DCoflow: Deadline-Aware Scheduling Algorithm for Coflows in Datacenter Networks

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

DCoflow: Deadline-Aware Scheduling Algorithm for Coflows in Datacenter Networks

Quang-Trung Luu*, Olivier Brun*, Rachid El-Azouzi†, Francesco De Pellegrini†,
Balakrishna J. Prabhu*, Cédric Richier†
*LAAS-CNRS, University of Toulouse, CNRS, 31400 Toulouse, France
†CERI/LIA, University of Avignon, 84029 Avignon, France
{qtluu, brun, bala}@laas.fr, {rachid.elazouzi, francesco.de-pellegrini, cedric.richier}@univ-avignon.fr

***Abstract*—Datacenter networks routinely support the data transfers of distributed computing frameworks in the form of coflows, i.e., sets of concurrent flows related to a common task. The vast majority of the literature has focused on the problem of scheduling coflows for completion time minimization, i.e., to maximize the average rate at which coflows are dispatched in the network fabric. Modern applications, though, may generate coflows dedicated to online services and mission-critical computing tasks which have to comply with specific completion deadlines. In this paper, we introduce `DCoflow`, a lightweight deadline-aware scheduler for time-critical coflows in datacenter networks. The algorithm combines an online joint admission control and scheduling logic and returns a σ-order schedule which maximizes the number of coflows that attain their deadlines. Extensive numerical results demonstrate that the proposed solution outperforms existing ones.**

***Index Terms*—Time-sensitive coflow scheduling, coflow admission control, σ-order, deadline.**

I. Introduction

Modern traffic engineering in datacenter networks is based on the notion of coflow originally defined in [1]. The interest for this traffic abstraction has originally been motivated by the need to capture the structure of the data exchanges occurring in distributed computing frameworks such as MapReduce or Spark [2, 3]. Such software frameworks rely on the so called *dataflow* computing model for large-scale data processing, i.e., a distributed computing paradigm, where each intermediate computation stage is distributed over a set of nodes and its output is transferred to nodes hosting the next stage. In between two computation stages, such dataflows are producing a set of network flows traversing the datacenter fabric and are abstracted as a *coflow*. A popular example of such data transfer is the *shuffle phase* of Hadoop MapReduce [2].

The customary performance metric for the data transfer phase is the makespan or the weighted coflow completion time (CCT). Minimizing the average CCT is an appropriate goal in order to increase the number of computing jobs dispatched per hour in a datacenter. The weighted CCT minimization has thus been addressed in several works, e.g., [1, 4–6]. A decade's research on the problem has shed light on its complexity and several algorithmic solutions have been devised. The problem was proven *NP*-hard and inapproximable below a factor of 2 by reduction to the job scheduling problem on multiple correlated machines. Near-optimal algorithms with 4-approximation performance bounds have been proposed in the literature [5–7]. However, the context changes radically in the case of time-sensitive jobs, where the data transfer phase may be subject to strict coflow deadlines.

Here, coflow scheduling is typically combined with admission control in order to reduce the number of violations, i.e., the number of coflows to complete after their deadlines. The resulting problem is the coflow deadline satisfaction (CDS) problem introduced in [8]. Each coflow is subject to a completion deadline and the target is to operate joint *coflow admission control and scheduling* in such a way to maximize the number of admitted coflows which respect their deadlines. This problem is *NP*-hard as well and it is proved inapproximable within any constant factor from the optimum [8].

Even though the problem has been identified quite early in the literature [9], with a few exceptions, the vast majority of works on coflow scheduling have not dealt with the problem of time-sensitive coflows. On the other hand, as confirmed later in our performance analysis, even near-optimal algorithms for CCT minimization may fail to respect the coflow deadline. In reality, the notion of time-sensitive coflows has become pervasive in the way how modern datacenters operate as distributed networks. In fact, not only computing frameworks are often tasked with time-sensitive jobs: modern web and mobile applications are implemented using microservice architectures, so that the users requests issued to an application may activate hundreds or thousands of services from as many servers to retrieve the users data. The last incoming bit of data, i.e., the CCT of this batch of flows, determines the lag to the service response, and large delays degrade the quality of experience.

In this paper, we address the problem of maximizing the Coflow Acceptance Rate (CAR), in which each coflow is subject to a completion time deadline. In principle, one can solve this problem by formulating a suitable Mixed Integer Linear Program (MILP). However, in datacenters with tens of thousands of coflows [4], techniques based on MILPs or their relaxations may be not viable. For designing scalable algorithms, the main idea appearing in many research works is to schedule coflows using a priority order of coflows. Once an ordering, denoted as σ, is determined, it is enough to adopt a work-conserving transmission policy. We focus on σ-order

schedulers since they offer a key implementation advantage: at the level of rate control, any work conserving preemptive dynamic rate allocation is allowed as long as it is compatible with the input coflow priority (the maximal performance loss within said rate allocation policies is bounded by a factor of 2 [6]). For instance, using fixed coflow priorities under DiffServ satisfies the definition of a σ-order scheduler. On the other hand, in order to avoid per-flow rate control, commercial switches have built-in priority queues and per-flow tagging can be used to prioritize active coflows. In principle, this permits to perform a greedy rate allocation which is compatible with a target σ-order. The exact mapping from a coflow σ-order to the switch priority queuing mechanism (and the inevitable limitations of legacy hardware therein) while an interesting subject, is out of the scope of the present paper.

Contributions. This paper proposes a lightweight method to perform coflow scheduling under deadlines. The proposed solution provably outperforms existing ones in the literature and does not rely on the solution of a linear program. In particular, we propose first a baseline offline admission control policy which is combined with a scheduler drawn in the class of σ-order coflow schedulers [6]. The output of the algorithm is an order of priority restricted to the set of admitted coflows. The algorithm is hence extended to perform joint admission control and scheduling in the online scenario, when coflows are generated at runtime at unknown release times. Through extensive numerical experiments on a wide variety of scenarios, we show that our algorithm systematically outperforms the ones currently available in the literature. These experiments are performed on both offline and online setting using both synthetic traces and real traces obtained from the Facebook data [9]. The main observation is that, under higher workloads (i.e., when the acceptance ratio is lower), our algorithm outperforms significantly the existing ones. Thus, it proves robust to workload variations as well to the type of data set used to generate the coflows.

The rest of the paper is organized as follows. Sec. II describes the general problem tackled in the paper and the coflow ordering models, whereas Sec. III describes the proposed algorithms. Numerical results are then provided in Sec. IV. In Sec. V, we describe the literature on deadline-aware coflow scheduling. Concluding remarks and future research directions are given in Sec. VI.

II. Problem Statement and Theoretical Analysis

In this section, we present the system model and formulate the acceptance rate maximization problem for a given input set of coflows. Table I summarizes the main notations used throughout the paper. The datacenter network (or datacenter fabric) is represented as a *Big-Switch* model [9], a non-blocking switch whose ingress (egress) port ℓ has capacity B_ℓ equal to the corresponding bandwidth inbound (outbound) capacity to connect servers to the top-of-rack (ToR) switch. Due to large bisection capacity and customary usage of load balancing, in fact, traffic congestion is typically observed only at the rack access ports leading to the ToR switches.

We consider a batch of N coflows $\mathcal{C} = \{1, 2, ..., N\}$. A coflow is a collection of flows, in which each flow represents a shuffle connection over a pair of fabric ingress-egress ports. Denote $\mathcal{F}_k$ as the set of flows of coflow $k \in \mathcal{C}$ and assume that the volume $v_{k,j}$ of each flow $j \in \mathcal{F}_k$ is known. For the sake of clarity, we suppose that all coflows arrive at the same time, i.e., their release time is zero. Also, each coflow k is subject to a completion deadline T_k. Similarly, we let $\mathcal{F}_{\ell,k}$ be the set of flows in $\mathcal{F}_k$ which uses port $\ell \in \mathcal{L}$ either as ingress port or as egress port. The total volume of data sent by coflow k on port ℓ is then given by $\hat{v}_{\ell,k} = \sum_{j \in \mathcal{F}_{\ell,k}} v_{k,j}$. Let $p_{\ell,k}$ be the transfer completion time in isolation, i.e., at full rate, of coflow k at port ℓ, which is given by $p_{\ell,k} = \hat{v}_{\ell,k}/B_\ell$. Let $r_{k,j}(t) \in \mathbb{R}_+$ be the rate allocated to flow $j \in \mathcal{F}_k$ at time t. The CCT of coflow k, denoted as c_k, thus writes

$$c_k = \max_{j \in \mathcal{F}_k} \frac{v_{k,j}}{\bar{r}_{k,j}}, \text{ where } \bar{r}_{k,j} = \frac{1}{CT_{k,j}} \int_0^{CT_{k,j}} r_{k,j}(t)\,\mathrm{d}t, \quad (1)$$

and $\bar{r}_{k,j}$ is the average rate of flow j through its lifetime and $CT_{k,j}$ is its completion time. This quantity rules the dispatching time for the volume traversing the so-called coflow bottleneck which determines the CCT of the said coflow. Hence a coflow k satisfies the deadline when the last flow of the coflow finishes before T_k, i.e., $c_k \leq T_k$.

Let $z_k \in \{0, 1\}$ be an indicator of whether coflow k finishes before T_k. Maximizing the number of coflows meeting their deadline corresponds to maximizing the sum of z_k subject to the constraint of bandwidth capacity at ingress and egress ports. The CAR maximization problem can be formulated as

$$\max_{r} \quad \sum_{k \in \mathcal{C}} z_k \quad (\text{P1})$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{C}} \sum_{j \in \mathcal{F}_{k,l}} r_{k,j}(t) \leq B_\ell, \quad \forall \ell \in \mathcal{L}, \forall t \in \mathcal{T}, \quad (2)$$

$$\int_0^{T_k} r_{k,j}(t)\,\mathrm{d}t \geq v_{k,j} z_k, \quad \forall j \in \mathcal{F}_k, \forall k \in \mathcal{C}, \quad (3)$$

Constraint (2) expresses that, at any instant t within the time horizon $\mathcal{T}$, the total rate that port ℓ assigns to flows cannot exceed its capacity B_ℓ. Constraint (3) ensures that the data of flows of each *accepted* coflow k should be completely transmitted before the deadline T_k.

Lemma 1 (Proposition 1 in [8])**.** *There exists a polynomial time reduction of the CAR problem* (P1) *to the problem of minimizing the number of late jobs in a concurrent open shop [10]. Hence, the CAR problem is NP-hard.*

A. Motivating Example

We will take `CS-MHA` [11] as our starting point to address the CAR problem. `CS-MHA` has introduced a new direction to solve the scheduling problem by means of a static coflow prioritisation. The prioritisation is used in order to approximate the solution of the coflow scheduling problem that maximizes the CAR. `CS-MHA` computes the scheduling orders at each port using the Moore-Hodgon's algorithm which also determines the set of admitted coflows at each port $\ell \in \mathcal{L}$. Since different ports could have different sets of admitted coflows, a coflow is

TABLE I
MAIN NOTATIONS.

Symbol	Description
$\mathcal{L}$	set of fabric ports
M	number of machines, $M = \|\mathcal{L}\|/2$
B_ℓ	available bandwidth of port $\ell \in \mathcal{L}$
$\mathcal{C}$	set of coflows. $\mathcal{C}$ has cardinality of N
σ	scheduling order of coflows, $\sigma = \{\sigma_1, \cdots, \sigma_{N-1}, \sigma_N\}$
T_k	deadline of coflow k
$\mathcal{F}_k$ ($\mathcal{F}_{\ell,k}$)	set of flows of coflow k (that use port ℓ)
$v_{k,j}$	volume of flow $j \in \mathcal{F}_k$ of coflow k
$\hat{v}_{\ell,k}$	total volume transmitted by coflow k on port ℓ
$p_{\ell,k}$	processing time of coflow k on port ℓ
c_k ($c_{\ell,k}$)	completion time of coflow k on port ℓ
$CT_{k,j}$	completion time of flow j of coflow k

admitted if it is admitted at all ports. For all rejected coflows, a second round is applied to check if some rejected coflows can actually satisfy their deadline. The algorithm selects a coflow with the minimum bandwidth required at the bottleneck port since it is more likely to catch up with its deadline.

The simple example depicted in Fig. 1 illustrates the limitations of `CS-MHA`. It uses the standard *Big-Switch* model to abstract a datacenter fabric: the example will be used as a running example throughout the paper. The instance contains 5 coflows: C_1 has 4 flows and C_2, C_3, C_4 and C_5 each have one flow. To ease the presentation, the flows are organised in virtual output queues at the ingress ports. The virtual queue index represents the flow output port, modulo the number of machines. The numbers on the flows' representations correspond to their *normalized* volumes. All fabric ports have the same *normalized* bandwidth of 1.

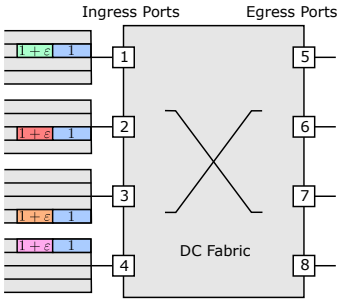


Fig. 1. Motivating example with a Big-Switch fabric composed of 4 ingress/egress ports connecting to 4 machines. Flows in ingress ports are organized by destinations and are color-coded by coflows. The example has 5 coflows. Coflow C_1 (blue) has 4 flows, with each ingress port sending 1 units of data to one egress port: its deadline is 1; coflows C_2 (green), C_3 (red), C_4 (orange) and C_5 (purple) have a single flow, each sending $(1+\varepsilon)$ unit of data. The deadline of these coflows is 2.

At the first iteration, `CS-MHA` computes the scheduling order on each port using Moore-Hodgson algorithm [12], which is based on the *Earliest Due Date* rule and is known to minimize the number of missed deadlines on a single machine (or *port* in the coflow context). Since coflow C_1 uses all ports and has the smallest deadline ($T_1 = 1$), then at each port, C_1 will be scheduled first. As consequence, all other coflows are rejected since they cannot satisfy their deadline when scheduled after C_1. Given that coflow scheduling, the CAR is $\frac{1}{5}$. However, an optimal scheduling solution would be $\{C_2, C_3, C_4, C_5, C_1\}$ or any combination that has coflow C_1 as the last one to be scheduled. The latter scheduler attains a CAR of $\frac{4}{5}$. From this example, we can see that `CS-MHA` can be made arbitrarily worse compared to the optimal solution.

The above example can easily be extended to show that in the worst-case scenario, the CAR of `CS-MHA` can be made arbitrarily small (as close to 0 as needed). Consider M machines, coflow C_1 that uses all ports, and coflows $C_2, \ldots, C_M$ with one flow each. The other parameters remain the same as above. It can be seen that the CAR obtained using `CS-MHA` and `DCoflow` are $\frac{1}{M}$, $\frac{M-1}{M}$, respectively. In this setting, the CAR obtained using `CS-MHA` is close to zero when M is high, while using `DCoflow`, the CAR is close to one.

The key observation is that `CS-MHA` neglects the impact that a coflow may have on other coflows on multiple ports. Indeed, a coflow causing multiple deadlines to be missed should have lesser priority, even when its deadline is the earliest. When this is neglected, the coflow ordering is misjudged thus degrading the CAR. In this work, we start from this observation and propose a new σ-order scheduler, called `DCoflow`.

B. Coflow Ordering

Problem (P1) implicitly depends on the CCT. An alternative approach to maximize the CAR is to *order* the coflows in some appropriate way, and then to leverage the priority forwarding mechanisms of the underlying transport network [13].

In this approach, once such an ordering σ is determined, it is enough to adopt work-conserving transmission policies and use forwarding priorities in such a way that a 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 $k' < k$, i.e., a flow of a coflow with higher priority according to σ. Such a flow scheduling is called *σ-order-preserving*.

The formulation of Problem (P1) can be transformed into an Integer Linear Program (ILP) as follows. We define the binary variable $\delta_{i,k}$ as 1 if coflow i has higher priority than coflow $k \neq i$, and 0 otherwise. The ordering of coflows can then be modeled using the following standard disjunctive constraints

$$\delta_{k,k'} + \delta_{k',k} = 1, \quad \forall k, k' \in \mathcal{C}, \tag{4}$$

$$\delta_{k,k'} + \delta_{k',k"} + \delta_{k",k} \leq 2, \quad \forall k, k', k" \in \mathcal{C}. \tag{5}$$

It should be clear that the ordering σ can easily be derived from the variables $\{\delta_{k,k'}\}_{k,k' \in \mathcal{C}}$. This ordering should be such that as many coflows as possible are accepted, that is, $\sum_{i \in \mathcal{C}} z_k$ is maximized. A central difficulty here is to compute the completion time of a coflow, which stems from the fact that data transmissions on the various ports are not independent: the transmission of a flow may be blocked until the ingress port becomes available even if the egress port is idle, and vice versa. However, assuming that the ports are independent, we can obtain a lower bound on the completion time of coflow k on port ℓ as follows

$$c_{\ell,k} \geq \sum_{k' \neq k} p_{\ell,k'} \delta_{k',k} z_{k'} + p_{\ell,k}, \quad \forall \ell \in \mathcal{L}, k \in \mathcal{C}. \tag{6}$$

This lower bound assumes that the transmission of coflow k on port ℓ can start as soon as all flows of all coflows k' scheduled before k have been transmitted on port ℓ. Note that the above constraint is not linear due to the product $\delta_{k',k} z_{k'}$. However, it can easily be linearized by introducing binary variables $y_{k',k}$ satisfying the constraints

$$y_{k',k} \leq z_{k'}; \quad y_{k',k} \leq \delta_{k',k}; \quad y_{k',k} \geq z_{k'} + \delta_{k',k} - 1, \quad (7)$$

(6) thus becomes

$$c_{\ell,k} \geq \sum_{k' \neq k} p_{\ell,k'} y_{k',k} + p_{\ell,k}, \quad \forall \ell \in \mathcal{L}, k \in \mathcal{C}. \quad (8)$$

A lower bound on the completion time of coflow k is then obtained as $c_k = \max_{\ell \in \mathcal{L}} c_{\ell,k}$, and this coflow can meet its deadline only if $c_k \leq T_k$. The deadline constraint can then be described as

$$c_{\ell,k} \leq T_k z_k, \quad \forall \ell \in \mathcal{L}, k \in \mathcal{C}. \quad (9)$$

We can thus obtain an upper bound on the number of accepted coflows by solving the following ILP,

$$\max \sum_{k \in \mathcal{C}} z_k, \quad \text{s.t. } (4, 5, 7, 8, 9). \quad \text{(P2)}$$

It is easy to verify that Problem (P2) is *NP*-hard and so that it may become unfeasible to find an optimal solution in large-scale datacenter networks. In the next section, we hence propose an efficient heuristic algorithm to determine the order in which coflows must finish in order to maximize the coflow acceptance rate.

III. σ-Order Scheduling with DCoflow

In this section, we present DCoflow, an algorithm to solve the problem of joint coflow admission control and scheduling. Given a list of N coflows, it provides a permutation $\sigma = (\sigma_1, \sigma_2, .., \sigma_N)$ of these coflows, with the aim of maximizing the coflow acceptance rate. We observe that, when a coflow is served according to the corresponding σ-order-preserving schedule, lower-priority coflows will be impacted. It is hence possible that some of these lower-priority coflows will not respect their deadline. Hence our objective is to evaluate the impact of a scheduled coflow with respect to the original CAR problem. The problem is combinatorial in nature, since there are $N!$ possible coflow orderings. Hence, we propose a new approach that uses the formulation of Problem (P2) by deriving a necessary condition satisfied by all feasible coflow orders solving Problem (P2).

Consider a feasible solution to Problem (P2) and a subset $\mathcal{S} \subseteq \mathcal{C}$ of coflows. Given a coflow $k \in \mathcal{C}$, let $\mathcal{S}_k^- = \{k' \in \mathcal{S} : \delta_{k',k} = 1\}$ be the set of coflows in $\mathcal{S}$ which are scheduled before k. Condition (6) may be rewritten as

$$c_{\ell,k} \geq p_{\ell,k} + \sum_{k' \in \mathcal{S}_k^-} p_{\ell,k'} z_{k'}, \quad (10)$$

which implies

$$c_{\ell,k} p_{\ell,k} z_k \geq p_{\ell,k}^2 z_k^2 + p_{\ell,k} z_k \sum_{k' \in \mathcal{S}_k^-} p_{\ell,k'} z_{k'}. \quad (11)$$

Using the inequality $T_k \geq c_{\ell,k} z_k$ and summing over all coflows $k \in \mathcal{S}$, we obtain

$$\begin{aligned}
\sum_{k \in \mathcal{S}} p_{\ell,k} T_k &\geq \sum_{k \in \mathcal{S}} c_{\ell,k} p_{\ell,k} z_k \geq \sum_{k \in \mathcal{S}} (p_{\ell,k} z_k)^2 + \sum_{k \in \mathcal{S}, k' \in \mathcal{S}_k^-} p_{\ell,k} p_{\ell,k'} z_k z_{k'} \\
&= \frac{1}{2} \sum_{k \in \mathcal{S}} (p_{\ell,k} z_k)^2 + \frac{1}{2} \left[\sum_{k \in \mathcal{S}} (p_{\ell,k} z_k)^2 + 2 \sum_{k \in \mathcal{S}} p_{\ell,k} z_k \sum_{k' \in \mathcal{S}_k^-} p_{\ell,k'} z_{k'} \right] \\
&= \frac{1}{2} \sum_{k \in \mathcal{S}} (p_{\ell,k} z_k)^2 + \frac{1}{2} \left(\sum_{k \in \mathcal{S}} p_{\ell,k} z_k \right)^2. \quad (12)
\end{aligned}$$

We thus conclude that any feasible solution to Problem (P2) satisfies the condition $\sum_{k \in \mathcal{S}} p_{\ell,k} T_k \geq f_\ell(\mathcal{S})$ for any subset $\mathcal{S} \subseteq \mathcal{C}$ of *accepted* coflows, where $f_\ell(\mathcal{S}) = \frac{1}{2} \sum_{k \in \mathcal{S}} p_{\ell,k}^2 + \frac{1}{2} \left(\sum_{k \in \mathcal{S}} p_{\ell,k} \right)^2$. These conditions are the so-called *parallel inequalities* and provide valid inequalities for the concurrent open shop problem [14]. Note that they do not depend on the ordering of the coflows.

We shall use the parallel inequalities to determine the coflows that should not be admitted. Consider a solution to Problem (P2) and assume that, in this solution, there exists a subset $\mathcal{S}$ of accepted coflows (i.e., $z_k = 1$ for all $k \in \mathcal{S}$) such that $\sum_{k \in \mathcal{S}} p_{\ell,k} T_k < f_\ell(\mathcal{S})$ for at least one port $\ell \in \mathcal{L}$. This implies that this solution is not feasible and can only become feasible by rejecting a coflow k' among the accepted coflows using port ℓ. We choose this coflow k' so as to minimize the quantity $f_\ell(\mathcal{S} \setminus \{k'\}) - \sum_{k \in \mathcal{S} \setminus \{k'\}} p_{\ell,k} T_k$, in the hope that it becomes negative. Observe that

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

from which it follows that

$$\begin{aligned}
f_\ell(\mathcal{S} \setminus \{k'\}) - \sum_{k \in \mathcal{S} \setminus \{k'\}} p_{\ell,k} T_k &= f_\ell(\mathcal{S}) - \sum_{k \in \mathcal{S}} p_{\ell,k} T_k \\
&\quad + p_{\ell,k'} \left(T_{k'} - \sum_{k \in \mathcal{S}} p_{\ell,k} \right) \quad (14)
\end{aligned}$$

The above relation will be used to define our coflow admission control algorithm. In general, it suggests possible heuristics to remove a coflow k' from $\mathcal{S}$ in order to satisfy the parallel inequalities. For instance, one such heuristics is to minimize the quantity

$$\Psi_{\ell,k'} := p_{\ell,k'} \left(T_{k'} - \sum_{k \in \mathcal{S}} p_{\ell,k} \right). \quad (15)$$

In fact, the term $\sum_{k \in \mathcal{S}} p_{\ell,k} - T_{k'}$ represents how much the completion time of coflow k' on port ℓ exceeds its deadline, assuming that it is scheduled latest. Hence, we could reject coflow k' with both large processing time $p_{\ell,k'}$ on port ℓ and large deadline violation. More admission rationales based on (14) will be detailed in describing DCoflow, whose pseudocode is reported in Algorithm 1. Generally, DCoflow takes a list of unsorted coflows and provides as output the

Algorithm 1: `DCoflow`

1 $\mathcal{S} = \{1, 2, \ldots, N\}$; ▷ initial set of unscheduled coflows
2 $\sigma = \varnothing$; ▷ initial scheduling order
3 $\sigma^\star = \varnothing$; ▷ initial set of pre-rejected coflows
4 $n = N$; ▷ round counter
5 **while** $\mathcal{S} \neq \varnothing$ **do**
6 $\quad \ell_b = \arg\max_{\ell \in \mathcal{L}} \sum_{k \in \mathcal{S}} p_{\ell,k}$; ▷ bottleneck port
7 $\quad \mathcal{S}_{\ell_b} = \{k \in \mathcal{S} : p_{\ell_b,k} > 0\}$; ▷ set of coflows using ℓ_b
8 $\quad$ *# Coflows that can finish in time when scheduled last*
9 $\quad \mathcal{S}^d_{\ell_b} \leftarrow \{j \in \mathcal{S}_{\ell_b} | \sum_{k \in \mathcal{S}_{\ell_b}} p_{\ell_b,k} \leq T_j\}$;
10 $\quad$ **if** $\mathcal{S}^d_{\ell_b} \neq \varnothing$ **then**
11 $\quad\quad k_n = \arg\max_{k \in \mathcal{C}} T_k$; ▷ admit coflow with largest deadline
12 $\quad\quad \sigma_n = k_n$; ▷ append k_n to σ
13 $\quad\quad \mathcal{S} = \mathcal{S} \setminus \{k_n\}$; ▷ remove k_n from $\mathcal{S}$
14 $\quad$ **else**
15 $\quad\quad k^\star = \texttt{RejectedCoflow}(\mathcal{S})$; ▷ select a coflow to reject
16 $\quad\quad \sigma_n = k^\star$; ▷ append $k^\star$ to σ;
17 $\quad\quad \sigma^\star = \sigma^\star \cup \{k^\star\}$; ▷ append $k^\star$ to $\sigma^\star$
18 $\quad\quad \mathcal{S} = \mathcal{S} \setminus \{k^\star\}$; ▷ remove $k^\star$ from $\mathcal{S}$
19 $\quad n = n - 1$; ▷ update the round index
20 $\sigma = \texttt{RemoveLateCoflows}(\sigma, \sigma^\star)$;
21 **return** σ; ▷ final scheduling order
22 **Function** `RemoveLateCoflows(`σ, $\sigma^\star$`):`
23 $\quad$ **while** $\sigma^\star \neq \varnothing$ **do**
24 $\quad\quad k^\star \leftarrow \arg\min_k \{\sigma_k \in \sigma^\star\}$; ▷ first coflow in $\sigma^\star$
25 $\quad\quad c_{\sigma_{k^\star}} \leftarrow \texttt{evalCCT}\left(\{\sigma_k\}_{k \in [1,k^\star]}\right)$; ▷ CCT of coflow $\sigma_{k^\star}$
26 $\quad\quad$ **if** $c_{\sigma_{k^\star}} > T_{\sigma_{k^\star}}$ **then**
27 $\quad\quad\quad \sigma \leftarrow \sigma \setminus \{\sigma_{k^\star}\}$;
28 $\quad\quad \sigma^\star \leftarrow \sigma^\star \setminus \{\sigma_{k^\star}\}$;
29 $\quad$ **return** σ;

scheduling order of accepted coflows. It works in rounds and at each round, it either accepts a coflow or it rejects one.

`DCoflow` starts by computing the total completion time at each port and finds the bottleneck ℓ_b, i.e., the port with the largest completion time. Having this, it determines $\mathcal{S}^d_{\ell_b}$, the set of coflows active on the bottleneck port ℓ_b (line 8) which complete before their deadline if scheduled last on ℓ_b. If $\mathcal{S}^d_{\ell_b}$ is not empty (Lines 10–13), it selects a coflow $k_n \in \mathcal{S}^d_{\ell_b}$ that has the largest deadline in $\mathcal{S}^d_{\ell_b}$. If $\mathcal{S}^d_{\ell_b}$ is empty (Lines 14–18), i.e., no coflow respects its deadline when scheduled last on the bottleneck (Line 13), the algorithm selects a coflow to be removed using function `RejectedCoflow`, whose aim is to comply with parallel inequalities according to (14). We consider two variants, each of which corresponds to a slightly different criterion to select the candidate coflow $k^\star$ by taking into account its weight.

The first variant, namely `DCoflow_v1`, finds the candidate $k^\star \in \mathcal{S}_b$ and every port ℓ used by k where it fails to meet the deadline (that is, $\Psi_{\ell,k} < 0$),

$$k^\star = \arg\min_{k \in \mathcal{S}_b} \Big(\sum_{\ell : \Psi_{\ell,k} < 0} \Psi_{\ell,k} \Big). \tag{16}$$

The second variant, namely `DCoflow_v2`, finds the candidate $k^\star \in \mathcal{S}_b$ and every port ℓ used by k that has at least γ times the congested level of the bottleneck, *i.e.*, $\ell : \sum_j p_{\ell j} \geq \gamma \sum_j p_{bj}$,

$$k^\star = \arg\min_{k \in \mathcal{S}_b} \Big(\sum_{\ell : \sum_j p_{\ell j} \geq \gamma \sum_j p_{bj}} \Psi_{\ell k} \Big). \tag{17}$$

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

Unscheduled coflows (set $\mathcal{S}$)	ℓ_b	$\{\overline{\Psi}_1, \overline{\Psi}_2, \overline{\Psi}_3, \overline{\Psi}_4, \overline{\Psi}_5\}$
$\mathcal{S} = \{\boldsymbol{C_1}, \boldsymbol{C_2}, C_3, C_4, C_5\}$	1	$\{-4(1+\varepsilon), -\varepsilon, \cdot, \cdot, \cdot\}$
$\mathcal{S} = \{\boldsymbol{C_2}, C_3, C_4, C_5\}$	1	$\{\cdot, 0, \cdot, \cdot, \cdot\}$
$\mathcal{S} = \{\boldsymbol{C_3}, C_4, C_5\}$	2	$\{\cdot, \cdot, 0, \cdot, \cdot\}$
$\mathcal{S} = \{\boldsymbol{C_4}, C_5\}$	3	$\{\cdot, \cdot, \cdot, 0, \cdot\}$
$\mathcal{S} = \{\boldsymbol{C_5}\}$	4	$\{\cdot, \cdot, \cdot, \cdot, 0\}$

Once the initial scheduling order σ is obtained, `DCoflow` uses the function `RemoveLateCoflows` to estimate the CCT and remove from σ the coflows that do not satisfy the deadline constraint (Line 20). Briefly, `RemoveLateCoflows` considers each pre-rejected coflow $k^\star$ in $\sigma^\star$ (Line 24). It calls the function `evalCCT` to evaluate the CCT of $k^\star$, given all the coflows in σ that are scheduled before $k^\star$ (Line 25). If $k^\star$ cannot meet its deadline, i.e., $c_{\sigma_{k^\star}} > T_{\sigma_{k^\star}}$, it will be removed permanently from $\sigma^\star$ and σ. `RemoveLateCoflows` performs a swipe on the admitted coflows (as in [11]) and iteratively removes coflows in σ that belong to $\sigma^\star$ until the estimated CCT c_{σ_k} of each coflow $\sigma_k \in \sigma$ is of at most T_{σ_k}.

It is important to note that the final solution yielded by `DCoflow` does not guarantee all coflows in σ to satisfy their deadlines. In Sec. IV, the prediction error of `DCoflow` indicates the gap between the estimated CAR and the actual CAR after applying resource allocation.

Example. To illustrate the difference between `DCoflow` and `CS-MHA`, we consider again the example illustrated in Fig. 1. Table II shows the execution of `DCoflow_v1` on that example. At the first step, `DCoflow_v1` chooses bottleneck ingress port 1, which is used by coflows C_1 and C_2. It computes $\overline{\Psi}_k = \sum_{\ell : \Psi_{\ell,k} < 0} \Psi_{\ell,k}$ for both coflows and selects the coflow with smallest $\overline{\Psi}_k$ (in this case, C_1) to be scheduled last. Since all unscheduled coflows do not share any port in the fabric, any ordering of remaining coflows gives the same average CAR. Given the final schedule, `DCoflow` obtains $\frac{4}{5}$ as the average CAR, which is the optimal solution.

Online Implementation. `DCoflow` can be also be run online, when coflows arrive sequentially and possibly in batches. For this, define f to be the frequency of updates, i.e., instants at which `DCoflow` recomputes a schedule. The updates can be performed either at arrival instants of coflows (in which case we set $f = \infty$) or periodically with period $1/f$. We assume that the scheduler knows the volumes of the flows of each arrived coflow. However, it neither knows the volumes nor the release times of future coflows.

At each update instant, `DCoflow` recomputes the scheduling order for coflows currently available in the network. These include the ones that were scheduled in the previous scheduling instants and have not yet finished; the ones that were rejected in the previous scheduling instants but whose deadline has not yet expired and the arrivals during the update interval. `DCoflow` calculates the new order for this set of coflows based on the remaining volumes of the flows and not on the original volumes. Note that it is assumed that coflows can be

preempted [4]. This process is repeated at each update instant.

Complexity Analysis. The complexity of DCoflow is $\mathcal{O}(N^2)$. Specifically, in DCoflow, the values $\sum_{i\in\mathcal{S}} p_{\ell,i}$ and $\Psi_{\ell,x}$ of each remaining coflow x can be pre-stored by calculating them at a 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 at Lines 11–13 is $\mathcal{O}(N)$ and at Lines 15–16 is $\mathcal{O}(N)$, so that finally across iterations it adds to $\mathcal{O}(N^2)$. On the other hand, it is easy to verify the complexity of RemoveLateCoflows is $\mathcal{O}(NL)$.

IV. Performance Evaluation

We evaluate via simulations our proposed heuristics (two variants of DCoflow[1]) along with some existing algorithms such as CS-MHA[2] and the solution provided by the optimization method CDS-LP proposed in [8]. The relaxed version of CDS-LP, named CDS-LPA, is also implemented[3]. By using the solution derived from CDS-LP as an upper bound, we would get the sense of how close the algorithms are to the optimum. A brief description of the reference algorithms has been given in Sec. I. We also compare the performance of our schedulers against Sincronia [13] and Varys [9] that aim to minimize the average CCT.

Once we obtained the σ-order, the actual coflow resource allocation for our solution is implemented by the greedy rate allocation algorithm GreedyFlowScheduling [13]. At any given point of time, GreedyFlowScheduling reserves the full port bandwidth to one flow at the time. It does so by complying to the scheduling order in σ of the coflow to which the flow belongs [13]. We note that, in the case of CDS-LP, CDS-LPA, and Varys, instead, the rate allocation is part of the algorithmic solution.

The network fabric is represented by M machines or end-hosts connected to a non-blocking Big-Switch fabric, of which each access port has a *normalized* capacity of 1. The algorithms will be evaluated on both small-scale and large-scale networks, where a network is denoted by $[M, N]$ to indicate different fabric size and number of coflows (N) used in the simulations. Small-scale networks have a fabric of size $M = 10$, whereas large-scale networks have a fabric with either 50 or 100 machines. The coflows in these networks are generated from both synthetic and real traffic traces.

The MILP solver gurobi is used to solved CDS-LP and CDS-LPA. Due to the high complexity, CDS-LP and CDS-LPA are only evaluated on small-scale networks. In what follows, the detailed setup, comparison metrics, and simulation results are presented.

[1]The flow-level simulator and the implementation of all algorithms tested in this paper are available at https://github.com/luuquangtrung/CoflowSimulator.

[2]We only reimplemented the centralized algorithm (CS-MHA) presented in [11], which has been reported to be better than the decentralized version (D^2-CAS) in terms of CAR.

[3]It is worth noting that the formulation of CDS-LP and CDS-LPA use the same decision variables $\{z_k\}_{k\in\mathcal{C}}$ as those introduced in Problem (P1). In CDS-LP, z_k are binaries, whereas in CDS-LPA, z_k are continuous variables taking values in $[0, 1]$. For any solution yielded by CDS-LPA, only coflows k whose z_k strictly equals 1 are considered as accepted ones.

A. Simulation Setup

Synthetic Traffic. The synthetic traffic comprises two types of coflows. Type-1 coflows have only one flow, whereas the number of flows of Type-2 coflows follows 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. Moreover, each coflow k is assigned a random deadline within $[\text{CCT}_k^0, 2\text{CCT}_k^0]$), where CCT_k^0 is the completion time of coflow k *in isolation*. Flows of Class-1 coflows are assigned a random volume of mean of 1 and standard deviation of 0.2. The volume ratio for the flows of Class-1 and Class-2 coflows is 0.8.

Real Traffic. Real traffic datasets are obtained by the Facebook traces dataset [9], based on a MapReduce shuffle trace collected from one of Facebook's 3000-machine cluster with 150 racks. The data traces consist of 526 coflows. It has a skewed coflow width distribution, ranging from coflows with a single flow to very large ones (the largest coflow has 21170 flows). For detailed statistics of the Facebook traces we refer the reader to [15]. For each configuration $[M, N]$, N coflows are randomly sampled from the Facebook dataset. Coflows are only selected from the ones that have at most M flows. The volume of each flow is given by the dataset.

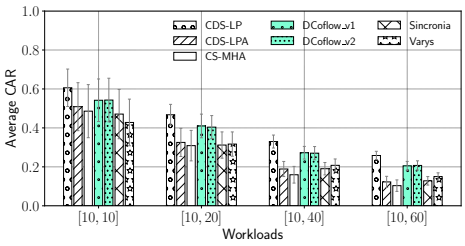
Metric. We evaluate the algorithms based on the average CAR. We also present the gains in percentiles of each algorithm with respect to the solution yielded by CDS-LP in terms of CAR. These gains are calculated using the formula: average gain in CAR $= \frac{\text{compared CAR}}{\text{CAR under CDS-LP}} - 1$.

B. Results with 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.

1) 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 $[10, 60]$.

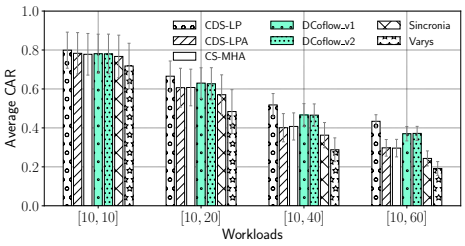
It is observed that our proposed heuristics are closest in terms of CAR to the optimum (CDS-LP) than all other algorithms, with both small- and large-scale networks. Among two variants of DCoflow, DCoflow_v1 yields the best performance and even outperforms CDS-LPA, the approximation version of CDS-LP. For instance, on the network $[10, 10]$, DCoflow_v1 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. Interestingly, the improvement becomes higher when the load is increased. For example, the corresponding improvement on average CAR on a $[10, 60]$ network are 67.2%, 98.3%, 59.9%, and 36.8% (see Fig. 2a). The improvement is even higher when performed on a large-scale netork. For example, compared to CS-MHA, Sincronia,



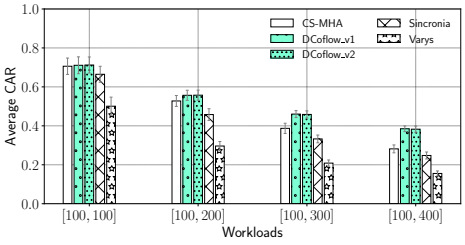
(a) Synthetic traffic traces on a small-scale network.

(b) Synthetic traffic traces on a large-scale network.

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 $[M, N]$.



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



(b) Facebook traffic traces on a large-scale network.

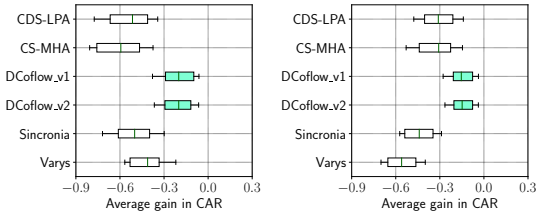
Fig. 3. Average CAR with Facebook traces using (a) small-scale network and (b) large-scale network. Each point in the x-axis represents network $[M, N]$.

and `Varys`, on the network $[100, 400]$, the improvement in terms of average CAR are respectively 648.1%, 32.3%, and 17.9%. (see Fig. 2b). It is worth noting how the performance of `CS-MHA` falls drastically when dealing with large-scale networks. This is expected since `CS-MHA` computes prioritizes coflows that use a large number of ports over those that use a few. In instances with a large number of coflows of the latter type, the CAR of `CS-MHA` goes to 0 (see detailed explanation with the motivating example in Sec. II-A)

The result in Fig. 4a shows that the two variants of `DCoflow` achieve a smaller gap to the optimal in almost all values of percentile compared to other algorithms. For instance, compared to `Sincronia`, `DCoflow_v1` improves the CAR in 50% of 100 instances by 50% and it achieves around 43% at 99^{th} percentile.

2) 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-B1. 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 $[10, 60]$ fabric are shown in Fig. 4b. Similar to what observed in the results with the synthetic traces (see Sec. IV-B1), `DCoflow_v1` and `DCoflow_v2` yield a significant improvement in terms of average CAR compared to other heuristics. For instance, with a $[10, 60]$ configuration, the two variants of `DCoflow` improve the average CAR on average by 24.4%, 25%, 52.2%, 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-scale network. For example, compared to `CS-MHA`, `Sincronia`, and `Varys`, on $[100, 400]$ 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 the two variants of `DCoflow` achieve a smaller gap to the optimal in almost



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

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

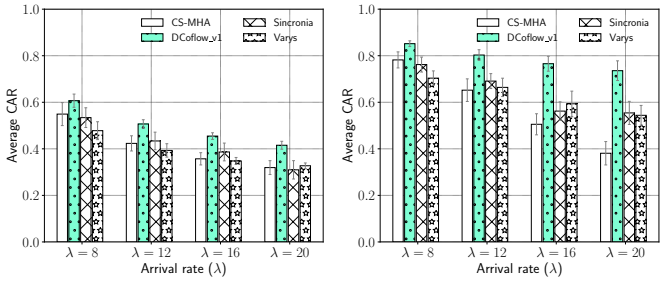
all values of percentile compared to the other algorithms. For instance, compared to `Sincronia`, `DCoflow_v1` improves the CAR in 57% of 100 instances by 50% and it achieves around 35% at 99^{th} percentile.

3) Prediction Error of `DCoflow`*:* As mentioned at the end of Sec. III, we also evaluate the prediction error $(|\sigma| - |\hat{\sigma}|)/|\sigma|$ of our heuristics, where $\hat{\sigma} \subseteq \sigma$ is the set of coflows in σ that satisfy the deadline constraint after performing the actual resource allocation using `GreedyFlowScheduling`. Both variants of `DCoflow` provide a prediction of CAR with an average error below 3.6% for both traffic traces.

C. Online Setting

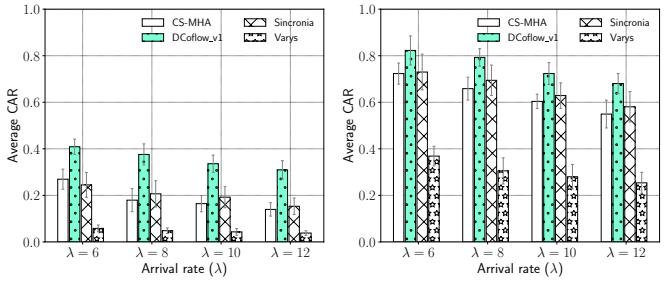
We now present a series of numerical results on the performance of the online version of `DCoflow`. The metric used for the performance evaluation is the average CAR obtained over 40 instances. In each instance, coflows arrive sequentially according to a Poisson process of rate λ, i.e., the inter-arrival time of coflows is exponentially distributed with rate λ. Unless stated otherwise, coflow priorities are computed upon arrival of a new coflow ($f = \infty$).

As both versions of `DCoflow` provide similar results, we only present the results obtained with `DCoflow_v1`. The



(a) Small fabric. (b) Large fabric.

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



(a) Small fabric. (b) Large fabric.

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

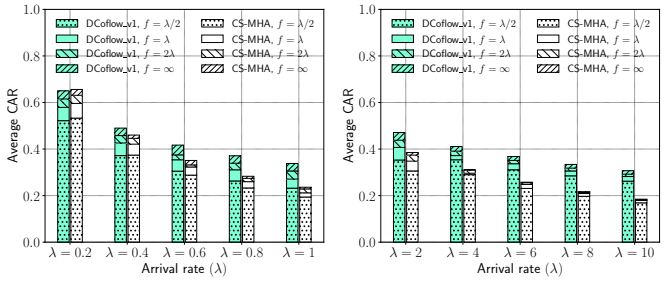
average CAR obtained with `DCoflow_v1` is compared against those obtained with the online version of Varys with deadline [16], `CS-MHA` and `Sincronia`. We investigate the effect of two main parameters: (*i*) the coflow arrival rate λ and (*ii*) the frequency f at which coflow priorities are updated.

1) Impact of Arrival Rate: We first study the impact of the arrival rate λ on the CAR obtained with the various algorithms. The CAR is averaged over 40 instances, each one with 4000 coflow arrivals. The deadline of a coflow k is drawn from a uniform distribution in $[\text{CCT}_k^0, 4\text{CCT}_k^0]$. We also consider two scenarios: a small fabric with $M = 10$ machines, and a large fabric with $M = 50$ machines. For each scenario, results are presented for the following values of λ: $\lambda = 8$, $\lambda = 12$, $\lambda = 16$, and $\lambda = 20$.

Our results are shown in Figs. 5a and 5b, for the small fabric scenario and the large one, respectively. We observe that `DCoflow_v1` achieves a higher average CAR for all values of λ, and that the gain with respect to the other scheduling algorithms increases with the value of λ. If all algorithms achieve more or less the same CAR for a lightly loaded fabric, `DCoflow_v1` clearly outperforms the other algorithms when the fabric is highly congested.

Figs. 6a and 6b show respectively the average CAR obtained with $M = 10$ and $M = 100$, both with 4000 coflows, using the Facebook dataset. Similar to what obtained with the synthetic traffic traces, `DCoflow_v1` outperforms all other methods with significant gains. When the fabric is highly congested (i.e., with $M = 10$), again `DCoflow_v1` yields a higher gain compared to other algorithms. For instance, `DCoflow_v1` achieves 9.3% higher CAR than `Sincronia` when $M = 100$ (see Fig. 6b), while with $M = 10$, the gap becomes 16.4% (see Fig. 6a).

2) Impact of Update Frequency: We now evaluate the impact of the update frequency f on the average CAR. We consider the following values of f: $f = \frac{\lambda}{2}$, $f = \lambda$, $f = 2\lambda$, and $f = \infty$. Recall that $f = \infty$ means that priorities are updated upon each coflow arrival. We assume that $M = 10$ and compute the CAR by averaging over 40 instances. For each instance, we simulate $8,000$ coflow arrivals according to a Poisson process at rate λ, assuming that the deadline of a coflow k is uniformly distributed in $[\text{CCT}_k^0, 2\text{CCT}_k^0]$. We present the average CAR obtained for different values of f ($f \in \{\frac{\lambda}{2}, \lambda, 2\lambda, \infty\}$) and for different values of the arrival rate λ (from 2 to 10).



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

Fig. 7. Average CAR of `DCoflow_v1` and `CS-MHA` using synthetic traffic with $[10, 8000]$ and varying λ, when obtaining (a) one single coflow per arrival; and (b) a random batch of coflow per arrival.

The results in Fig. 7a are obtained from a simulation, in which each arrival corresponds to one single coflow. We note, as before, that for a low arrival rate, both algorithms provide a similar average CAR (for $\lambda = 2$, `CS-MHA` achieves a slightly higher CAR than `DCoflow_v1`), but that `DCoflow_v1` clearly outperforms `CS-MHA` when the fabric is highly congested. We also note that a higher frequency f significantly improves the CAR for both algorithms. For instance, for $\lambda = 2$ (resp. $\lambda = 10$), the average CAR is increased by 52% (resp. 46%) if we update coflow priorities upon arrival of each new coflow instead of using the periodic scheme with $f = \frac{\lambda}{2}$. These results suggest that there is a need for a trade-off between the computational complexity of updating coflow priorities at a high frequency and the CAR achieved. In Fig. 7b, we present a similar result, but assuming that coflows arrive in batches. The batch size follows a uniform distribution $\mathcal{U}([5, 15])$. Since the average number of coflows in each batch is 10, in this setting, we divide the batch arrival rate by 10 to obtain the same coflow arrival rates as in Fig. 7a. The results obtained for batch arrivals are similar to those obtained previously, but we note that the gains of `DCoflow_v1` with respect to `CS-MHA` are significantly higher in this case. Moreover, we note that the benefit of using a higher update frequency is lower in this case (e.g., for $\lambda = 10$, the average CAR is increased only by 17% if we use $f = \infty$ instead of $f = \frac{\lambda}{2}$).

V. Related Work

As discussed in Sec. I, most works in the literature focus on CCT minimization, and deadline scheduling has received comparatively less attention. `Varys` [9] was one of the first algorithms for deadline-sensitive coflow scheduling. It uses a cascade of coflow admission control and scheduling. The scheduler strives for CCT minimization combining (*i*) a coflow ordering heuristic based on the per coflows bottleneck's completion time; and (*ii*) an allocation algorithm to assign bandwidth to individual flows of each coflow. Rate allocation in `Varys` is performed to approximately align the completion time of all coflows to the bottleneck one.

`Chronos` [16] is a heuristic for deadline scheduling which avoids starvation for flows that do not meet their deadlines by granting them the residual bandwidth. A priority order is determined first, and coflows are hence allocated the minimum necessary bandwidth to meet their individual deadlines. Once all the flows that meet their deadlines have been allocated bandwidth, the residual bandwidth is shared by the remaining coflows in proportion to their demands.

In [11], a connection between deadline scheduling of coflows and the well-known problem of minimizing late jobs in a concurrent open shop—a known *NP*-hard problem, is made. A heuristic based on the Moore-Hodgson's algorithm [12] for a single link is proposed. Both centralized as well as decentralized heuristics are introduced (namely `CS-MHA` and `D`^{2}`-CAS`, respectively).

A formal description of the deadline scheduling problem including bandwidth allocation of flows was given in [8]. The CDS maximization problem is formulated as an MILP (called `CDS-LP`). Time is divided into intervals whose boundaries are the coflow deadlines arranged in increasing order. The program determines which coflows to accept and the amount of bandwidth to allocate in each interval. `CDS-LP` is shown to be *NP*-hard, and an approximation based on LP relaxation (called `CDS-LPA`) of the binary variables is also proposed. `CDS-LPA` only accepts coflows for which the relaxed variable is strictly equal to 1, i.e., only coflows that are completely accepted by the LP relaxation are retained.

For completeness, we also cite a few works considering the minimization of CCT [4–6, 17, 18] as well as the survey article [19]. Popular among CCT minimization algorithms is `Sincronia` proposed in [13]. It considers scheduling on the network bottlenecks and returns a scheduling coflow order achieving a 4–approximation factor.

VI. Conclusion and Future Work

In this paper, we introduced a new joint coflow admission control and scheduling algorithm for a batch of coflows with deadlines. The proposed schemes leverage results from open-shop scheduling to determine a subset of coflows to schedule and a corresponding σ-order which is then employed in order to schedule coflows in priority.

Numerical results show that on small-scale networks, our algorithms perform similar to or better than other deadline-sensitive algorithms proposed in the literature. On large-scale networks, however, it shows significant improvements with respect to existing algorithms, e.g., 98% higher CAR than `CS-MHA` in an offline setting. Our scheme also has a low prediction error: even though the admission control is performed using a bottleneck approximation for the CCT, almost all accepted coflows actually finish within their deadline when being actually scheduled.

This behaviour is observed in both offline and online settings, both with synthetic traces and for real traces from the Facebook data set. This shows that the algorithm is robust with respect to the coflow size distribution and performs very well across a wide range of network sizes.

Several extensions of this research line are possible. In future works, we shall study the performance of our online solution in the case when coflows tend to be released in batches, e.g., when a distributed framework polls worker nodes with given period. Furthermore, a relevant case is that of incomplete information on flow volumes, i.e., because the volume of a flow is not directly available to the scheduler but only, for instance, via some *a priori* distribution. Finally, issues of starvation and fairness issue among coflows represent interesting issues we have not addressed yet.

References

[1] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. ACM HotNets*, Redmond, Washington, 2012, p. 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] N. M. K. Chowdhury, "Coflow: A networking abstraction for distributed data-parallel applications," Ph.D. dissertation, University of California, Berkeley, 2015.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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.

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

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

[19] 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.