ \qquad \text{if } P^{(j-1)}(w - w_j) + p_j \leq T_j, \\ P^{(j-1)}(w),\ \text{otherwise.} \end{cases} \tag{15}$$

At the end of the algorithm, the maximum weight of a feasible set is the largest value of $w$ such that $P^{(n)}(w)$ is finite. The maximum-weight feasible set is easily obtained with standard backtracking techniques. Interestingly, we note that when coflows have equal weights, or more generally when their processing times and weights are oppositely ordered, the problem $1\|\sum w_j U_j$ can be solved in $\mathcal{O}(N \log N)$ time with the Moore-Hodgson algorithm (MHA) [21].

The above DP algorithm is used by `WDCoflow-DP` in function `RejectCoflow` to compute the set $\mathcal{R}$ of candidate coflows for rejection among those using the bottleneck port. The main advantage is that, as is easily proven, `WDCoflow-DP` is optimal when there is only one input port and one output port. This is not the case of `DCoflow` and `WDCoflow`, even for coflows with equal weights. However, the downside is that the running time of `WDCoflow-DP` is only pseudo-polynomial in the sum of coflow weights, whereas the complexity of `WDCoflow` is $\mathcal{O}(N^2)$ (see below).

### D. Complexity Analysis

The complexity of `WDCoflow` is $\mathcal{O}(N^2)$, as the `DCoflow` algorithm proposed in [19]. Specifically, in `WDCoflow`, the values $t_\ell$ and $\Psi_{\ell,k}$ can be initialized at cost $\mathcal{O}(NL)$, where $L = |\mathcal{L}|$ is the number of ports. Then these values can be updated at a cost $\mathcal{O}(L)$ per coflow at each iteration. The number of operations required to determine the coflow $k' \in \mathcal{S}_b$ with the largest deadline is $\mathcal{O}(N)$ and the complexity of `RejectCoflow` is $\mathcal{O}(N)$, so that finally across iterations it adds to $\mathcal{O}(N^2)$. As reported in [19], the complexity of `RemoveLateCoflows` is $\mathcal{O}(NL)$.

The only difference between `WDCoflow-DP` and `WDCoflow` is that the former uses a DP algorithm in `RejectCoflow` to restrict the choice of coflows that can be rejected. As the complexity of the DP algorithm is $\mathcal{O}(N\,W)$, where $W = \sum_j w_j$, the total complexity of `WDCoflow-DP` is $\mathcal{O}(N^2\,W)$. In the special case when coflow weights and processing times are oppositely ordered, the DP algorithm can be replaced by the Moore-Hodgson algorithm whose complexity is $\mathcal{O}(N \log(N))$, so that the complexity of `WDCoflow-DP` is reduced to $\mathcal{O}(N^2 \log(N))$ (we refer to this variant as `WDCoflow-MHA`).

The computational complexity of the different variants is summarized in Table III.

### E. Online Algorithm

The three variants of `WDCoflow` can also be performed in an online setting where coflows arrive sequentially and possibly in batches. For this, we introduce the update frequency, denoted as $f$. This frequency represents the instances at which `WDCoflow` recomputes the coflow scheduling order. The updates can occur either when new coflows arrive (in which case $f$ is set to infinity) or periodically with a period of $1/f$. In the online scenario, the scheduler is aware of the flow volumes of the coflows currently in the system. However, it does not have knowledge of the volumes or release times of future coflows.

During each update instant, the scheduler recalculates a new order of the current coflows in the network. These include coflows that were scheduled in the previous update but have not yet completed, the ones that were rejected in the previous update but still have remaining time before their deadline, and the ones that have arrived during the update interval. The new ordering is determined based on the remaining volumes of the flows, rather than the original volumes. Note that coflows can be preempted in this process [5]. This recomputation of the schedule occurs at each update instant.

TABLE III
COMPUTATIONAL COMPLEXITY OF VARIANTS OF `DCoflow`.

| `DCoflow` | `WDCoflow` | `WDCoflow-MHA` | `WDCoflow-DP` |
|---|---|---|---|
| $\mathcal{O}(N^2)$ | $\mathcal{O}(N^2)$ | $\mathcal{O}(N^2 \log(N))$ | $\mathcal{O}(N^2\,W)$ |

## IV. Performance Evaluation

In this section, we conduct an evaluation of our algorithms in comparison to state-of-the-art algorithms proposed in the literature. To ensure fairness and clarity, we begin by assigning equal weights to all coflows. This allows us to compare our algorithms against others that were developed without the ability to handle different coflow weights. In the second part of this section, we extend our evaluation to consider the case with different coflow weights.

### A. Simulation Setup

We evaluate via simulations[4] our proposed heuristics (three variants: `DCoflow`, `WDCoflow` and `WDCoflow-DP`) along with some existing algorithms such as `CS-MHA`[5] and the solution provided by the optimization method `CDS-LP` proposed in [12]. The relaxed version of `CDS-LP`, named `CDS-LPA`, is also implemented[6]. By using the solution obtained from `CDS-LP` as an upper bound, we can gain insight into how closely the evaluated algorithms approach the optimal solution. A concise overview of the reference algorithms has been provided in Sec. I. Furthermore, we conduct a comparative analysis by comparing our schedulers against two established algorithms,

[4]The flow-level simulator and the implementation of all algorithms can be found at https://github.com/luuquangtrung/CoflowSimulator.

[5]Only the centralized algorithm (`CS-MHA`) presented in [20] is reimplemented, as it has been shown, in the same paper, to be better than the decentralized version (`D`$^2$`-CAS`) in terms of CAR.

[6]It is worth noting that both `CDS-LP` and `CDS-LPA` use the same decision variables $\{z_k\}_{k\in\mathcal{C}}$ as those introduced in Problem (WCAR). In `CDS-LP`, $z_k$ are binaries, whereas in `CDS-LPA`, $z_k$ are continuous numbers and can take values in the range $[0, 1]$. For any solution obtained using `CDS-LPA`, only coflows $k$ for which the corresponding $z_k$ strictly equals 1 are considered as accepted ones.

namely Sincronia [24] and Varys [13] that aim to minimize the average CCT.

After obtaining the $\sigma$-order, the actual coflow resource allocation for our solution is performed using the greedy rate allocation algorithm GreedyFlowScheduling introduced in [24]. This algorithm reserves the entire bandwidth of a port for one flow at a time. It follows the order specified by $\sigma$, taking into account the corresponding coflow to which each flow belongs [24]. Note that for CDS-LP, CDS-LPA, and Varys, the rate allocation is incorporated within the algorithm itself.

The network comprises $M$ machines connected to a non-blocking Big-Switch fabric, where each access port has a normalized capacity of 1 unit. We assess the algorithms on small-scale and large-scale networks denoted as $[M, N]$, representing the fabric size and number of coflows ($N$) in the simulations. Small-scale networks consist of $M = 10$ machines, while large-scale networks have either 50 or 100 machines. Coflows in these networks are generated using either synthetic or real traffic traces from the Facebook dataset.

CDS-LP and CDS-LPA are solved using the MILP solver gurobi. Due to their high complexity, we only evaluate them on small-scale networks. The subsequent sections provide a comprehensive overview of the experimental setup, comparison metrics, and simulation results.

*Synthetic Traffic Traces.* The synthetic traffic consists of two coflow types. Type-1 coflows have only one flow, whereas Type-2 coflows have a varying number of flows following a uniform distribution in $[2M/3, M]$. Each generated coflow is randomly assigned to either Class 1 or Class 2 with probability of respectively 0.6 and 0.4.

Additionally, each coflow $k$ is assigned a random deadline within $[\text{CCT}_k^0, \alpha\text{CCT}_k^0]$, where $\text{CCT}_k^0$ represents the CCT of coflow $k$ *in isolation*, and $\alpha$ is a positive real value in $[2, 10]$. A higher value of $\alpha$ indicates that the scheduler has more flexibility in meeting the coflow deadlines.

*Real Traffic Traces* Real traffic traces are obtained from the Facebook dataset [13]. This dataset is based on a MapReduce shuffle trace collected from one of Facebook's 3000-machine cluster with 150 racks. The data traces contains a total of 526 coflows with varying widths, ranging from small ones with only one flow to the largest ones with 21170 flows. Detailed statistics of the Facebook dataset can be found in [5].

For each configuration $[M, N]$, $N$ coflows are randomly sampled from the Facebook dataset. They are only chosen from the coflows that have at most $M$ flows. The volume of each flow is already given by the dataset.

*Weight Classes.* For WDCoflow and WDCoflow-DP, we categorize coflows into two classes of weights for both synthetic and real traffic. The weight assigned to each coflow reflects its importance level. Class-1 coflows are assigned a weight $w_1 = 1$, while Class-2 coflows are assigned a weight $w_2$ of either 2 or 10. The probability that a generated coflow falls into Class 1 and Class 2 are respectively $p_1$ and $p_2 = 1 - p_1$.

*Metric.* We evaluate the algorithms based on the average weighted CAR, $\text{WCAR} = \frac{\sum_{k\in\mathcal{C}} w_k z_k}{\sum_{k\in\mathcal{C}} w_k}$ for the weighted setting. In the unweighted setting, WCAR is just the average CAR, where $w_k = 1$, $\forall k \in \mathcal{C}$. We also present the gains in percentiles of each algorithm with respect to the solution provided by CDS-LP in terms of WCAR. These gains are calculated using the formula: average gain in WCAR $= \frac{\text{compared WCAR}}{\text{WCAR under CDS-LP}} - 1$.

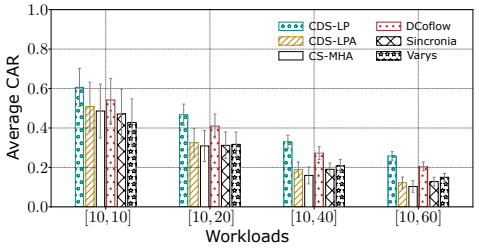
In addition, the per-class CAR is evaluated. For class $i \in \{1, 2\}$, it is defined as the number of admitted coflows of class $i$ divided by the total number $N_i$ of coflows of this class, i.e., $\left(\sum_{k\in\mathcal{C}} \mathbb{1}_{k,i} z_k\right)/N_i$, where $\mathbb{1}_{k,i} = 1$ if coflow $k$ is of class $i$, and $\mathbb{1}_{k,i} = 0$, otherwise.

### B. Scheduling Unweighted Coflows

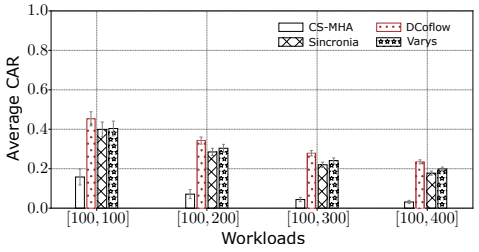
In this section, we assess the performance of our unweighted algorithm, DCoflow, and compare it with other algorithms described in section IV-A using the same weights for all coflows. We recall that the unweighted case represents the evaluation of DCoflow initially developed in [19].

*1) Results with Offline Setting:* In the offline setting, we assume that all coflows arrive simultaneously with a release time of zero. For each simulation with a specific scale of the network and either synthetic or real traffic traces, we randomly generate 100 different instances and calculate the average performance of all algorithms over 100 runs.

*a) Average CAR Under Synthetic Traffic:* Figs. 2a–2b show the average CAR with respectively small-scale networks and large-scale networks. The percentile gains of each algorithm with respect to CDS-LP are shown in Fig. 4a, in terms of average CAR for the configuration $[10, 60]$.




(a) Synthetic traffic traces on small-scale networks.




(b) Synthetic traffic traces on large-scale networks.

Fig. 2. Average CAR with synthetic traffic traces using (a) small-scale and (b) large-scale networks. Each point in the x-axis represents the network $[M, N]$.

The results show that DCoflow exhibits the closest performance to the optimal solution yielded by CDS-LP in terms of CAR compared to all other algorithms. This holds true for both small- and large-scale networks. Surprisingly, DCoflow even outperforms CDS-LPA which is an approximation version of CDS-LP. These findings indicate the effectiveness of DCoflow in achieving near-optimal performance

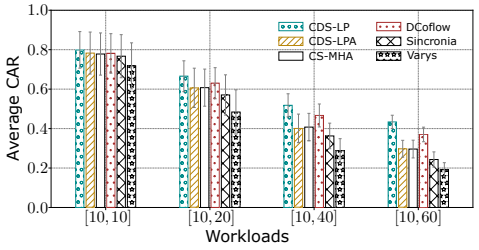
for coflow scheduling. For instance, with the configuration $[10, 10]$, DCoflow improves the CAR on average by $6.5\%$, $11.5\%$, $15.1\%$, and $26.6\%$, compared respectively to CDS-LPA, CS-MHA, Sincronia, and Varys. The improvement in average CAR becomes more pronounced as the load increases. Specifically, the corresponding improvement on average CAR a configuration $[10, 60]$ are $67.2\%$, $98.3\%$, $59.9\%$, and $36.8\%$ (see Fig. 2a). The improvement in performance is even more substantial when evaluated on a large-scale network. For example, compared to CS-MHA, Sincronia, and Varys, with the configuration $[100, 400]$, the improvement in terms of average CAR are respectively $648.1\%$, $32.3\%$, and $17.9\%$. (see Fig. 2b). It is worth noticing how the performance of CS-MHA falls drastically when dealing with large-scale configurations. This behavior can be attributed to the prioritization strategy of CS-MHA, which favors coflows that utilize a large number of ports over those that require only a few. In scenarios where there are numerous coflows with a small number of ports, the CAR of CS-MHA tends to approach zero (see detailed explanation of this behavior with a motivating example in Sec. II-C).

The results depicted in Figure 4a highlight that DCoflow consistently achieves a smaller gap to the optimal solution across a wide range of percentile values compared to other algorithms. In particular, when compared to Sincronia, DCoflow improves the CAR in $50\%$ of the $100$ instances by $50\%$, and it achieves an approximately $43\%$ improvement at the $99^{\text{th}}$ percentile.

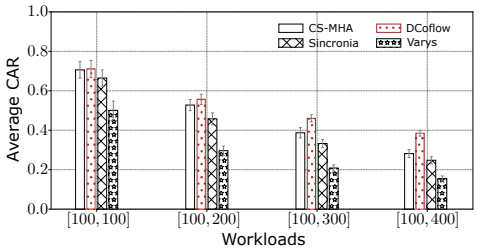
*b) Average CAR Under Real Traffic Traces:* This section presents the results obtained with the Facebook traffic traces, using the same configurations as those used in Sec. IV-B1a. Figs. 3a–3b show the average CAR with respectively small- and large-scale networks. The gains in percentiles of each algorithm with respect to CDS-LP, in terms of average CAR when using a $[10, 60]$ network are shown in Fig. 4b. Similar to the results obtained using the synthetic traces (see Sec. IV-B1a), DCoflow demonstrates a substantial improvement in terms of average CAR compared to other heuristics. For instance, with a $[10, 60]$ configuration, DCoflow improves the average CAR by an average of $24.4\%$, $25\%$, $52.2\%$, and $93.1\%$ compared respectively to CDS-LPA, CS-MHA, Sincronia, and Varys(see Fig. 3a). The improvement is even higher when performed on a large network configuration. For example, compared to CS-MHA, Sincronia, and Varys, on a $[100, 400]$ network, the improvement in terms of average CAR are respectively $36.6\%$, $55.3\%$, and $147.5\%$.

Moreover, the results in Fig. 4b show that DCoflow consistently achieves a smaller gap to the optimal solution across various percentiles compared to the other algorithms. Specifically, compared to Sincronia, DCoflow improves the CAR in $57\%$ of $100$ instances by $50\%$, and it achieves around $35\%$ at the $99^{\text{th}}$ percentile.

*c) Prediction Error of* DCoflow*:* It is worth noticing that the final solution provided by DCoflow does not necessarily guarantee that every coflows in $\sigma$ will eventually meet their deadlines. The estimated CCT of coflows may differ from the actual CCTs obtained after the rate allocation process due to the coupling between input and output ports. The prediction

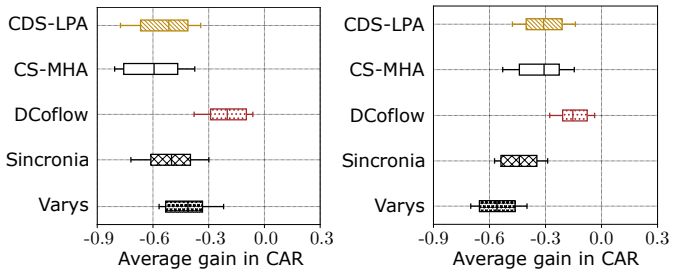



(a) Facebook traffic traces on small-scale networks.




(b) Facebook traffic traces on large-scale networks.

Fig. 3. Average CAR with Facebook traces using (a) small-scale and (b) large-scale networks. Each point in the x-axis represents network $[M, N]$.




(a) Synthetic traces. (b) Facebook traces.

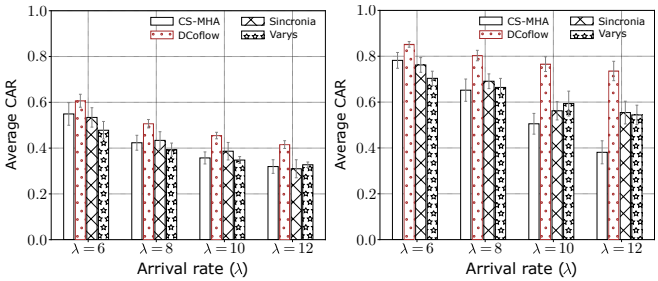
Fig. 4. The $1^{\text{st}}$-$10^{\text{th}}$ -$50^{\text{th}}$-$90^{\text{th}}$-$99^{\text{th}}$ percentiles of the average gain in CAR with small-scale network $[10, 60]$ using (a) synthetic and (b) Facebook traces.

error of DCoflow represents the gap between the estimated CAR and the actual CAR after resource allocation. This error is given by $(|\sigma| - |\hat{\sigma}|)/|\sigma|$, where $\hat{\sigma} \subseteq \sigma$ is the subset of coflows in $\sigma$ that meet their deadlines after applying the actual rate allocation using GreedyFlowScheduling.

In the simulations presented in Sec. IV-B, we observe an average CAR prediction error of below $3.6\%$ of DCoflow for both synthetic and real traffic traces.

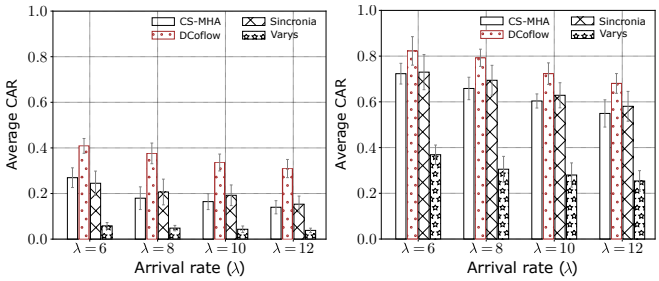
*2) Online Setting:* We now present a series of numerical results regarding the performance of the online version of DCoflow. The evaluation metric used is the average CAR obtained from $40$ instances. For the synthetic traffic, in each instance, coflows arrive sequentially based on a Poisson process with a rate of $\lambda$, i.e., the inter-arrival time of coflows is exponentially distributed with rate $\lambda$. For real traffic traces, since the inter-arrival time of coflows traces from Facebook dataset drive a very low load on the network, we consider the actual Facebook arrival traces (525 in total) and, to obtain different arrival rates, we scale them over different time scales to obtain a network load of between 0.8 and 0.98. In this way, the inter-arrival distribution of the coflow used in our experimentation is identical to that of the Facebook traces. By default, coflow priorities are computed when a new coflow arrives ($f = \infty$), unless otherwise specified.

We compare the average CAR achieved by DCoflow with




(a) Small-scale network. (b) Large-scale network.

Fig. 5. Average CAR using synthetic traffic with varying $\lambda$ and (a) $M = 10$ and (b) $M = 50$.




(a) Small-scale network. (b) Large-scale network.

Fig. 6. Average CAR using Facebook traffic with varying $\lambda$ and (a) $M = 10$ and (b) $M = 100$.

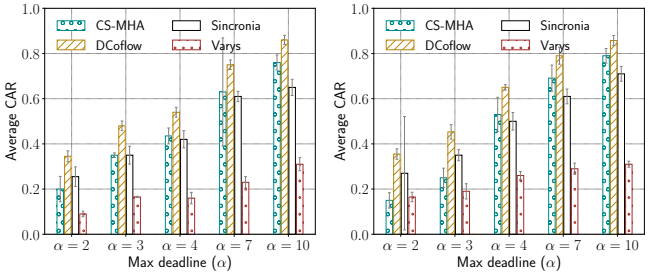
the online version of Varys with deadline [25], `CS-MHA`, and `Sincronia`. We examine the impact of two key parameters: (*i*) the coflow arrival rate $\lambda$ and (*ii*) the frequency $f$ at which coflow priorities are recomputed.

*a) Impact of Arrival Rate:* We begin by examining the impact of the arrival rate $\lambda$ on the CAR achieved by different algorithms. The CAR is averaged over 40 instances, each consisting of 4000 coflow arrivals. The deadline for each coflow $k$ is randomly selected from a uniform distribution in the range $[\text{CCT}_k^0, 4\text{CCT}_k^0]$. Two scenarios are considered: a small-scale networks with $M = 10$ machines and a large-scale networks with $M = 50$ machines. In each scenario, we present the results for the following values of $\lambda$: $\lambda = 8$, $\lambda = 12$, $\lambda = 16$, and $\lambda = 20$.

Figs. 5a and 5b depict the results for respectively the small and large network. These results show that `DCoflow` obtains a higher average CAR for all values of $\lambda$. Moreover, the gain performance of `DCoflow` with respect to the other algorithms increases with the value of $\lambda$. While the other algorithms may exhibit similar CAR in lightly loaded fabrics, `DCoflow` clearly outperforms them when the network is heavily congested.

Figs. 6a and 6b show respectively the average CAR when using the configuration of $M = 10$ and $M = 100$, both with 4000 coflows, with the Facebook dataset. Similar to what was observed with the synthetic traffic traces, `DCoflow` significantly outperforms all other methods. When dealing with a highly congested network (i.e., high value of $\lambda$), again `DCoflow` yields a higher gain compared to the other methods. For instance, when $M = 10$ (see Fig. 6a), `DCoflow` achieves a gain of 13%–34% against `CS-MHA` and 19%–31% against `Sincronia` for moderated network load. For high network load, `DCoflow` achieves a gain of 62.25%–75.18% against `CS-MHA` and 15%–43.33% against `Sincronia`. An important observation is that `CS-MHA` obtains good performance when the network load is slightly high, but its performance deteriorates when the network load increases. On the other hand, `DCoflow` performance is not affected by network load and always delivers the best performance under all network load conditions. These observations also apply to large-scale networks, and the performance of `DCoflow` compared to other schedulers is even more better than in the case of $M = 10$ (see Fig. 6b).

*b) Impact of Deadlines:* Here we examine the impact of the maximum deadline, i.e., $\alpha CCT^0$, on the CAR achieved by different algorithms. The CAR is averaged over 40 instances, each consisting of 4000 coflow arrivals, with $M = 10$ and $M = 100$. Fig. 7a and Fig. 7b show respectively the average CAR when changing the maximum deadline from tight deadline ($\alpha = 2$) to flexible deadline ($\alpha = 10$). For large-scale networks (see Fig. 7b), we observe that `DCoflow` achieves a gain of up to 136% against `CS-MHA` when the deadline is tight, and its gain decreases as the deadline becomes more flexible, reaching 8% for $\alpha = 10$. In comparison with `Sincronia` the gain remains almost stable between 20% and 31%. The same conclusion can be drawn for small-scale networks (see Fig. 7a).

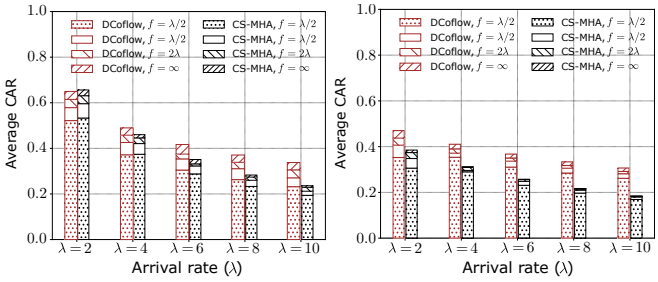



(a) Small-scale network, $\lambda = 10$. (b) Large-scale network, $\lambda = 12$.

Fig. 7. Average CAR using Facebook traffic with real arrival times and varying $\alpha$ and (a) $M = 10$ and (b) $M = 100$.

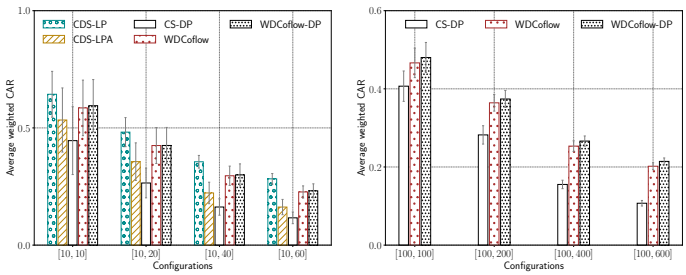
*c) Impact of Update Frequency:* To evaluate the impact of the update frequency $f$ on the average CAR, the following values of $f$ are considered: $f = \frac{\lambda}{2}$, $f = \lambda$, $f = 2\lambda$, and $f = \infty$. Recall that $f = \infty$ indicates that priorities are recomputed upon each arrival of a new coflow. We assume that $M = 10$ and compute the CAR by averaging over 40 instances. For each instance, 8000 coflow arrivals are generated, following a Poisson process of rate $\lambda$. The deadline of a coflow $k$ follows a uniform distribution in the range $[\text{CCT}_k^0, 2\text{CCT}_k^0]$. We examine the average CAR for different values of $f$ ($f \in \{\frac{\lambda}{2}, \lambda, 2\lambda, \infty\}$) and of the arrival rate which takes values in the range $[2, 10]$.

Fig. 8a shows the results obtained from a simulation, in which each arrival corresponds to one single coflow. Similar to previous findings, for a low arrival rate $\lambda$, both `DCoflow` and `CS-MHA` achieve a similar average CAR performance: for $\lambda = 2$, `CS-MHA` achieves a slightly higher CAR than `DCoflow`). But when the network is highly congested, `DCoflow` significantly outperforms `CS-MHA`. Additionally, increasing the frequency $f$ has a noticeable positive impact on the CAR for both algorithms. For example, for $\lambda = 2$ (resp. $\lambda = 10$),




(a) Without batch. (b) With batch.

Fig. 8. Average CAR of `DCoflow_v1` and `CS-MHA` using synthetic traffic with $[10, 8000]$ and varying $\lambda$, when obtaining (a) one single coflow per arrival; and (b) a random batch of coflow per arrival.




(a) Small-scale networks. (b) Large-scale networks.

Fig. 9. Average WCAR with coflows of all classes using synthetic traffic traces and (a) small-scale and (b) large-scale networks. The values of $(p_2, w_2)$ are set to $(0.2, 2)$. Each point in the x-axis represents the network $[M, N]$.

updating coflow priorities upon each arrival (i.e., $f = \infty$) instead of using the periodic scheme with $f = \frac{\lambda}{2}$ leads to an average CAR increase of $52\%$ (resp. $46\%$). These results suggest that there is a trade-off between the computational complexity of updating coflow priorities at a high frequency and the achieved CAR. Fig. 8b shows a similar analysis, but this time we assume that coflows arrive in batches. The size of each batch is randomly drawn from a uniform distribution $\mathcal{U}([5, 15])$. In this scenario, to ensure that the coflow arrival rates are comparable to the previous setting (Fig. 8a), where coflows arrive individually, we divide the batch arrival rate by 10. This adjustment allows us to maintain the same coflow arrival rates for both settings.

The results achieved for simulations with batch arrivals are similar to those obtained with the previous setting, but we note that `DCoflow` continues to exhibit significant gains over `CS-MHA`. Additionally, we observe that the benefits of using a higher update frequency are relatively lower in this scenario. For instance, when $\lambda = 10$, increasing the update frequency from $f = \frac{\lambda}{2}$ to $f = \infty$ results in only a $17\%$ increase in the average CAR.

### C. Scheduling Weighted Coflows

We now evaluate the weighted versions of our proposed algorithm, `WDCoflow` and `WDCoflow-DP`, along with `CS-DP`, `CDS-LP`, and `CDS-LPA`. `CS-DP` is the adapted version to the weights of `CS-MHA` presented in [20] (in which the Moore-Hodgson algorithm is replaced by the DP algorithm in Section III-C), whereas `CDS-LP` and its relaxed variant `CDS-LPA` are straightforward adaptations of the linear-programming methods proposed in [12] to account for coflow weights. By using the solution derived from `CDS-LP` as an upper bound, we can get the sense of how close the algorithms are to the optimum.

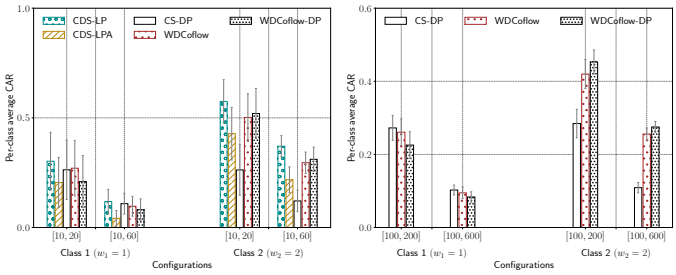
*1) Offline Setting:* In the offline setting, we consider that all coflows arrive at the same time, i.e., their release time is zero. For each simulation with a specific scale of the network and either synthetic or real traffic traces, we randomly generate 100 different instances and compute the average performance of algorithms over 100 runs. We evaluate `WDCoflow` and `WDCoflow-DP` against existing algorithms for the offline case.

*a) Synthetic Traffic:* Figs. 9a and 9b shows the average WCAR with synthetic traffic traces using small-scale ($M = 10$) and large-scale ($M = 100$) networks. It is observed that our proposed heuristics (`WDCoflow` and `WDCoflow-DP`) are closest in terms of WCAR to the optimum (`CDS-LP`) than all other algorithms in both small- and large-scale network configurations. `WDCoflow-DP` yields a slightly higher performance compared to `WDCoflow` (around $4.2\%$) when using large-scale network settings (see Fig. 9b). For small scale network, we observe that the optimum `CDS-LP` obtain only a gain of $9\%$ and $7\%$ compared to `WDCoflow` and `WDCoflow-DP` respectively for $N = 10$ . On the other hand, CDS-LPA and CS-DP are far from the optimal solution by $17\%$ and $30\%$ respectively. For the worst case when $N = 60$, `WDCoflow` and `WDCoflow-DP` are far from the optimal solution by $19\%$ and $17\%$ respectively, but CDS-LPA and CS-DP moves away from the optimum by $42\%$ and $58\%$ respectively.

Now, for large-scale networks, we observe that `WDCoflow` and `WDCoflow-DP` obtain a significant performance improvement compared to `CS-DP`. Indeed, for $N = 100$, `WDCoflow` and `WDCoflow-DP` obtain a gain of $14\%$ and $18\%$ respectively. When the number of coflows increases, both algorithms obtain gains up to $184\%$ for `WDCoflow` and $192\%$ for `WDCoflow-DP` compared to `CS-DP`.

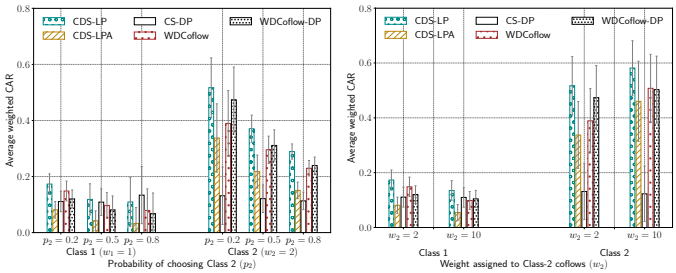
With respect to the performance of each class, Figs. 10a and 10b show the average WCAR of each coflow class using small and large-scale networks. As expected, the performance is even more significant for traffic of Class 2 since `WDCoflow` and `WDCoflow-DP` consider both the network conditions and coflows' importance to perform the scheduling, while `CS-DP` prioritizes coflows that use a large number of ports over those that use a few. In Fig. 10b, we can see that `WDCoflow` and `WDCoflow-DP` perform about $21\%$ and $51\%$ for $N = 100$ and $247\%$ and $258\%$ for $N = 600$ better than `CS-DP` for Class 2. For class-1 coflows, our heuristic achieves a moderate gain of up to $10\%$ for $N = 600$ compared to `CS-DP`.

Figs. 11a and 11b illustrate respectively the per-class WCAR with synthetic traffic traces using network configuration $[10, 60]$ when varying $p_2$ (with fixed $w_2$) and $w_2$ (with fixed $p_2$). We can see that both schedulers `WDCoflow` and `WDCoflow-DP` obtain almost the same performance compared to the optimal solution and they handle the priority between classes as `CDS-LP`. But for Class 1, we can see that `CS-DP` performs best for $p_2 = 0.5$ and $p_2 = 0.8$. This means that `WDCoflow` and `WDCoflow-DP` take into account the importance of weight on how to schedule the coflows.



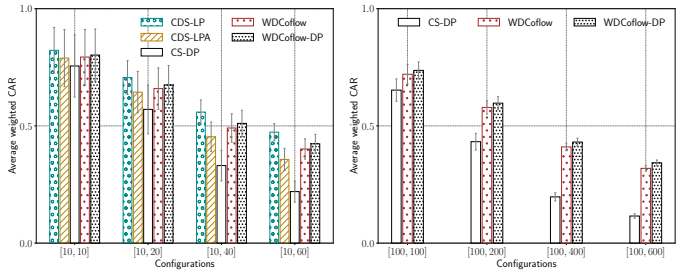

(a) Small-scale networks. (b) Large-scale networks.

Fig. 10. Average per-class CAR with synthetic traffic traces using (a) small-scale and (b) large-scale networks. The value of $p_2$ is set to 0.2. Each point in the x-axis represents the network $[M, N]$.



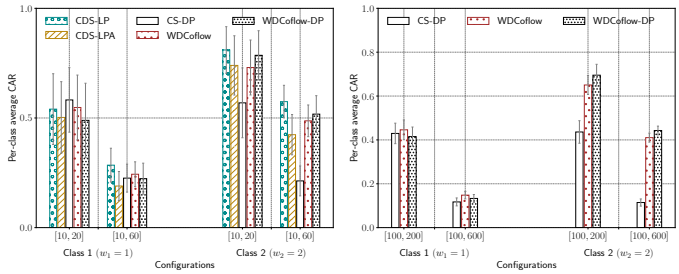

(a) Varying $p_2$ with fixed $w_2$. (b) Varying $w_2$ with fixed $p_2$.

Fig. 11. Average per-class CAR with synthetic traffic traces using network configuration $[10, 60]$ when (a) varying $p_2$ with fixed $w_2 = 2$ and (b) varying $w_2$ with fixed $p_2 = 0.2$.




(a) Small-scale networks. (b) Large-scale networks.

Fig. 12. Average WCAR with Facebook traffic traces using (a) small and (b) large-scale networks. The values of $(p_2, w_2)$ are set to $(0.2, 2)$. Each point in the x-axis represents the network $[M, N]$.




(a) Small-scale networks. (b) Large-scale networks.

Fig. 13. Average per-class CAR with Facebook traffic traces using (a) small-scale and (b) large-scale networks. Each point in the x-axis represents the network $[M, N]$.

*b) Real Traffic Traces:* Now we use Facebook traces to evaluate the performance of `WDCoflow` and `WDCoflow-DP`.

Figs. 12a and 12b show the average WCAR with Facebook traffic traces using small and large-scale networks. The figures show that `WDCoflow` and `WDCoflow-DP` provide near-optimal solutions (the difference is less than $3\%$) while CDS-LPA and CS-DP are far from the optimum of $5\%$ and $8\%$ respectively. For high load ($N = 60$), `WDCoflow` and `WDCoflow-DP` lose only $10\%$ and $15\%$ respectively compared to the optimal but the other two algorithms lose more ground by about $53\%$ compared to the optimal. Moreover, the performance gap becomes higher with the increase of the network scale. For instance, with network $[100, 100]$, `WDCoflow` and `WDCoflow-DP` perform around $6.8\%$ and $8.4\%$ better than `CS-DP` while with the network $[100, 600]$, these gaps become $20\%$ and $22\%$, respectively.

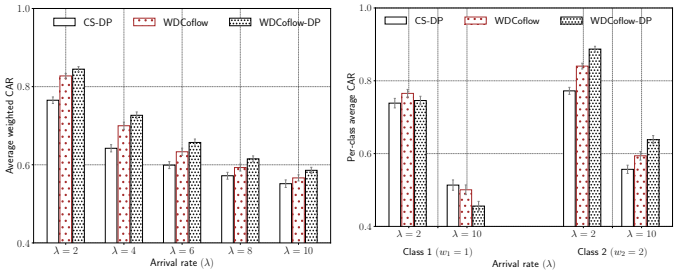
We observe that `WDCoflow` and `WDCoflow-DP` approximate the performance of the optimal solution for the small-scale network and perform better than `CS-DP` w.r.t. average WCAR. Moreover, the performance gap becomes higher with the increase of the network scale. For instance, with network $[100, 100]$, `WDCoflow` and `WDCoflow-DP` perform around $10\%$ and $13\%$ better than `CS-DP` while with network $[100, 600]$, these gaps become $331\%$ and $345\%$, respectively.

Figs. 13a and 13b shows the average CAR of each coflow class with Facebook traffic traces using small and large-scale networks, with $(p_2, w_2) = (0.5, 2)$. We can see that `WDCoflow` and `WDCoflow-DP` achieve big improvement for Class 2, while `CS-DP` obtains worse performance for both classes. The reason for this is that `CS-DP` schedules coflows only according to the network conditions, while `WDCoflow` and `WDCoflow-DP` consider both network conditions and coflow weights. Under `WDCoflow` and `WDCoflow-DP` higher weight coflows have higher priority, thus the average WCAR greatly increases.

*2) Online Setting:* We now present numerical results comparing the performance of the online version of `WDCoflow` and `WDCoflow-DP` against the online version of `CS-DP`. The results are obtained on instances generated using our workload generator, with $50$ machines and $3000$ coflow arrivals. Coflow arrivals follow a Poisson process with an average rate of $\lambda$ (coflows/time slot). The arrival rate $\lambda$ varies from 2 to 10, and the probability and weight of Class-2 coflows are fixed to respectively $0.5$ and $10$. For the sake of comparison, we have used the greedy allocation algorithm (see the beginning of Sec. IV) to perform the resource allocation after obtaining the $\sigma$-order. For each algorithm the average performance is calculated over $40$ runs with $40$ different instances of the same setting.

Fig. 14a and 14b illustrate the WCAR and per-class CAR of coflows. We observe that `WDCoflow` and `WDCoflow-DP` improve the average WCAR as compared to `CS-DP`. For instance, with $\lambda = 4$, the WCAR improvement of `WDCoflow` and `WDCoflow-DP` compared to `CS-DP` are respectively $7\%$ and $12\%$. In addition, they greatly improve the CAR for Class 2 for all $\lambda$ compared to `CS-DP` (see Fig. 14b). This shows that our proposed solution consider both network conditions and the importance of coflows to determine the $\sigma$-order. This allows to improve the average CAR and also to differentiate the CAR

for a specific target class.




(a) Average WCAR. (b) Per-class CAR.

Fig. 14. (a) Average WCAR and (b) per-class CAR with synthetic traffic traces when varying $\lambda$ and fixing $p_2 = 0.5$ and $w_2 = 2$.

## V. Related Work

In the literature, there has been a stronger emphasis on minimizing the CCT of coflows rather than considering deadline-sensitive scheduling. This discrepancy highlights the relatively lower attention given to deadline scheduling. One of the earlier algorithms that addresses deadline-sensitive coflow scheduling is `Varys` [13]. `Varys` employs a cascade of coflow admission control and scheduling mechanisms. The scheduler aims to minimize CCT through a combination of strategies, including (*i*) a heuristic for coflow ordering based on the bottleneck's completion time for each coflow and (*ii*) an allocation algorithm that assigns bandwidth to individual flows within each coflow. The rate allocation in `Varys` is designed to approximately align the completion times of all coflows with the bottleneck completion time.

`Chronos` [25] is another heuristic algorithm specifically designed for deadline scheduling. It addresses the issue of flow starvation by allocating residual bandwidth to flows that do not meet their deadlines. The algorithm begins by establishing a priority order among the coflows. Each coflow is then allocated the minimum required bandwidth to meet its individual deadline. If there is insufficient bandwidth available for a particular coflow, it is removed from the allocation and marked for multiplexing. After allocating bandwidth to all flows that meet their deadlines, the remaining bandwidth is distributed proportionally among the remaining coflows based on their demands. This ensures that coflows that cannot fully meet their deadlines still receive a fair share of the available bandwidth.

In [20], the authors establish a connection between the problem of deadline scheduling of coflows and the concurrent open shop problem, which is a well-known *NP*-hard problem. They propose a heuristic approach based on the Moore-Hodgson algorithm [21], which deals with the case of single link. A centralized and decentralized version of the heuristic are introduced, namely `CS-MHA` and `D`$^2$`-CAS`, respectively.

A formal formulation for the deadline scheduling problem including bandwidth allocation of flows is introduced in [12]. The CDS maximization problem is cast as an MILP (called `CDS-LP`). In the formulation, time is divided into intervals based on the boundaries set by the coflows deadlines, arranged in increasing order. The objective of `CDS-LP` is to determine which coflows to accept and the corresponding amount of bandwidth to allocate in each interval to maximize the overall satisfaction of coflow deadlines. The problem takes into account the inherent trade-off between accepting more coflows and allocating sufficient bandwidth to meet their deadlines. `CDS-LP` is proven to be *NP*-hard, indicating that finding an optimal solution is computationally challenging. As an alternative, they propose an approximation algorithm based on LP relaxation, referred to as `CDS-LPA`. `CDS-LPA` relaxes the binary variables in the MILP formulation and retains only the coflows that are completely accepted according to the relaxed variables (i.e., their relaxed variables are strictly equal to 1).

An online heuristic to maximize coflow admissions, while respecting their deadlines, is presented in [26]. The authors only focus on comparing their heuristic with `Varys`, acknowledging that other more efficient algorithms have been developed in the literature, such as those presented in [12, 20, 25].

`MixCoflow` [27] addresses the problem of simultaneous optimization of coflows with and without deadline. The paper formulates an optimization framework to schedule coflows, with objective to minimize and balance the bandwidth usage of coflows with deadlines, allowing coflows without deadlines to be scheduled as soon as possible. The framework is fist cast as an ILP, then an equivalent LP problem has been investigated to obtain the optimal solution with lesser computational complexity.

The work in [28] considers the scenario where the network is overloaded and it becomes impossible to complete all coflows within their respective deadlines. The proposed solution, namely `Poco`, leverages the observation that certain parallel time-sensitive data applications can tolerate incomplete or partial transmission of their data. `Poco` proposes a mechanism to order the coflows at the limit of the tolerance of each application.

For completeness, we also cite additional works focusing on the problem of minimizing CCT of coflows [5–8, 29–33] as well as the survey article [34]. For instance, [31] introduces `SmartCoflow` to solve the joint problem of endpoint placement and coflow scheduling, with objective to minimize the average CCT of coflows across geo-distributed datacenters. Another coflow scheduler, namely `Parrot`, is presented in [32]. `Parrot` leverages the least per-coflow attained service policy to infer the job with shortest remaining processing time. Among these work on coflow CCT minimization, the algorithm `Sincronia` [24] has gained popularity. It addresses CCT minimization by scheduling coflows on network bottlenecks and provides a scheduling order that achieves a 4-approximation factor.

## VI. Conclusion and Future Work

In this paper, we have presented a novel approach for handling coflow admission control and scheduling in the context of batch processing with deadline constraints. Our algorithm takes advantage of open-shop scheduling techniques to identify a subset of coflows to be scheduled and determines a $\sigma$ priority order, for efficient execution. By utilizing this $\sigma$-order, coflows are scheduled based on their priority, ensuring

effective management of deadlines and improved overall performance.

The experimental evaluation of our algorithms demonstrates promising performance on small-scale networks, where they either match or outperform other existing deadline-sensitive algorithms proposed in prior works. However, the true strength of our approach is revealed on large-scale networks, where it exhibits substantial improvements compared to the existing algorithms. For instance, in an offline setting, our scheme achieves a significantly higher CAR, such as a remarkable $98\%$ increase compared to `CS-MHA`. Additionally, our proposed algorithm showcases a remarkable accuracy in prediction: Even though the admission control is performed using a CCT approximation with bottleneck ports, the proposed algorithm ensures that nearly all accepted coflows are able to complete within their assigned deadlines when they are actually scheduled.

This behavior is observed in various scenarios and network settings, including offline and online scenarios, using a wide range of network scales with either synthetic or real traces from the Facebook data set. This demonstrates the robustness and efficacy of the proposed algorithm when dealing with different situations.

Several extensions of this research line are possible and will be considered for future works. Specifically, the problem of scheduling coflows with incomplete information, e.g., the volume of flows of different coflows. This could occur when the exact volume of a flow of a given coflow is not directly available to the scheduler, but is instead inferred from *a priori* distribution. Understanding how our algorithm performs under such circumstances can provide insights into its robustness and adaptability to uncertain or incomplete information. Finally, fairness among coflows is also an important issue that has not been addressed in our current work. Future research could focus on developing new algorithms that promote fairness among coflows, ensuring equitable treatment of coflows and improving overall system performance.

## REFERENCES


[1] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. ACM HotNets*, Redmond, Washington, 2012, pp. 31–36.

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

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica *et al.*, "Spark: Cluster computing with working sets." *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[4] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 98–109, 2011.

[5] M. Chowdhury, "Coflow: A networking abstraction for distributed data-parallel applications," Ph.D. dissertation, University of California, Berkeley, Nov. 2015.

[6] M. Shafiee and J. Ghaderi, "An improved bound for minimizing the total weighted completion time of coflows in datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1674–1687, 2018.

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

[8] A. Arfaoui, R. El-Azouzi, F. De Pellegrini, C. Richier, and J. Leguay, "Elite: Near-optimal heuristics for coflow scheduling," in *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2022, pp. 665–674.

[9] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang, "On scheduling coflows," *Algorithmica*, vol. 82, no. 12, pp. 3604–3629, 2020.

[10] M. Shafiee and J. Ghaderi, "An improved bound for minimizing the total weighted completion time of coflows in datacenters," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1674–1687, 2018.

[11] M. Chowdhury *et al.*, "Near optimal coflow scheduling in networks," in *Proc. ACM SPAA*, Phoenix, AZ, USA, June 22-24 2019, pp. 123–134.

[12] S.-H. Tseng and A. Tang, "Coflow deadline scheduling via network-aware optimization," in *Proc. Annu. Allert. Conf. Commun. Control Comput.*, 2018, pp. 829–833.

[13] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient Coflow Scheduling with Varys," in *Proc. ACM SIGCOMM*, 2014, pp. 443–454.

[14] C. G. Jones, R. Liu, L. Meyerovich, K. Asanovic, and R. Bodik, "Parallelizing the web browser," in *Proc. the First USENIX Workshop on Hot Topics in Parallelism*, 2009.

[15] J. Xia, G. Zeng, J. Zhang, W. Wang, W. Bai, J. Jiang, and K. Chen, "Rethinking transport layer design for distributed machine learning," in *Proc. the 3rd Asia-Pacific Workshop on Networking 2019*, 2019, pp. 22–28.

[16] J. K. Lenstra and D. B. Shmoys, "Elements of scheduling," *arXiv preprint arXiv:2001.06005*, 2020.

[17] A. S. Schulz, "Polytopes and Scheduling," Ph.D. dissertation, Technische Universität Berlin, 1996.

[18] B. Lin and A. Kononov, "Customer order scheduling to minimize the number of late jobs," *Eur. J. Oper. Res.*, vol. 183, no. 2, pp. 944–948, 2007.

[19] Q.-T. Luu, O. Brun, R. El-Azouzi, F. De Pellegrini, B. J. Prabhu, and C. Richier, "Dcoflow: Deadline-aware scheduling algorithm for coflows in datacenter networks," in *2022 IFIP Networking Conference (IFIP Networking)*, 2022, pp. 1–9.

[20] S. Luo, H. Yu, and L. Li, "Decentralized deadline-aware coflow scheduling for datacenter networks," in *Proc. IEEE ICC*, 2016, pp. 1–6.

[21] J. M. Moore, "An n job, one machine sequencing algorithm for minimizing the number of late jobs," *Manag. Sci.*, vol. 15, no. 1, pp. 102–109, 1968.

[22] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan, "Minimizing the sum of weighted completion times in a concurrent open shop," *Oper. Res. Lett.*, vol. 38, no. 5, pp. 390–395, 2010.

[23] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," in *Annals of discrete mathematics*. Elsevier, 1979, vol. 5, pp. 287–326.

[24] S. Agarwal, R. Agarwal, S. Rajakrishnan, D. Shmoys, A. Narayan, and A. Vahdat, "Sincronia: Near-Optimal Network Design for Coflows," in *Proc. ACM SIGCOMM*, 2018, pp. 16–29.

[25] S. Ma, J. Jiang, B. Li, and B. Li, "Chronos: Meeting Coflow Deadlines in Data Center Networks," in *Proc. IEEE ICC*, 2016.

[26] A. Hasnain and H. Karl, "Coflow scheduling with performance guarantees for data center applications," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, 2020, pp. 850–856.

[27] R. Xu, W. Li, K. Li, X. Zhou, and H. Qi, "Scheduling mix-coflows in datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2002–2015, 2020.

[28] S. Luo, P. Fan, H. Xing, and H. Yu, "Meeting coflow deadlines in data center networks with policy-based selective completion," *IEEE/ACM Transactions on Networking*, vol. 31, no. 1, pp. 178–191, 2023.

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

[30] W. Li, D. Guo, A. X. Liu, K. Li, H. Qi, S. Guo, A. Munir, and X. Tao, "Coman: Managing bandwidth across computing frameworks in multiplexed datacenters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 5, pp. 1013–1029, 2018.

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

[32] W. Li, S. Chen, K. Li, H. Qi, R. Xu, and S. Zhang, "Efficient online scheduling for coflow-aware machine learning clusters," *IEEE Transactions on Cloud Computing*, vol. 10, no. 4, pp. 2564–2579, 2022.

[33] L. Shi, Y. Liu, J. Zhang, and T. Robertazzi, "Coflow scheduling in data centers: routing and bandwidth allocation," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2661–2675, 2021.

[34] S. Wang, J. Zhang, T. Huang, J. Liu, T. Pan, and Y. Liu, "A survey of coflow scheduling schemes for data center networks," *IEEE Commun. Mag.*, vol. 56, no. 6, pp. 179–185, 2018.

**Olivier Brun** is a CNRS research staff member at LAAS, in the SARA group. He graduated from the Institut National des Télécommunication (INT, Evry, France) and he was awarded his PhD degree from Université Toulouse III (France). His research interests lie in queueing and game theories as well as network optimization.

**Rachid El-Azouzi** is a full professor at the University of Avignon. He received his PhD in Applied Mathematics from Mohammed V University in 2000. He joined the National Institute for Research in Computer Science and Control (INRIA), in Sophia Antipolis, where he held positions as a postdoctoral fellow and research engineer. In 2003, he joined the University of Avignon as an associate professor. His research interests include networked games, resource allocation, wireless networks, complex systems and performance evaluation.

**Quang-Trung Luu** is currently a lecturer at Hanoi University of Science and Technology (HUST), Hanoi, Vietnam. He received a Ph.D from CentraleSupélec, Paris-Saclay University, France in 2021 (in collaboration with Nokia Bell Labs France). Before joining HUST, he was a postdoctoral fellow at LAAS-CNRS and University of Avignon, France. His research focuses on the optimization of resource management in next-generation communication networks.

**Francesco De Pellegrini** received the MSc 2000, and Ph.D. 2004, University of Padova, Italy, in Information Engineering. He is professor in networking and artificial intelligence at LIA, the Computer Science department of the University of Avignon. Before he was a researcher at Fondazione Bruno Kessler, Italy. He applies algorithms on graphs, stochastic control, and game theory for the design and the perfomance evaluation of networked systems.

**Balakrishna Prabhu** is a CNRS researcher at LAAS-CNRS, Toulouse, France. His research interests are in performance analysis of communication systems using stochastic modelling and game theory. He obtained his PhD from INRIA Sophia Antipolis (France) in 2005 and M.Sc (Engg.) from the IISc (India). Before joining LAAS-CNRS, he did postdoctoral stints at VTT (Finland), CWI, Eurandom and TU/e (The Netherlands).

**Cédric Richier** is a research engineer at CNRS and the Avignon University, Avignon, France. He was awarded his master's degree in 2012 from the Avignon University. He has worked on several diverse research projects such as social networks, multimedia, data centers and resource allocation.