Semi-Distributed Coflow Scheduling in Datacenters

Rachid El-Azouzi, Francesco De Pellegrini, Afaf Arfaoui, Cedric Richier, Jeremie Leguay, Quang-Trung Luu, Youcef Magnouche, Sebastien Martin

Semi-distributed Coflow Scheduling in Datacenters

Rachid El-Azouzi, Francesco De Pellegrini, Afaf Arfaoui, Cédric Richier, Jeremie Leguay, Quang-Trung Luu, Youcef Magnouche, and Sebastien Martin

***Abstract*—With the advent of big data applications, coflow scheduling has become a cornerstone for the engineering of traffic in datacenters. Minimizing the average weighted Coflow Completion Times (CCT) is a crucial step to minimize the execution time of jobs running in distributed computing frameworks.**

In this paper, we present a new σ-order coflow scheduling solution, ONE-PARIS, an online semi-clairvoyant and semi-distributed implementation suitable to minimize the weighted CCT in production environments. We achieves this through ONE-PARIS scheduler for ordering coflows and a decentralized resource allocation mechanism, called *Sync-Rate*, enabling to respect the order of priority of coflows provided by ONE-PARIS and ensuring efficient synchronization between flows of the same coflow in order to free up bandwidth for low-priority flows. Extensive simulations on both synthetic and real traffics show that our proposed coflow scheduler outperforms other state-of-art schemes.

***Index Terms*—Coflow scheduling, σ-order, distributed rate allocation, pricing, task scheduling, resource allocation.**

I. Introduction

Most cloud providers nowadays feature the provisioning of cluster computing as a service. Customers can launch their compute-intensive tasks on big data frameworks such as MapReduce [1] or Spark [2]. Such software frameworks rely on the so called *dataflow* computing model for large-scale data processing. It consists in 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, these dataflows are producing a set of flows, called a *coflow* [3], that are bound together by the same application task. Coflows represent a standard traffic pattern abstraction in datacenters. In MapReduce for instance, a coflow is a set of concurrent flows sent from *mapper* nodes, i.e., senders, to a set of *reducer* nodes, i.e., receivers. Such flows are launched after the mappers have completed their computing tasks. The data transfer phase between the mappers and reducers is called the *shuffle phase* and completes only when all constituent flows are over.

Research on the scheduling of coflows has started with the seminal work in [3], [4], proving that accounting for the coflow structure in traffic management significantly improves application-level performance. Since then, coflow scheduling has become a core topic in datacenter traffic engineering. The main challenge lies in the fact that computing frameworks can generate simultaneously tenths of thousands of flows per job [4]. When many jobs run in parallel, network congestion occurs due to concurrent coflows.

However, datacenter fabrics are largely over-provisioned with respect to bisection capacity [5] in order to cope with congestion events. Modern leaf-spine architectures enable load balancing with ECMP to prevent inner network congestion [6]. Nevertheless, congestion can still happen at the edge and a careful scheduling of application traffic is needed. The standard abstraction for resource allocation used in this context is hence based on the *Big Switch* model [7] (e.g., see Fig. 1), capturing the fact that congestion occurs at Top-of-Rack (ToR) switches only. Coflow scheduling performs the bandwidth allocation at ToR switch ports in the datacenter fabric.

The acceleration of big data frameworks is a wide research topic and various technologies have been proposed for packet scheduling, congestion control or load balancing [8]. However, it has been observed in real traces that coflow scheduling has a significant impact on the completion time of applications [9]. On average, the shuffle phase accounts for 33% of the running time in observed jobs. In 26% of the jobs with reduce tasks, it accounts for more than 50%. The reference objective function to measure acceleration at network layer is the weighted Coflow Completion Time (CCT) [4], [10].

One popular idea appearing in many research works suggests to equalize flow transfer times per coflow [9], i.e., to let all flows of a coflow finish at the same time. In fact, finishing some flows before the bottlenecked one is irrelevant w.r.t. the CCT of a coflow. In standard flow scheduling, shortest-flow-first heuristics grant average flow service time minimization [11]. Varys [7] is a baseline reference for clairvoyant heuristics. Even though recent scheduling algorithms like [12], [13] outperform it, Varys has introduced several key concepts at once. First, it combines shortest-flow-first, with coflow equalization. Furthermore, it works based on the notion of a *bottleneck* link of a coflow, i.e., the port of the fabric which experiences the maximal data transfer time. The schedule is performed using a *priority order*: the priority of coflows is assigned *dynamically* and given to the coflow that would end the soonest in isolation (i.e, if it was alone in the network). Hence, its traffic on the bottleneck link is served in priority, possibly pre-empting lower-priority coflows.

In general, the coflow scheduling problem to minimize CCT is strongly *NP*-hard and exact solutions based on time-indexed MILPs (Mixed Integer Linear Programs) suffer obvious scalability issues [14]. Most theoretical works have focused on coflow scheduling in Big Switch model, where the

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

Jeremie Leguay, Youcef Magnouche, and Sebastien Martin are with Huawei Technologies, France Research Center (e-mails: firstname.lastname@huawei.com).

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).

communication graph is a complete bipartite graph. Since this problem is *NP*-hard, the main results concern the development of approximation algorithms. A series of papers [13], [15]–[21] reduced the approximation factor from $\frac{67}{3}$ to 4 for the case without release time, i.e., the time at which the coflow or flow is available. In order to design scalable algorithms, the main idea appearing in many research works is to give coflows a *static* priority order, known as a σ-order. For a given σ order, it is sufficient to adopt a work-preserving transmission policy which consists of never allowing the port to remain inactive when there is data from any flow (of any priority) to be transmitted. e.g., using priority queuing in the data plane. Inspired by the work [22] on open-shop scheduling problem, many different schemes have been developed to provide a σ-order that determines priority between coflows [13], [15]–[18]. Here, we highlight two of them, namely [12], [13], that are more relevant to ours. In both papers, the authors proposed an algorithm based on the primal-dual optimization problem: their solution is practical since it does not require to solve a linear program. Also, a deterministic 4-approximation guarantee is provided when all release time are zero. On top of this, Sincronia [12] proved that the approximation factor holds as long as flow-rate allocations are work-conserving and assigned by respecting the proposed σ-order. Ahmadi *et al.* [13] developed a centralized resource allocation scheme using the edge-shifting technique to avoid wasting resources if coflows are scheduled independently and sequentially. The main idea behind their scheme is to let all flows belonging to the same coflow to finish at almost the same time. A different class of algorithms rely on linear programming relaxation techniques. They return a solution whose value is guaranteed to be within a constant fraction of the optimal [19], [21], [23], [24]. However, when dealing with datacenters that handle a high number of coflows, techniques relying on linear programming relaxation may not be practical or feasible, due to the high number of variables involved.

Several types of schedulers have been proposed in the literature when no prior knowledge is available (e.g. flow volumes, flow arrivals). Semi-clairvoyant schedulers, where only volumes aggregated at ingress ports are available, have been studied in [25]. The proposed solution shows that it is robust to the lack of information about the egress ports. Non-clairvoyant methods have been studies as well in [4], [26]–[28], when prior knowledge (e.g., flow volumes, flow arrivals) is not available. A similar scheduler based on scheduling without prior knowledge of coflows was introduced in Aalo [29]. In [30], another joint scheduling and routing of flows in data center networks was presented, where similar heuristics based on a minimum time remaining first policy were developed.

It is important to observe that, up to now, all schedulers proposed in the literature cannot not offer tight approximation factors. This leaves room to explore heuristics beyond, e.g., those due to the open shop scheduling problem. The key idea we adopt in this work is based on the idea of measuring the impact of each coflow on the CCT, considering not only the gain per coflow, but also the impact that one coflow has on the average CCT of other coflows. This permits to design a new algorithm, called ONE-PARIS (ONE-steP-Ahead-loweR-Impact-firSt), to provide a σ-order. The main idea of ONE-PARIS is to approximate at each step the impact of scheduling a certain coflow onto the CCT of the remaining ones to be scheduled afterwards. In other words, the coflow to be scheduled at each step is the one that has the least impact on the other coflows in term of average CCT.

Main contributions. The proposed coflow scheduling framework has two components: (*i*) a robust coflow ordering algorithm; and (*ii*) a decentralized rate allocation scheme. First, we introduce ONE-PARIS, which outperforms state-of-the-art solutions such as Sincronia [12] (currently the best *in-class* algorithm). Similarly to Sincronia, it belongs to the scalable class of σ-order schedulers that decouple coflow prioritization from rate allocation. The main novelty though is to combine the feed-forward scheduling from Varys with the σ-order concept introduced by [12], [13]. But, while those algorithms establish the priority of coflows based on the bottleneck only, ONE-PARIS accounts for all links engaged by coflows and evaluates the priority based on the per-flow bottleneck evolution. Finally, we describe how to use the algorithm to schedule coflows in the online scenario, i.e., when coflow release times are unknown since coflows are generated at runtime.

Second, we propose a distributed bandwidth sharing algorithm, called Sync-Rate, that respects a given σ-order. It is proved to outperform Greedy, an algorithm for bandwidth sharing proposed along with Sincronia [12] that allocates rates accordingly to what priority queuing would do in the data plane. To further minimize the weighted CCT and address scalability issues, Sync-Rate allocates flow rates in a distributed manner striving to let flows in the same coflow finish at the same time while respecting the σ-order. It is rooted in Kelly's work [31] for network utility maximization. We further discuss how it can be implemented, in practice, in ToR switches and end-hosts.

Finally, we provide a performance evaluation for our comprehensive semi-decentralized coflow scheduling solution. For larger-scale instances, we present extensive numerical experiments to highlight the gains compared to existing schedulers, in particular Varys and Sincronia. Theses experiments are performed on both offline and online settings as well as clairvoyant and semi-clairvoyant settings using synthetic traces and real traces. Our solution is proved to outperform them in both clairvoyant and semi-clairvoyant cases, i.e., when the exact size of flows is unknown and only aggregated volumes are available from MapReduce APIs. In addition, when compared to Varys, ONE-PARIS provides very consistent improvements for almost all the coflows (about 96%) while Sincronia degrades the performance of up to 30% of them. The experiment results also show the comparison of Sync-Rate when compared to Greedy algorithm, the baseline rate allocation proposed along with Sincronia [12]. Our distributed algorithm achieves a moderate improvement up to 3.7%. Finally, we evaluate the performance of ONE-PARIS and Sincronia over a testbed composed of 60 machines. We use priority queuing in the data plane in order to perform rate allocation. Baseline performance figures are obtained when no coflow scheduling is applied. The results show that ONE-PARIS achieves a 187% gain against

Varys, while Sincronia improves completion time by 151%.

The rest of the paper is organized as follows. Sec. II introduces the general reference model for coflow scheduling. Sec. III derives the σ-order feedforward procedure of ONE-PARIS and Sec. IV presents the online version. Sec. V describes our distributed rate allocation procedure. Sec. VI reports numerical results and Sec. VII summarizes previous studies on coflow scheduling. Sec. VIII concludes the paper.

II. Background and Motivation

This section presents the coflow scheduling problem and discusses the limitations of state of the art approaches.

A. Problem Formulation

We consider a datacenter where K coflows are running in parallel. We consider a batch of coflows $\mathcal{C} = \{1, 2, ..., K\}$. For the sake of clarity, we suppose that all coflows arrive at the same time. However, in Section IV, we adapt our scheduler when coflows arrive sequentially and possibly in batches. The datacenter network can be modeled as a directed graph $G = (V, \mathcal{L})$, representing a Big Switch fabric, where $\mathcal{L}$ is the set of ingress/egress ports and V is the set of source/destination nodes. Each port l has a capacity b_l equals to the corresponding port's capacity; in the rest of the paper, all ports have unitary capacity.

Each coflow k is composed of n_k flows with $F_k = \{f_{k1}, f_{k2}, .., f_{kn_k}\}$. Each constituent flow f_{kj} is defined by a 3-tuple (s^{kj}, d^{kj}, v^{kj}) where $s^{kj}, d^{kj} \in V$ are source and destination nodes, and v^{kj} is the flow volume, i.e., the total amount of data to be transferred. Since a coflow is considered completed only when all its constituent flows are over, the Coflow Completion Time (CCT) for coflow k is given by

$$T_k = \max_{j \in \{1,...,n_k\}} \frac{v^{kj}}{\bar{r}^{kj}}, \tag{1}$$

where $\bar{r}^{kj}$ represents the average rate of flow f_{kj} through its lifetime. It is defined as follows

$$\bar{r}^{kj} = \frac{1}{CT^{kj}} \int_0^{CT^{kj}} r^{kj}(t)dt,$$

with $r^{kj}(t)$ is the bandwidth allocated to flow f_{kj} at time t and CT^{kj} its completion time. This quantity $\bar{r}^{kj}$ rules the transfer time of the data volume traversing the so-called coflow bottleneck and determines in fact the CCT of the coflow. We use w_k to denote the weight, i.e., the importance of each coflow $k \in \mathcal{C}$, that is typically given by the application scheduler so that the overall completion time can be optimized with regard to priorities. Define

$$S = \sum_{k \in \mathcal{C}} \sum_{j \in F_k} \frac{v^{kj}}{\min_{l \in \mathcal{L}} b_l}$$

The value of S can be viewed as the upper bound of minimum time required for scheduling all the coflows. The total Weighted Coflow Completion Time Minimization (WCCT) problem is defined as follows:

$$\min_{\mathbf{r}} \quad \sum_{k \in \mathcal{C}} w_k T_k \tag{2a}$$

$$\text{s.t.} \quad r^{k_1 j}(t) \geq 0, \quad \forall t \in [0, S], \tag{2b}$$

$$\sum_{k \in \mathcal{C}} \sum_{j \in F_k} r^{kj}(t) x_l^{kj} \leq b_l, \quad \forall l \in \mathcal{L}, \forall t \in [0, S]. \tag{2c}$$

where x_l^{kj} is an indicator of whether flow f_{kj} passes through port l. Constraints (2c) express that, at any instant $t \in [0, S]$, the total flow cannot exceed the port capacity.

As showed in [7], for preemptive scheduling, the WCCT problem is proved *NP*-hard by reduction to the concurrent open-shop problem.

B. Motivating Example

Using a simple example, we illustrate to what extent ignoring the cumulative impact that a coflow has on others in terms of resources utilization can ultimately harm the average CCT. As depicted in Fig. 1, we consider a batch of 4 coflows that are represented by different color bands: C_1 is composed by 5 flows (blue) and C_2, C_3 and C_4, are composed by only one flow each (green, black and orange, respectively). We used the standard pictorial representation of a Big Switch fabric [7], where the vertical position of the color band indicates the egress port. We consider unitary capacity on all ports and unitary weights for all coflows, i.e., $w_k = 1$, $k \in \{1, ..., K\}$. Finally, the number in the input band represents the flow volume, normalized in traffic units. Using

Table I
Coflows and their flows at the 4 ingress and 4 egress ports

Coflow	Flow Id	Volume	Ingress Ports	Egress Ports
C_1	f_{11}	6.6	4	5
	f_{12}	6.6	3	5
	f_{13}	6.6	1	6
	f_{14}	3.3	2	8
	f_{15}	3.3	1	8
C_2	f_{21}	$10 + \epsilon$	4	7
C_3	f_{31}	10	1	7
C_4	f_{41}	$10 + 2\epsilon$	2	7

Varys algorithm introduced in [7], the CCT of each coflow is computed first considering all of them in isolation. As coflow C_3 has the smallest CCT in isolation, Varys schedules it first and gives all the bandwidth to its unique flow f_{31}. This decision blocks coflows C_2 and C_4 on egress port 7 and coflow C_1 on ingress port 1 since flows f_{13} and f_{15} use these ports. According to the Minimum-Allocation-for-Desired-Duration (MADD) algorithm used in Varys, no bandwidth is allocated to any of the flows of C_1 during this round. Overall, Varys provides 25.8s as average CCT.

Now we characterize the coflow ordering produced by Sincronia [12]. At each step, the algorithm selects the bottleneck ingress or egress port. It chooses the coflow with the largest weighted processing time and places it last among all unscheduled coflows. Finally, Sincronia scales the weights of

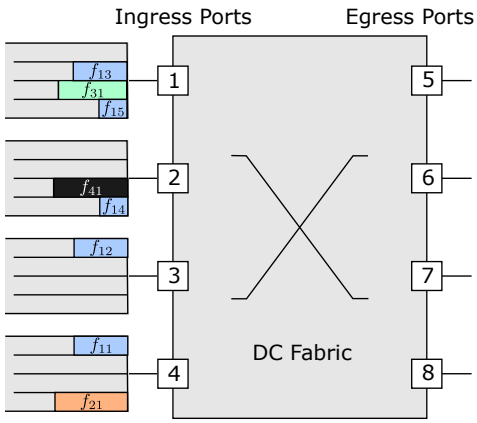


Figure 1. Coflow scheduling over a Big Switch datacenter fabric with 4×4 ingress/egress ports. Flows in ingress are organized vertically by destination (i.e., egress) ports and color-marked per coflow: C_1 in blue, C_2 in orange, C_3 in green and C_4 in black.

Table II
EXECUTION OF SINCRONIA ON THE EXAMPLE OF FIG. 1. THE AVERAGE CCT IS $(13.2 + 26.6 + 19.9 + 33.3)/4 = \mathbf{23.25}$S (USING GREEDY RATE ALLOCATION).

Unscheduled coflows (set $\mathcal{C}_u$)	**Bottleneck**	**Sincronia's weights** $\{w_1, w_2, w_3, w_4\}$
		$\{1, 1, 1, 1\}$
$\mathcal{C}_u = \{C_1, C_2, C_3, C_4\}$	7	$\{1, \frac{\epsilon}{100+2\epsilon}, \frac{2\epsilon}{100+2\epsilon}, 0\}$
$\mathcal{C}_u = \{C_1, C_2, C_3\}$	7	$\{1, 0, \frac{\epsilon(100-2\epsilon)}{(100+2\epsilon)(100+\epsilon)}, 0\}$
$\mathcal{C}_u = \{C_1, C_3\}$	7	$\{1, 0, 0, 0\}$
$\mathcal{C}_u = \{C_1\}$	5	$\{0, 0, 0, 0\}$

all unscheduled coflows active on the same bottleneck. Table II shows the execution on the example of Fig. 1. At the first iteration, Sincronia chooses bottleneck egress port 7, and selects the largest coflow (in volume) to schedule last, i.e., coflow C_4. The algorithm then scales the weights of coflow C_2 and C_3 while keeping coflow C_1's weight unchanged. The weights of Coflow C_2 and C_3 are scaled as follows: $w_k \leftarrow w_k - w_4 \frac{v^{k1}}{v^{41}}$, $k = 2, 3$. At the second iteration, the egress port 7 is still the bottleneck and coflow C_2 is selected. By following the same procedure, the ordering given by Sincronia in that example is $\{C_1, C_3, C_2, C_4\}$. Given this coflow ordering, the Greedy algorithm [12] used by Sincronia to allocate flow rates obtains 23.5s as the average CCT.

In this example Sincronia and Varys are proved to perform very far from the optimal solution.Indeed, we observe that coflow C_4 uses ingress ports 2 and 7 and ingress port 2 is only used by flow f_{14}. Hence, it is more efficient to schedule C_4 first since its impact on other coflows is limited compared to other coflows. If coflow C_4 is scheduled first, then coflow C_1 will monopolize all other ports since coflow C_2 and C_3 are blocked by coflow C_4. At $10s$, coflow C_4 finishes to transfer its demand, and then C_1 can use ingress port 2 to transfer the demand of flow f_{14} and obtains $13.3s$ as CCT. After C_1, coflows C_2 and C_3, will subsequently inherit the bottleneck port bandwidth to transfer their demands. The order of coflow C_2 and C_3 has no impact on the average CCT. As a result, all coflows C_4, C_1, C_2, C_3 finish at times 10s, 13.3s, 20s and 30s, respectively. The average CCT is 18.325s, which can be proved to be optimal via ILP: it improves by 22% compared to Sincronia and by 31.8% compared to Varys.

The need for an all-port approach. In this example, Varys and Sincronia neglect the impact that scheduling a coflow has on the average CCT of the other ones. The root cause is that both evaluate the impact of one coflow on other coflows by looking to its bottleneck and not through all ports where it is active. Therefore, they misjudge the order of coflows and degrade the average CCT. In this sense, Sincronia does much better than Varys because, in constructing its schedule, it can decreases the weight of each coflow using the bottleneck. In this case, if a coflow uses more links, its weight may decrease rapidly, increasing the chance that it will be scheduled in lower priority. On the other hand, Sincronia needs several steps before evaluating the impact of a coflow on other coflows. The need for an all-port approach motivates the design of a new σ-order scheduler, called ONE-PARIS.

We further design a distributed rate control algorithm, called Sync-Rate, that complies with the given σ-order.

III. σ-ORDER SCHEDULING WITH ONE-PARIS

In this section, we present the core of ONE-PARIS (ONE-steP-Ahead-loweR-Impact-firSt) for coflow scheduling. Given a list of K coflows and their weights, it provides a permutation $(\sigma(1), \sigma(2), \ldots, \sigma(K))$ that indicates the order in which coflows must be transmitted to minimize the weighted average completion time.

When a coflow is scheduled, it will impact other lower priority (i.e., rank) coflows. Hence our primary objective is to evaluate the impact of such scheduled coflow on the weighted CCT. In practice, this makes the search space exponentially large because there are $K!$ possible coflow orderings. Hence, we propose a new approach that approximates at each step the impact of scheduling a certain coflow on the remaining coflows scheduled afterwards. The main idea of ONE-PARIS is to give priority to the coflow that has the least impact on other coflows.

We suppose that all coflows arrive at the same time to ease the analysis. Note that our scheduler applies also to coflows with different arrival times (by using updated information on volumes). ONE-PARIS only needs to compute an estimation of the CCT of each coflow based on their respective bottlenecks. The algorithm is described in Alg. 1.

For ease of notation, let $\mathcal{C}_s = \{\sigma(1), \sigma(2), .., \sigma(n-1)\}$ be the set of coflows that have already been scheduled at step $n-1$. For each coflow $k \in \mathcal{C} \backslash \mathcal{C}_s$, we compute the CCT of coflow k and $k' \in \mathcal{C} \backslash \mathcal{C}_s \cup \{k\}$ when coflow k is scheduled after all coflows in $\mathcal{C}_s$, and coflow k' after k:

$$T_k^{\mathcal{C}_s} = \max_{l \in \mathcal{L}} \left\{ \frac{\sum_{i \in \mathcal{C}_s \cup \{k\}} \bar{v}_l^i}{b_l} \right\} \quad (3)$$

$$T_{(k,k')}^{\mathcal{C}_s} = \max_{l \in \mathcal{L}} \left\{ \frac{\sum_{i \in \mathcal{C}_s \cup \{k,k'\}} \bar{v}_l^i}{b_l} \right\} \quad (4)$$

where $\bar{v}_l^i$ is the total data volume of coflow i transferred through port l. Thus, the total weighted CCT of coflow k and

Algorithm 1 ONE-PARIS.

Input: Sets of unscheduled coflows $\mathcal{C}_u = \{1, \ldots, K\}$ and associated weights $\mathcal{W} = \{w_1, \ldots, , w_K\}$.
Output: Coflow ordering $\mathcal{C}_s = (\sigma(1), \sigma(2), \ldots, \sigma(K))$.

1: $\mathcal{C}_s \leftarrow \emptyset$.
2: **while** $\mathcal{C}_u$ is not empty **do**
3: **for** $k \in \mathcal{C}_u$ **do**
4: **for** $k' \in \mathcal{C}_u \backslash \{k\}$ **do**
5: $\overline{T}^{\mathcal{C}_s}_{(k,k')} = w_k.T^{\mathcal{C}_s}_k + w_{k'}.T^{\mathcal{C}_s}_{(k,k')}$, (Eq. (5));
6: **end for**
7: Compute $\hat{\mathbf{T}}^{\mathcal{C}_s}_k = \sum_{k' \in \mathcal{C}_u \backslash \{k\}} \overline{T}^{\mathcal{C}_s}_{(k,k')}$ (Eq. (6));
8: **end for**
9: Select the coflow with lowest impact on unscheduled coflows:

$$\sigma(n) = \arg\min_{k \in \mathcal{C}_u} \hat{\mathbf{T}}^{\mathcal{C}_s}_k;$$

10: Update the set of unscheduled coflows: $\mathcal{C}_u \leftarrow \mathcal{C}_u \backslash \{\sigma(n)\}$;
11: Update orderred coflows: $\mathcal{C}_s \leftarrow (\mathcal{C}_s, \sigma(n))$;
12: **end while**

k' under coflow ordering $(\mathcal{C}_s, k, k')$ is (Line 7 in Algo. 1)

$$\overline{T}^{\mathcal{C}_s}_{(k,k')} = w_k T^{\mathcal{C}_s}_k + w_{k'}.T^{\mathcal{C}_s}_{(k,k')} \tag{5}$$

Then, for each coflow, we calculate the following quantity in order to estimate the global impact of coflow C_k on others coflows if it is scheduled after coflows in $\mathcal{C}_s$

$$\hat{\mathbf{T}}^{\mathcal{C}_s}_k = \sum_{k' \in \mathcal{C}_u \backslash \{k\}} \overline{T}^{\mathcal{C}_s}_{(k,k')} \tag{6}$$

The coflow that will be scheduled at step n, i.e. $\sigma(n)$, is the coflow that has minimal value $\overline{T}^{\mathcal{C}_s}_k$, i.e.,

$$\sigma(n) = \arg\min_{k \in \mathcal{C}_u} \hat{\mathbf{T}}^{\mathcal{C}_s}_k. \tag{7}$$

We then update the set of coflows that have been scheduled till step n, i.e., $\mathcal{C}_s = \{\sigma(1), \sigma(2), .., \sigma(n-1), \sigma(n)\}$

To illustrate the difference between ONE-PARIS, Varys and Sincronia, we consider again the example of Fig. 1.

Table III shows the execution of ONE-PARIS on the example of Fig. 1. In the first step, the algorithm evaluates the impact of each coflow on other coflows if it is scheduled first. To do that we compute the value $\hat{\mathbf{T}}^{\mathcal{C}_s}_k$ using Eq. (6). The algorithm chooses coflow C_4 since it achieves the minimum $\hat{\mathbf{T}}^{\mathcal{C}_s}_k$, i.e., 833. At the next iteration, the algorithm makes a similar calculation considering unscheduled coflows until all of them are scheduled. In that example, the final σ-ordering produced by ONE-PARIS is $\mathcal{C}_s = (C_4, C_1, C_3, C_2)$. Using the *Greedy* algorithm developed in [12][1] to allocate bandwidth based on a σ-order, the average CCT is 18.325s. Hence, we observe that the average CCT is decreased by 22% compared to Sincronia and by 31.8% compared to Varys.

Now, we evaluate ONE-PARIS when only a partial prior knowledge on flow volumes is available. In particular, we consider the case when the exact volume per flow is unknown. From our analysis in a real MapReduce cluster, the aggregated volume sent by mappers at each ingress port can be collected through the management API. Hence, under such *semi-clairvoyant* setting, only the aggregated volumes at ingress ports are available. Using just this information, ONE-PARIS obtains the order C_4, C_2, C_3, C_1, attaining same average CCT as for the case with full information. Sincronia obtains the order (C_2, C_4, C_1, C_3) since the bottleneck at step 1 is egress port 4 instead of ingress port 7. Curiously, Sincronia achieves better performance than in the case of full information, since the average CCT obtained by this order is $19.15s$. This shows clearly that ONE-PARIS is more robust even when using partial volume information, as Sincronia doesn't take into account the real full impact of a coflow on other coflows.

[1] At any time, a flow of a coflow is blocked if only and if its ingress or egress port is fully occupied by other flows of higher priority coflows. It mimics a strict priority scheduling of flows in the data plane with order priorities.

Table III
EXECUTION OF ONE-PARIS ON THE EXAMPLE OF FIG. 1. THE AVERAGE CCT IS $(13.3 + 30 + 20 + 10)/4 = \mathbf{18.325}$S (USING GREEDY RATE ALLOCATION).

Set $\mathcal{C}_s$	$C_1^{\mathcal{C}_s}$	$C_2^{\mathcal{C}_s}$	$C_3^{\mathcal{C}_s}$	$C_4^{\mathcal{C}_s}$
$\mathcal{C}_s = \emptyset$	894	866	899	**833**
$\mathcal{C}_s = \{C_4\}$	**631**	866	899	
$\mathcal{C}_s = \{C_4, C_1\}$		500	**500**	
$\mathcal{C}_s = \{C_4, C_1, C_3\}$		**300**		
$\mathcal{C}_s = \{C_4, C_1, C_3, C_2\}$				

Algorithm 2 Online Algorithm for Coflows Scheduling

1: **if** a new coflow arrives at time t_n **then**
2: Update previous coflow batch based one of the scheme: *Full update strategy* or or *Weighted update strategy* $\rightarrow \mathcal{C}_n$;
3: Include the newly arrived coflow in $\mathcal{C}_n$;
4: Calculate the remaining volume of coflows in $\mathcal{C}_n$;
5: **if** Weighted update strategy is used **then**
6: Update the weight of each coflow using (8);
7: **end if**
8: Compute $\sigma^{(n)}$-order ONE-PARIS algorithm for coflows in $\mathcal{C}_n$;
9: Bandwidth allocation $\mathbf{r}^{kj} \leftarrow$ Sync-Rate(σ^n) or Greedy(σ^n);
10: **end if**

IV. FROM OFFLINE TO ONLINE SCHEDULING

We now study how to incorporate the ONE-PARIS scheduler into a system where coflows may arrive at arbitrary times. The output σ-order is modified in order to adapt at arrival events depending on the input traffic. To this aim, we provide a high-level description of the online version of ONE-PARIS. Let us consider a batch $\mathcal{C}$ of coflows which arrive at different times. In the online setting, the information on coflow k, namely its flow set $F_k = \{f_{k1}, .., f_{kq}\}$ and the each flow 4-tuple $f_{kj} = (s^{kj}, d^{kj}, v^{kj}, T_n)$, is disclosed at arrival time. The main idea of the online version of the algorithm is that when a new coflow arrives or a flow completes its transmission, the rate allocation for each existing flows in the fabric has to be updated. However, a new σ-order should be generated just when a new coflow arrives. The pseudocode of the online algorithm is reported in Alg. 2. The Online version of ONE-PARIS is detailed in Algo. 2. It has three parts: lines 1-7 detail the list of coflows that the scheduler recalculate the σ-order when a new arrival coflow happened; line 8 calls one of the coflow ordering algorithm to update the new order of coflows. Actually, as indicated in Alg. 2, after computing the

order of coflows, we can use either our decentralized resource allocation algorithm Sync-Rate, presented in Sec. V, or the Greedy rate allocation algorithm described in [12].

In particular, as seen in line 2, the algorithm can use two different schemes to re-compute the order of existing coflows by including the newly arrived one:

- **Full Update Strategy (FUS)**: We update the remaining volume for each flow and based on the updated volume of all existing coflows;
- **Weighted Update Strategy (WUS)** The weighted update scheme makes a tradeoff between weighted CCT minimization and coflow starvation. The weight of a coflow already scheduled is increased over time as follows

$$w_k(t_{n+1}) = w_k(t_n) + \eta \max(0, (T_k^{t_n} - T_k^0)), \quad (8)$$

where η is a positive scaling factor and $T_k^{t_n}$ (resp. T_k^0) is the estimated CCT of flow k at time t_n (resp. upon arrival). This scheme allows to avoid starvation since the weights of the existing coflows increase if their CCT increases due to new coflow arrivals. This scheme allows to keep priorities between coflows while taking into account the impact of new coflow arrivals on existing ones. We note that if a coflow is not impacted by an arrival, its weight remains unchanged. This scheme makes our online algorithms more fair when treating big-coflows against small coflows without sacrificing the average WCCT.

For both WUS and FUS, once the input coflow batch is updated, the σ-order is re-calculated by using ONE-PARIS.

V. Sync-Rate: Distributed Rate Control

As the total number of active flows may be in the order of tenths of thousands in a production datacenter, a centralised rate allocation is not viable. Actually, the σ-order could simply be used to enforce priorities in the data plane using a strict priority DiffServ scheduler [32]. But, in practice, the number of queues is limited in standard devices (e.g., to 8 typically in production switches). Also, priority queuing performs a local greedy allocation of bandwidth, similarly to Greedy [12], which is not necessarily the most efficient choice for rate allocation. The average CCT can be further improved with a better control over bandwidth sharing. Indeed, individual flow rates need to be continuously updated based on their progress, i.e., based on the volume of data transferred.

To make rate allocation scalable and fit to real-time network performance, we design *Sync-Rate*, a new distributed algorithm for rate allocation. It takes as input the σ-order of coflows (i.e., priorities) given by ONE-PARIS and decides how network bandwidth must be shared among the different flows. Sync-Rate performs a distributed rate control of flows and, as shown later, it can be implemented at end-hosts and ToR switches. It is based on a distributed pricing mechanism and performs a real-time rate allocation based on the actual congestion and the actual data volumes transferred by flows. It strives to let all flows of each coflow finish at the same time or as close as possible. This is known to be beneficial for the average CCT as it strives to leave bandwidth for lower-priority coflows. The algorithm is based on a backpressure signaling, able to converge to the optimal rate allocation.

The intuition behind Sync-Rate is as follows: once the σ-order is established, priority in bandwidth allocation is granted by that order. As such, in between the departure of flows, it is possible to consider the equivalent stationary system, where rates are allocated respecting the pre-emption order and trying to equalise the completion of flows belonging to the same coflow by using Kelly's theory for rate allocation [31].

A. Network Utility Maximization

Since the publication of the seminal work [31] by Kelly et al., the Network Utility Maximization (NUM) framework has found many applications for rate allocation in networks. In [33], Dynamic Bandwidth Allocation (DBA) is proposed as a scheme based on pricing mechanism for resource allocation for coflows, but it requires the knowledge of the total remaining completion time of coflows. Moreover, it cannot control flow rates according to a given priority in a dynamic case (e.g., after a departure of flow). In our scheme we adapt the NUM framework to the case where coflows are scheduled based on a σ-order and the rate control is operated at each source without direct coordination with other sources.

Let now consider a set of coflows that should be scheduled with following order $\sigma = (\sigma_1, \sigma_2, .., \sigma_K)$. The NUM problem associated to coflow σ_n is

$$P_{\sigma_n} : \max_{r^{\sigma_n}} \quad \sum_{j \in F_{\sigma_n}} v^{\sigma_n j} \log\left(r^{\sigma_n j}\right) \quad (9a)$$

$$\text{subject to} \quad r^{\sigma_n j} \geq 0, \ \ \forall j \in F_{\sigma_n}, \quad (9b)$$

$$\sum_{j \in F_{\sigma_n}} r^{\sigma_n j} x_l^{\sigma_n j} \leq \max(0, b_l - \sum_{z=1}^{n-1} \sum_{j' \in F_{\sigma_z}} r^{\sigma_z j} x_l^{\sigma_z j'}), \forall l \in \mathcal{L}. \quad (9c)$$

The solution of P_{σ_n} depends on the solution of problems P_{σ_z}, where σ_z with $z = 1.., n-1$ are coflows of higher-priority than σ_n. Indeed, P_{σ_n} aims at maximizing the overall utility of σ_n on the remaining capacity left by higher-priority coflows. As such, it is by design compliant with the input σ-order.

As showed by Kelly, the weighted log-sum objective function in P_{σ_n} can be seen as the network utility function associated to coflow σ_n. Such function, as it will be clear in the rest of the discussion, serves the purpose to attain a proportional fair share of the link capacity among the flows of the same coflow at each time interval. In practice, this optimisation problem forces all flows of a coflow sharing a common link to finish at the same time. The fairness criterion avoids the use of a complicated model with additional constraints between flows belonging to the same coflow. The constraints (9c) represent capacity constraints, which limit the sum of the rates to be less than or equal to the remaining capacity left by the highest priority coflows. The optimisation problem P_{σ_n} can be solved in centralized way, since each optimisation problem concerns a coflow is mathematically fairly tractable (with a strictly concave objective function and a convex feasible region). However, with a very high number of flows in each coflow, this may not be practical or feasible

in datacenters handling a high number of coflows. For this reason, we focus only on the distributed solution, as it's more scalable than the centralized one.

We consider the dual problem for P_{σ_n}, whose structure suggests treating the ingress-egress ports and flows as processes of a distributed computation system by means of a gradient projection method. Indeed, each flow of coflow σ_n can execute a local algorithm based on the dual variables $\lambda_l^{\sigma_n}$ associated to capacity constraints (9c).

To derive the algorithm steps, we first define the Lagrangian function as follows

$$\begin{aligned} &L^{\sigma_n}(r^{\sigma_n}, \lambda^{\sigma_n}) = \\ &\sum_{j\in F_{\sigma_n}} v^{\sigma_n j}\log\big(r^{\sigma_n j}\big) - \sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}\Big(\sum_{z=1}^{n}\sum_{j\in F_{\sigma_z}} r^{\sigma_z j}x_l^{\sigma_z j} - b_l\Big) \\ &= \sum_{j\in F_{\sigma_n}} v^{\sigma_n j}\log\big(r^{\sigma_n j}\big) - r^{\sigma_n j}\sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}x_l^{\sigma_n j} \qquad (10)\\ &- \sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}\Big(\sum_{z=1}^{n-1}\sum_{j\in F_{\sigma_z}} r^{\sigma_z j}x_l^{\sigma_z j} - b_l\Big), \end{aligned}$$

where $\mathcal{L}_n$ is the set of links used by coflow σ_n. As the two terms in (10) are separable in $r^{\sigma_n j}$, we have

$$\begin{aligned} &\max_{r^{\sigma_n}} \sum_{j\in F_{\sigma_n}} v^{\sigma_n j}\log\big(r^{\sigma_n j}\big) - r^{\sigma_n j}\sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}x_l^{\sigma_n j} \\ =& \sum_{j\in F_{\sigma_n}} \max_{r^{\sigma_n j}}\Big(v^{\sigma_n j}\log\big(r^{\sigma_n j}\big) - r^{\sigma_n j}\sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}x_l^{\sigma_n j}\Big) \\ \stackrel{\text{def}}{=}& \sum_{j\in F_{\sigma_n}} G_{\sigma_n j}(\vec{\lambda}^{\sigma_n}) \end{aligned}$$

where $\vec{\lambda}^{\sigma_n} = (\lambda_l^{\sigma_n})_{l\in\mathcal{L}}$. The dual problem is

$$D_{\sigma_n} : \max_{\vec{\lambda}^{\sigma_n}} \sum_{j\in F_{\sigma_n}} G_{\sigma_n j}(\vec{\lambda}^{\sigma_n}) - \sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}\Big(\sum_{z=1}^{n-1}\sum_{j'\in F_{\sigma_z}} r^{\sigma_z j'}x_l^{\sigma_z j'} - b_l\Big) \qquad (11)$$

We observe that the first term of (11) can be decomposed into $|F_{\sigma_n}|$ separable subproblems. If we interpret the Lagrangian multiplier $\lambda_l^{\sigma_n}$ as the price per unit of bandwidth on port l paid by coflow σ_n, thus the total price paid by each flow $j \in F_{\sigma_n}$ is $\mu^{\sigma_n j} = \sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}x_l^{\sigma_n j}$. The second term of the dual problem takes into account the priority of coflow σ_n and in particular higher-priority coflows σ_z, with $z = 1, .., n-1$.

The price vector $\vec{\lambda}^k$, $k \in \mathcal{C}$ is used to handle the resource allocation for all flows of coflows. It allows to satisfy the σ-order and to perform rate control for each flows depending only on its remaining volume. It ensures that all flows of a coflow finish at the same time if they share a bottleneck link.

Hence each flow of a coflow σ_n continuously solves the following maximization problem:

$$P_f : G_{\sigma_n j}(\vec{\lambda}^{\sigma_n}) = \max_{r^{\sigma_n j}}\Big(v^{\sigma_n j}\log\big(r^{\sigma_n j}\big) - r^{\sigma_n j}\sum_{l\in\mathcal{L}_n}\lambda_l^{\sigma_n}x_l^{\sigma_n j}\Big) \qquad (12)$$

For each fixed $\vec{\lambda}^{\sigma_n}$, there exists an unique optimal rate $r^{\sigma_n j}$ for each flow j since the function $\log$ is strictly concave.

In practice, the optimization problem (12) allows each flow, given a fixed price, to find the optimal rate allocation without having to coordinate with the other flows. Let b^{kj} denote the minimum bandwidth of ingress and egress ports used by flow f_{kj}. Thus $b^{kj} = \min_{l\in\mathcal{L}}\{\, b_l | x_l^{kj} = 1\}$. The optimal rate of flow $f_{\sigma_n j}$ satisfying the KKT conditions is:

$$r^{\sigma_n j}(\mu^{\sigma_n j}) = \begin{cases} \frac{v^{\sigma_n j}}{\mu^{\sigma_n j}} & \mu^{\sigma_n j} \geq \frac{v^{\sigma_n j}}{b^{\sigma_n j}} \\ b^{\sigma_n j} & \text{otherwise} \end{cases} \qquad (13)$$

We can see that if several flows of a coflow share a common link their rate will be proportional to their remaining volume.

Now If the solution provided by the optimization problem P_f via equation (13), is incorporated into the optimization problem D_{σ_n}, then the resulting solution $(\mathbf{r}^{\sigma_n}, \vec{\lambda}^{\sigma_n})$ satisfies the optimality condition of P_{σ_n}.

B. Sync-Rate Design and Implementation

We now discuss a possible implementation of Sync-Rate where its functions are distributed between end-hosts and ToR switches. Indeed, rate control can be operated at end-hosts while congestion management (i.e, pricing of network resources) can be handled at switches. Implementations with all functions running either at sources or switches are also possible, we will omit them to ease presentation.

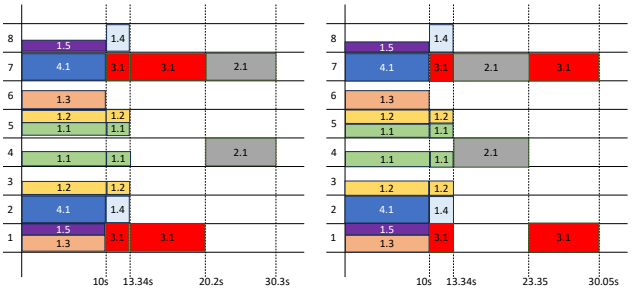
1) Congestion Management at Switches: Each ToR switch can maintain for each of its outgoing links $l \in \mathcal{L}$ the price $\lambda_l^{\sigma_n}$ for each coflow σ_n based on the σ-order. To update the price of each coflow at each link (i.e., port in the Big Switch model) periodically, the switch solves the dual problem (11) using the following low-complexity gradient descent step

$$\lambda_l^{\sigma_n}(t+1) = \Big[\lambda_l^{\sigma_n}(t) + \gamma_{\sigma_n}\Big(\sum_{z=1}^{n}\sum_{j\in F_{\sigma_z}} r^{\sigma_z j}x_l^{\sigma_z j} - b_l\Big)\Big]_+, \qquad (14)$$

where $\gamma_{\sigma_n} > 0$ denotes the step-size. In Theorem 1 we show the condition on γ_{σ_n} that ensures the convergence of $\lambda_l^{\sigma_n}(t)$. We observe that price $\lambda_l^{\sigma_n}(t+1)$ depends on the network traffic of higher-priority coflows and coflow σ_n. Thus, the price should be interpreted as a measure of congestion: it increases when traffic approaches link capacity and decreases otherwise. By design, the price of coflow σ_n is not affected by coflows with lower priority than coflow σ_n. This ensures that the resource allocation of flows belonging to the coflow σ_n is not affected by the resource allocation of coflows $\sigma_{n'}$, $n' > n$ flows, i.e., coflows whose priority is lower than that of the coflow σ_n. Hence the Langrange multipliers $\vec{\lambda}^{\sigma_n}$, $n = 1 \dots K$, act as a coordinator to manage the priorities given by ONE-PARIS. Incidentally, $\vec{\lambda}^{\sigma_n}$ can be interpreted as the congestion price, which only takes into account flows with a higher priority than coflow σ_n and itself.

As we shall indicate in the following, in order to overcome the signaling overhead, it is possible to directly embed the price inside IP forward packet headers. Thus, prices piggybacked via transport layer ACK packets can drive the backpressure mechanisms accordingly.

2) Rate allocation at end-hosts: Each source (i.e., end-host) collects the total price $\mu^{\sigma_n j}$, which corresponds to the marginal utility function $v^{\sigma_n j}\log\big(r^{\sigma_n j}\big)$. When the source of flow j of coflow σ_n receives a new value of the total price, it then



(a) $\sigma = (C_4, C_1, C_3, C_2)$. (b) $\sigma = (C_4, C_1, C_2, C_3)$.

Figure 2. Sync-Rate resource allocation on the example of Fig. 1.

updates the rate according to Eq. (13). Here we summarize how Sync-Rate operates:

1) Every switch computes the total price for all active flows based on their order σ. It adds a field into ACK headers to indicate $\mu^{\sigma_n j}$ to sources, the cumulative congestion along ingress/egress ports. When the packet of flow j of coflow σ_n passes through a port l, $\mu^{\sigma_n j}$ is increased by $\lambda_l^{\sigma_n}$;
2) The source adjusts its rate $r^{\sigma_n j}$ according to Eq. (13), considering the updated volume and the coflow price received through in-band signaling.

Theorem 1. *Let $\{\vec{\lambda}^{\sigma_n}\}$ be a sequence generated by Eq. (14) such that $\vec{\lambda}^{\sigma_n}(0) \in \mathcal{R}^{|\mathcal{L}|}$ and $\gamma_{\sigma_n} \in (0, \frac{2}{M_{\sigma_n}})$ where*

$$M_{\sigma_n} = \sqrt{|\mathcal{L}|} \sum_{j \in F_{\sigma_n j}} 2\frac{(b^{\sigma_n j})^2}{v^{\sigma_n j}}$$

the sequence $\vec{\lambda}^{\sigma_n}(t)$ converge to optimal solution $\bar{\lambda}^{\sigma_n}$ of the dual problem (11), i.e., $\lim_{t\to\infty} \lambda^{\sigma_n}(t) = \bar{\lambda}^{\sigma_n}$.

Proof. The proof is given in the appendix. □

A run of Sync-Rate on is depicted in the example in Fig. 1 using the order provided by ONE-PARIS. We observe in Fig. 2(a) that Sync-Rate is able to allocate rates in a distributed way similar to offline mechanisms. In particular, note that Sync-Rate guarantees all flows of coflow C_1 sharing the same port do terminate at the same time. This is why flows f_{11} and f_{12} or f_{13} and f_{15} share the capacity of a port in proportion to their volume. Now, if the σ order is $\sigma = \{C_4, C_1, C_2, C_3\}$, we can see in Fig. 2(b) that the flow f_{31} of coflow 3 is interrupted when the f_{21} flow starts using output port 7 making ONE-PARIS a pre-emptive scheduler, as required by σ-order rate allocation theory [12].

VI. Performance Evaluation

We now present the performance evaluation of ONE-PARIS and Sync-Rate. The results are obtained on the instances generated by our workload generator and a self-developed flow-level simulator in MATLAB®. We compare the performance of our solutions against state of the art schedulers, namely Varys and Sincronia. As the solution proposed in [13] achieves almost the same performance as Sincronia[2], we keep only Sincronia as a reference for comparison with our solution.

We first generated different workloads to reflect typical coflow patterns observed in data centers [34]. We then used popular workloads from Facebook traces [7].

We tested our solutions both in the clairvoyant and the semi-clairvoyant settings. For the semi-clairvoyant setting, we relaxed the prior knowledge assumption on flow volumes by assuming that only the aggregated volume at ingress ports is known. In a specific set of results, we evaluated our distributed rate control solution against Greedy, the baseline rate allocation algorithm proposed along with Sincronia [12], which schedules flows in an order-preserving manner. Furthermore, we evaluated the online version against the online versions of Sincronia and Varys. Finally, we evaluated ONE-PARIS in a testbed with 60-machines.

A. Simulation Setup

Synthetic Traffic: We considered M machines or end-hosts connected to a non-blocking Big Switch fabric. Access links have unitary capacity. For each workload, we generated 1000 instances by fixing the number of machines, the number of coflows and the average of flow volume. The number of coflows ranges from 10 to 50 for workloads 1 and 2, with $M = 10$ and from 50 to 200 for for workloads 3 and 4, with $M = 200$. Each workload comprises two types of coflows: coflows with a single flow, named *type* 1, and coflows with multiple flows, named *type* 2. Coflows of type 2 coflows have a random number of flows uniformly generated in $\frac{M}{3}$ and M, where M is the number of machines. The percentage of coflows of type 2 equals to 80% for workloads 1 and 3, and 20% for workloads 2 and 4.

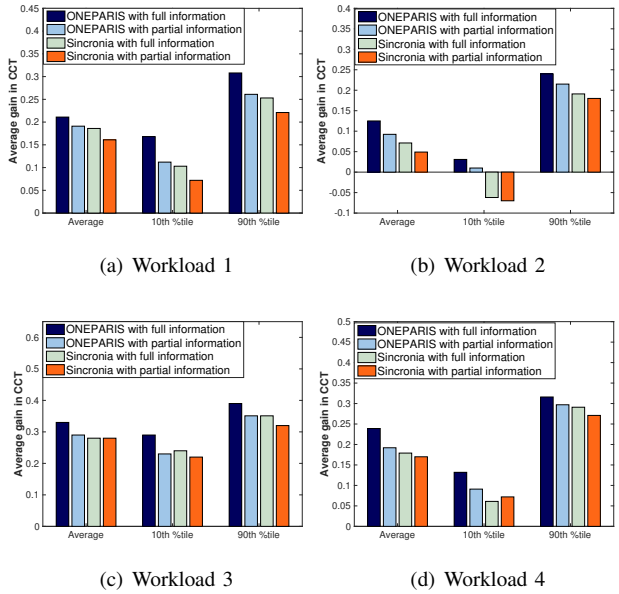
Real Traffic: Real traffic datasets are obtained by the Facebook traces dataset [7], based on a MapReduce shuffle trace collected from one of Facebook's 3000-machine cluster with 150 racks. The data traces contains 150 ports and 526 coflows (more than 7×10^5 flows). It has a skewed coflow width distribution, ranging from coflows with a single flow to very large ones (the largest coflow has 21,170 flows). For each configuration $[M, N]$, N coflows are randomly sampled from the Facebook dataset. Coflows are only selected from the ones that have fewer than K flows. The volume of each flow is given by the dataset.

Metric: We evaluate the algorithms based on the average CCT, i.e, we consider all coflows having same weight. For each workload, we report on the gains for ONE-PARIS and Sincronia over Varys for the CCT using the following formula:

$$\text{Average gain CCT} := \frac{\text{CCT under Varys-Compared CCT}}{\text{CCT under Varys}}$$

We also present the gains in percentiles of each algorithm with respect to the solution yielded by Varys in terms of CCT.

[2]Sincronia and the scheduler in [13] achieve the same order σ for at least 97% of instances in all workloads



(a) Workload 1 (b) Workload 2

(c) Workload 3 (d) Workload 4

Figure 3. Gain in CCT of ONE-PARIS and Sincronia against Varys using Greedy rate allocation (with and without full information).

B. 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 1000 different instances and compute the average performance of algorithms over 1000 runs.

1) Clairvoyant Setting Under Synthetic Traffic: The comparison is made based on the σ-order induced by the two algorithms. For the sake of fairness, in fact, we used Greedy Flow Scheduling [12] to calculate the CCT for both ONE-PARIS and Sincronia. As a result, any difference in performance between the two solutions is solely due to the σ-order they produce. Also, we have assessed the gain in percentiles to outline how much such gain is distributed across the coflows in each workload.

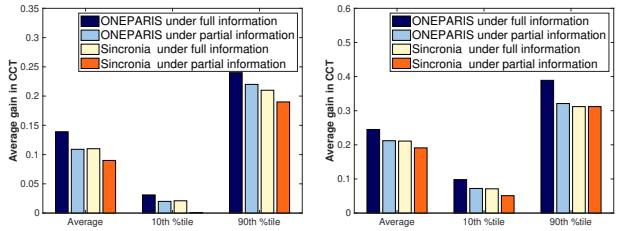
Fig. 3 shows the gains in CCT for all workloads. For workload 1 and 2, ONE-PARIS improves the CCT by $13\%-22\%$ on average over Varys while Sincronia achieves only $8.4\%-18\%$ as improvement compared to Varys. More importantly, CCT improvement covers 96% of coflows, while Sincronia degrades CCT for 30% of coflows compared to Varys. Indeed, for workload 2, where there are 80% of coflows with a single flow, Sincronia's average performance improvement is not fully consistent as 40% of the coflows experience performance degradation compared to Varys. The improvement of ONE-PARIS is even higher on a large-scale network, i.e., workloads 3 and 4. The average CCT performance gap is larger for workload 4 where, as before, coflows with a single flow are more frequent. In fact we observed that Sincronia sometimes prioritizes larger flows over smaller ones, as its scheduling is based on the bottleneck. Overall ONE-PARIS performs better than Sincronia in terms of average CCT for all percentiles. These results are expected, since ONE-PARIS can measure the cumulative impact of each coflow on all the others, whereas Sincronia is an elimination procedure which ranks coflows iteratively based only on the bottleneck. Finally, in this set of experiments, the percentile breakdown of the gain with respect to Varys demonstrates that the gain of ONE-PARIS is rather uniform across instances. Conversely, Sincronia experiences a larger variance, as reflected in the results for higher percentiles.

2) Semi-Clairvoyant Setting Under Synthetic Traffic: In this section, we discuss the results in the *semi-clairvoyant* case reported in Fig. 3 (i.e., *with partial information*). This scenario is very interesting in practice, as we have observed during the implementation of ONE-PARIS. In fact, while developing a tool to collect information about the coflow structure, we engineered a solution to gather the information on the data transfers between mappers and reducers. However, the available MapReduce APIs only allow to collect the volume of data to be transferred from each mapper to the respective set of reducers. Thus, even though we cannot access the volume per flow, it is possible to collect the total volume at each ingress port for each coflow before the shuffle phase starts and get updates along with the progress of the computing tasks.

Based on the previous observations, in the non-clairvoyant setting the σ-order is obtained by considering only the volume sent by each coflow into each ingress port. The results are shown in Fig. 3 in comparison with the full information case. It can be observed that the loss in performance for ONE-PARIS is limited and all workloads show that under the semi-clairvoyant setting performance are equivalent to those of Sincronia with full information. More precisely, semi-clairvoyant ONE-PARIS performs about $19-42\%$ better than clairvoyant Varys and 2% better than clairvoyant Sincronia. As in the clairvoyant setting, the performance of ONE-PARIS for the non-clairvoyant setting is consistent across all workloads and coflows. This shows that ONE-PARIS brings an improvement on all coflows compared to Varys, which is not the case for Sincronia and in particular when there are more coflows with just one flow.

3) Clairvoyant and Non-Clairvoyant Settings Under Real Traffic Traces: Now we evaluate the performance of ONE-PARIS using Facebook traces. As observed in the literature [4], the number of simultaneously active coflows in this trace is small. In order to reproduce a congested data center fabric, we have up-sampled the trace using our generator. The new workload contains more than 10^4 coflows and 3 million flows. For a thorough comparison, we classify coflows according to their length (i.e, size of its largest flow) and width (i.e., number of flows). As in [4], we consider 4 types of coflows Narrow&Short (N-S), Narrow&Long (N-L), Wide&Short (W-S) and Wide&Long (W-L), where a coflow is considered to be short if its length is less than 100MB and wide if it contains at least 20 flows.

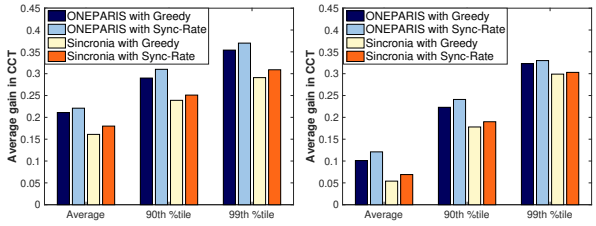
Fig. 4 shows that ONE-PARIS reduces the average CCT across all coflow types. It performs $17\%-25\%$ better than Varys especially for N-L types. For the 95th percentile, over all coflows in the N-L case, ONE-PARIS performs about 40% better than Varys, while Sincronia achieves only a 34% gain. As in previous tests, semi-clairvoyant ONE-PARIS approximates the performance of clairvoyant Sincronia and allows



(a) Facebook N&S (b) Facebook N&L

(c) Facebook W&S (d) Facebook W&L

Figure 4. Gain in CCT of ONE-PARIS and Sincronia against Varys for all coflow types in Facebook traces (with and without full information).



(a) Workload 1 (b) Workload 2

(c) Workload 3 (d) Workload 4

Figure 5. Gain in CCT of ONE-PARIS and Sincronia against Varys for Sync-Rate and Greedy.

an improvement for all coflows (more 99% of coflows in the fabric).

4) Sync-Rate Evaluation: We now evaluate Sync-Rate using previous workloads by only testing 100 instances per workload. We compared ONE-PARIS and Sincronia to Varys when either Sync-Rate or Greedy is used for rate allocation. We set RTT to 1 ms (i.e., algorithm iterations) and $\gamma = 0.02$ to satisfy the condition of Thm. 1. In Fig. 5, we observe that Sync-Rate performs better than Greedy for all workloads. The gap between Sync-Rate and Greedy is about $2.3 - 3.7\%$ on average. We have observed that Sync-Rate let most flows of a coflow finish practically at the same time. This is a desired feature to leave bandwidth for coflows of lower priority. Note

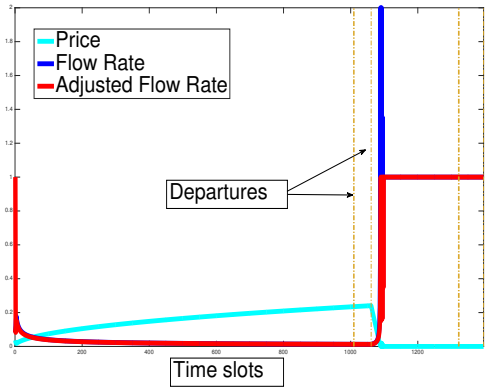


Figure 6. Progress of rates (actual and decided) and prices for flow f_{14} from example of Fig. 1 with Sync-Rate.

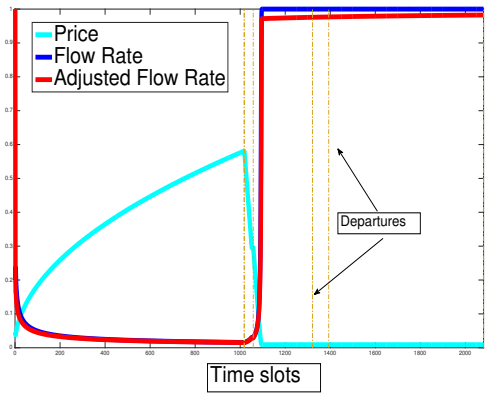


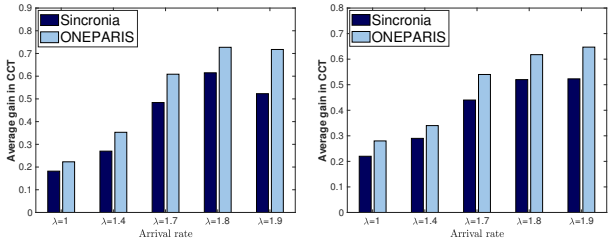
Figure 7. Progress of rates (actual and decided) and prices for flow f_{31} from example of Fig. 1 with Sync-Rate.

that Sync-Rate strives to perform flow equalization by design. Also, numerical results not reported here for the space's sake confirm that Sync-Rate has negligible loss w.r.t. the corresponding optimal solution of (9a-9c).

Fig. 6 and Fig. 7 show, for a typical run, the flow rates and prices on the example of Fig. 1. We observe that the initial rate of flow 1.4 is zero since it shares the egress port with flow 4.1, which has a high priority. When flow 4.1 is over, flow 1.4 starts at full capacity on egress ports 4 and 5. In fact, the price of flow 1.4 reacts quickly after departure of flow 4.1, granting priority. Similar behavior is observed for flow 3.1.

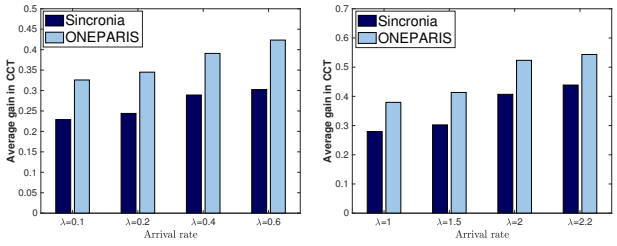
C. Online Setting

We now present a series of numerical results comparing the performance of the online version of ONE-PARIS. The results are obtained on instances generated by our workload generator and the comparison by running our flow-level simulator. We assess the performance of our algorithms against the online version of Varys and Sincronia. The online version of Varys uses a specific strategy to avoid starvation. Coflow arrivals follow a Poisson process with an average rate of λ coflows/slot. The values of λ are chosen in order to cover different trafic load in the fabric (from 0.7 to 0.99).



(a) Small fabric (b) Large fabric

Figure 8. Gain in CCT of online versions of ONE-PARIS and Sincronia against Varys using synthetic traffic with varying λ and (a) $M = 10$ and (b) $M = 100$.



(a) Small fabric (b) Large fabric

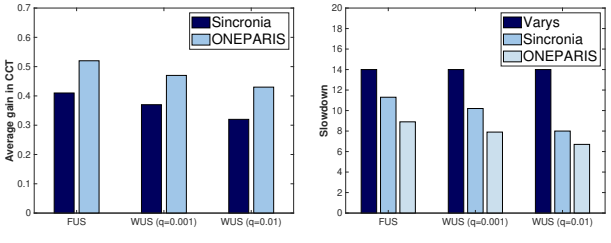
Figure 9. Gain in CCT of online versions of ONE-PARIS and Sincronia against Varys using real traffic traces with varying λ and (a) $M = 10$ and (b) $M = 100$.

For each scheduling algorithm at each input load (obtained by varying λ), we simulate 8000 coflow arrivals to get the average CCT of all completed coflows. For the sake of comparison, we have used the Greedy algorithm to calculate the CCT for all σ-order schedulers. We also consider two scenarios: a small fabric with $M = 10$ machines, and a large fabric with $M = 100$ machines.

a) Synthetic Traffic: As in the offline scenario, we consider two types of coflow; in the online setting the percentage of coflows of type 1 equals to 60% for all values of λ. We observe that ONE-PARIS achieves a higher gain in term of average CCT for all values of λ, and that the gain with respect to the other scheduling algorithms increases with the value of λ. It is worth observing that ONE-PARIS and Sincronia achieve more or less the same gain for a lightly loaded fabric, but, when the fabric is highly congested, ONE-PARIS clearly outperforms the other algorithms. Fig. 8 shows an improvement of 22–72% over Varys, whereas Sincronia achieves only 17–52%.

b) Real Traffic Traces: For real traffic traces, we evaluate ONE-PARIS against Varys and Sincronia under different values of arrival rate λ. Fig. 9 shows the gains in CCT for all values of λ. ONEPARIS improves CCT by 32–54% on average over Varys, while Sincronia achieves only 22–40%. As expected, the improvements are even more significant for large fabric and larger load.

1) Full Update versus Weighted Update Strategies: In the following, we evaluate how WUS can attain a good trade-of between efficiency and starvation in coflow scheduling. The trade-off is measured with respect to system performance (e.g., average CCT) on the one hand and individual coflow performance (e.g., isolation guarantee) on the other. Most works use static progression to avoid starvation [7]. Thus, while the static progress guarantees a minimum rate allocation per coflow, the CCT of some of the coflows can be degraded without any actual gain for the slowdown of the other ones.



(a) CCT (b) Slowdown

Figure 10. Gain in CCT and Slowdown of ONE-PARIS and Sincronia against Varys for the online setting with $\lambda = 0 : 6$.

We use the *slowdown* as a metric [35], which measures the ratio of the CCT of a coflow in the presence of other flows to the CCT when the flow is scheduled alone in the fabric. We recall that Varys has a specific strategy to avoid the starvation problem.

Fig. 10 illustrates the comparison between FUS, the strategy used in the previous online simulations, and WUF with $q = 0.001$ and $q = 0.01$. In Fig. 10(a), we observe that WUS decreases slightly the gain of ONE-PARIS compared to Varys since it tries to make a tradeoff between average CCT and coflow starvation. However, as shown in Fig. 10(b), the average slowdown is highly decreasing when SUS is used with $q = 0.01$, indicating a better trade-off.

D. Testbed Evaluation

Finally, we have built a testbed that contains 15 servers connected to a Gigabit Ethernet switch. Each server is a Dell OptiPlex 7080 that contains 4 virtual servers running Ubuntu 22.04 LTS kernel. The network fabric is then represented by 60 virtual end-hosts connected to a non-blocking Big-Switch fabric. Each end host has access link with capacity of 1 GB. The algorithms have been evaluated on both small-scale and large-scale networks, where a network is denoted by $[M, N]$ to indicate different sized fabrics and the number of coflows, N, used in the simulations. The small-scale network has a fabric of size $M = 10$, whereas the large-scale network has a fabric with either 60 machines. The coflows in these networks have been generated using real traffic traces. For ONE-PARIS and Sincronia, we used the TC (Traffic Control) in Linux by considering two different types of qdisc: ttb and prio. This allows to implement the strict priority queuing at each port which realizes the intended σ-order of coflows. In practice, this corresponds to what the Greedy algorithm does to implement bandwidth sharing. Fig. 11 shows the gain in CCT of ONE-PARIS and Sincronia when compared to the case without coflow scheduling (i.e., no QoS policy or rate control). ONE-PARIS improves the CCT by 52–187%, while Sincronia improves it by 41–151%, confirming the performance figures obtained before via simulation.

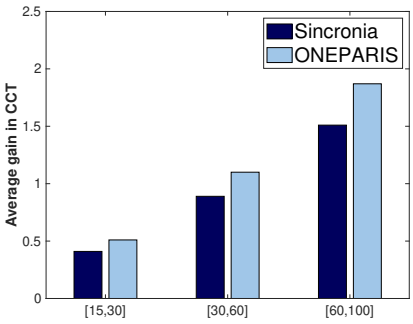


Figure 11. Gain in CCT with Facebook traces of ONE-PARIS and Sincronia against the case without coflow scheduling. Each point in the x-axis represents network $[M; N]$

VII. Related Works

Research on the scheduling of coflows has started with early research works of Chowdhury and Stoica [4], [7], [36]. There, they proved that, by accounting for the coflow structure in traffic management, they could significantly improve application-level performance. Efficient coflow scheduling methods to minimize the average CCT have been studied in the literature since then. One of the first schedulers is Varys [7], a clairvoyant scheduler under perfect information on traffic sources (e.g., engaged ports, flow volumes and flow release times). Varys 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. It then uses the Minimum-Allocation-for-Desired-Duration (MADD) procedure to calculate the bandwidth allocation per flow. Other works have been developed to reduce the average CCT of coflows [12], [35], [37]–[40]. Popular among average CCT minimization algorithms is `Sincronia`, proposed in [12]. It considers scheduling on the network bottlenecks and returns a scheduling coflow order achieving a 4–approximation factor. While the aforementioned works are efficient in reducing the CCTs of non-deadline coflows, they do not consider the effect of non-bottleneck traffic in the design of coflow schedules.

A number of efforts have also been devoted to minimizing the CCT without full prior knowledge of the coflow traffic, i.e., the total size of its constituent flows, see for example [27], [41], [42], [17], [29]. CODA [27] relies on a fully blind approach, i.e., neither routing nor flow information is available. A classifier is proposed, DBSCAN, to group coflows based on certain specific characteristics, which are showed to help scheduling coflows effectively. This work, conversely, proposes a semi-clairvoyant scheduler of the type of Aalo, [29], IAOA [17] or Elite [41]. They assume that coflow size is unknown but the information on to which flows belongs to which coflow is known. Aalo leverages on the current received size of all coflows. Similarly, IAOA [17] measures coflows volumes cumulatively online, ranks them and performs greedy rate allocation afterwards. Elite [41] extends Sincronia to cover multi-bottleneck case. FAI [42] introduces a non-clairvoyant scheduler that improves the the bottleneck performance of a coflow without affecting other coflows. Note that all of the above solutions require a centralised solution, which results in high costs to collect data from applications and end-hosts. More recently, a decentralised solution of the type proposed in this work, has been proposed for the fully blind approach in [43].

VIII. Conclusion

We presented a comprehensive semi-distributed coflow scheduling solution. It relies on a new σ-order coflow scheduler, called ONE-PARIS, that operates in the controller with full or partial information on flow volumes. Contrary to existing solutions in the literature, which mostly establish coflow priorities based on the bottleneck only, ONE-PARIS accounts for all links engaged by coflows and evaluates the priority based on the per-flow bottleneck evolution. Furthermore, to efficiently share network bandwidth, we proposed a scalable distributed rate allocation solution, called Sync-Rate, that respects the σ-order given by ONE-PARIS.

The numerical results demonstrate the significant performance improvements of ONE-PARIS with respect to existing schedulers. This behavior is observed in both offline and online settings, both for synthetic and for the real traces obtained from the Facebook data set. We have also created a testbed to evaluate ONE-PARIS in a real network environment. The performance figures we obtained demonstrate a net advantage against a standard TCP-based data transfer, and the superiority of our approach compared to state of art in coflow scheduling in a real setting.

Several extensions of this work are possible, including the integration of the coflow scheduler within a big data framework like MapReduce and the extensions of ONE-PARIS to support deadlines and fairness. In that context, an interesting direction of research is the joint optimization of coflow scheduling and workload assignment to servers to further reduce the completion time of computing jobs.

Appendix: Proof of Theorem 1

Proof. We first start to prove the convergence for the coflow with highest priority, i.e. $n = 1$, and by induction we show the convergence for all $n = 2, \ldots, K$. The core of the proof is to show that the gradient of the dual function in (11) is M_{σ_n}-Lipschitz continuous.

From the definition of $r^{\sigma_1 j}$ in (13), we can show easily that the function $r^{\sigma_1 j}(p)$ is $\frac{(b^{(\sigma_1 j)})^2}{v^{\sigma_1 j}}$-Lipschitz. On the other side, the objective function of the dual problem (11) can be rewritten as

$$D_{\sigma_1}(\vec{\lambda}^{\sigma_1}) = \sum_{j \in F_{\sigma_1}} G_{\sigma_1 j}(\vec{\lambda}^{\sigma_1}) + \sum_{l \in \mathcal{L}_1} \lambda_l^{\sigma_1} b_l, \qquad (15)$$

and the partial derivatives are determined as follows:

$$\frac{\partial D_{\sigma_1}(\vec{\lambda}^{\sigma_1})}{\partial \lambda_l^{\sigma_1}} = -\sum_{j \in F_{\sigma_1}} \frac{v^{\sigma_1 j} x_l^{\sigma_1 j}}{\sum_{l \in \mathcal{L}_1} \lambda_l^{\sigma_1} x_l^{\sigma_1 j}} + b_l \qquad (16)$$

$$= -(\sum_{j \in F_{\sigma_1}} r^{\sigma_1 j}(\vec{\lambda}^{\sigma_1}) x_l^{\sigma_1 j} - b_l). \qquad (17)$$

Thus, the gradient descent step (14) for the coflow σ_1 becomes

$$\lambda_l^{\sigma_1}(t+1) = \left[\lambda_l^{\sigma_1}(t) - \gamma_{\sigma_1} \frac{\partial D_{\sigma_1}(\vec{\lambda}^{\sigma_1})}{\partial \lambda_l^{\sigma_1}}\right]_+. \quad (18)$$

For each link $l \in \mathcal{L}$, let define the function $X_l^{\sigma 1} : \mathcal{R}^{|\mathcal{L}|} \to \mathcal{R}^+$ as

$$X_l^{\sigma 1}(\vec{\lambda}) = \sum_{j \in F_{\sigma_1}} r^{\sigma_1 j}(\vec{\lambda}) x_l^{\sigma_1 j}.$$

Then we can drive the following

$$\begin{aligned} |X_l^{\sigma_1}(\vec{\lambda}) - X_l^{\sigma_1}(\vec{\lambda}')| &= \sum_{j \in F_{\sigma_1}} |r^{\sigma_1 j}(\vec{\lambda}) - r^{\sigma_1 j}(\vec{\lambda}')| x_l^{\sigma_1 j} \\ &\leq \sum_{j \in F_{\sigma_1}} \frac{(b^{\sigma_1 j})^2}{v^{\sigma_1 j}} \left\|\vec{\lambda} - \vec{\lambda}'\right\|_1 x_l^{\sigma_1 j}, \end{aligned} \quad (19)$$

where $\|\cdot\|_1$ is the L_1 norm in vector space in $\mathcal{R}^L$. Since each flow cross only two links in the fabric, we have

$$|X_l^{\sigma_1}(\vec{\lambda}) - X_l^{\sigma_1}(\vec{\lambda}')| \leq 2 \sum_{j \in F_{\sigma_1}} \frac{(b^{\sigma_1 j})^2}{v^{\sigma_1 j}} \left\|\vec{\lambda} - \vec{\lambda}'\right\|_1 \quad (20)$$

For any $\vec{\lambda} \in \mathcal{R}^L$, we have $\left\|\vec{\lambda}\right\| \leq \left\|\vec{\lambda}\right\|_1 \leq \sqrt{L} \left\|\vec{\lambda}\right\|$, where $\|.\|$ is the L_2 norm. Thus we have the following result

$$\left\|X^{\sigma_1}(\vec{\lambda}) - X^{\sigma_1}(\vec{\lambda}')\right\| \leq 2\sqrt{L} \sum_{j \in F_{\sigma_1}} \frac{(b^{\sigma_1 j})^2}{v^{\sigma_1 j}} \left\|\vec{\lambda} - \vec{\lambda}'\right\|. \quad (21)$$

Therefore, the gradient of $D_{\sigma_1}(.)$ is K-Lipschitz where $M_{\sigma_1} = 2\sqrt{L} \sum_{j \in F_{\sigma_1}} \frac{(b^{\sigma_1 j})^2}{v^{\sigma_1 j}}$. According to the proof in [44], if the gradient of $D_{\sigma_1}(.)$ is K-Lipschitz continuous, then given a step-size $\gamma_{\sigma_1} \in (0, 2/M_{\sigma_1}]$, gradient descent step $\vec{\lambda}^{\sigma_1}(t)$ in (14) converges to the optimal solution of the dual problem D_{σ_n}. □

REFERENCES

[1] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, p. 107–113, jan 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[2] 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.

[3] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. ACM HotNets workshop*, 2012.

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

[5] Y. Liu, J. Muppala, M. Veeraraghavan, D. Lin, and M. Hamdi, *A Survey of Data Center Network Architectures*. Springer, 2013.

[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic Flow Scheduling for Data Center Networks," in *Proc. USENIX NSDI*, 2010.

[7] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proc. ACM SIGCOMM*, 2014.

[8] W. Li, J. Liu, S. Wang, T. Zhang, S. Zou, J. Hu, W. Jiang, and J. Huang, "Survey on traffic management in data center network: From link layer to application layer," *IEEE Access*, vol. 9, pp. 38 427–38 456, 2021.

[9] 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.

[10] O. Brun, R. El-Azouzi, Q.-T. Luu, F. De Pellergrini, B. J. Prabhu, and C. Richier, "Weighted scheduling of time-sensitive coflows," *arXiv preprint arXiv:2303.17175*, 2023.

[11] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Communications Surveys Tutorials*, vol. 20, no. 2, pp. 1492–1525, 2018.

[12] 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.

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

[14] L. Cheng, Y. Wang, Y. Pei, and D. Epema, "A coflow-based co-optimization framework for high-performance data analytics," in *2017 46th International Conference on Parallel Processing (ICPP)*, 2017, pp. 392–401.

[15] F. De Pellegrini, V. K. Gupta, R. El Azouzi, S. Gueye, C. Richier, and J. Leguay, "Fair coflow scheduling via controlled slowdown," *ArXiv*, vol. abs/2208.06513, 2022.

[16] 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.

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

[18] ——, "Efficient scheduling of weighted coflows in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 9, pp. 2003–2017, 2019.

[19] S. Khuller and M. Purohit, "Brief announcement: Improved approximation algorithms for scheduling co-flows," in *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 239–240. [Online]. Available: https://doi.org/10.1145/2935764.2935809

[20] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 294–303. [Online]. Available: https://doi.org/10.1145/2755573.2755592

[21] 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.

[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," *Operations Research Letters*, vol. 38, no. 5, pp. 390–395, 2010.

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

[24] R. Mao, V. Aggarwal, and M. Chiang, "Stochastic non-preemptive coflow scheduling with time-indexed relaxation," in *IEEE INFOCOM WKSHPS*, 2018.

[25] T. Zhang, F. Ren, R. Shu, and B. Wang, "Scheduling coflows with incomplete information," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–10.

[26] Y. Gao, H. Yu, S. Luo, and S. Yu, "Information-agnostic coflow scheduling with optimal demotion thresholds," in *2016 IEEE International Conference on Communications (ICC)*, 2016, pp. 1–6.

[27] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark," in *Proc. ACM SIGCOMM*, 2016.

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

[29] Z. Li, J. Bi, Y. Zhang, and C. Wang, "Eaalo: Enhanced coflow scheduling without prior knowledge in a datacenter network," in *2017 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2017, pp. 1136–1141.

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

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

[32] K. J. Babiarz and F. B. Chan, "Bconfiguration guidelines for diffserv service classes," in *RFC 4594*, 2006.

[33] T. Zhang, R. Shu, Z. Shan, and F. Ren, "Distributed bottleneck-aware coflow scheduling in data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1565–1579, July 2019.

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

[35] L. Wang, W. Wang, and B. Li, "Utopia: Near-optimal coflow scheduling with isolation guarantee," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 891–899.

[36] M. Chowdhury, Z. Liu, A. Ghodsi, and I. Stoica, "{HUG}: Multi-resource fairness for correlated and elastic demands," in *13th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 16)*, 2016, pp. 407–424.

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

[38] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[39] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 11, pp. 3366–3380, 2016.

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

[41] A. Arfaoui, R. Elazouzi, 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.

[42] L. Liu, C. Gao, P. Wang, H. Huang, J. Li, H. Xu, and W. Zhang, "Bottleneck-aware non-clairvoyant coflow scheduling with fai," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 1011–1025, 2023.

[43] J. Du and K. C.-J. Lin, "Distributed in-network coflow scheduling," in *2022 IEEE 30th International Conference on Network Protocols (ICNP)*, 2022, pp. 1–11.

[44] H. Yaiche, R. Mazumdar, and C. Rosenberg, "A game theoretic framework for bandwidth allocation and pricing in broadband networks," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 667–678, 2000.