Branch-and-Benders-Cut Algorithm for the Weighted Coflow Completion Time Minimization Problem

Youcef Magnouche, Sebastien Martin, Jeremie Leguay
Huawei Technologies, France Research Center
Boulogne-Billancourt, France
{firstname.name}@huawei.com

Francesco De-Pellegrini, Rachid El-Azouzi, Cedric Richier
University of Avignon
Avignon, France
{firstname.name}@univ-avignon.fr

ABSTRACT

An ever increasing amount of data is being processed by parallel computation frameworks such as MapReduce [9] in data centers. In this context, the coflow scheduling problem aims at accelerating the completion of data processing tasks. In particular, it consists in scheduling individual flows according to their relationship at application level, i.e. their membership to *coflows*, such that the total average weighted coflow completion time is minimized. In this paper, we propose a compact mixed integer linear formulation for the problem and apply a Branch-and-Benders-Cut algorithm to efficiently solve it. On a diverse set of instances, we show that our algorithm significantly improves the CPU time computation compared to the solution of the compact model.

1 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 [9] or Spark [21]. 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* [5], 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 mappers have completed their computing tasks. The data transfer phase between mappers and reducers is called the shuffle phase and completes only when all constituent flows are over.

The research line on datacenter coflow scheduling has been initiated by the the seminal work of Chowdhury and Stoica [5, 8]. They observed that traffic management policies accounting for the coflow structure significantly improve application-level performance. Since then coflow scheduling is a mainstream topic in network traffic engineering. The main challenge lies in the fact that computing frameworks can generate simultaneously thousands of flows per job [8]. When many jobs run in parallel, network congestion occurs due to concurrent coflows.

In general, the coflow scheduling problem to minimize the Coflow Completion Time (CCT) is strongly NP-hard and exact solutions based on time-indexed MILPs (Mixed Integer Linear Programs) suffer obvious scalability issues. Furthermore, it has been proved recently that, for a related open-shop scheduling problem, approximating the optimal solution is only possible by a factor of $2-\epsilon$, for any $\epsilon > 0$ [4]. In other words, this renders not viable to provide tight approximations of the optimal solution in a short amount of time. In fact, to date, the best deterministic approximation ratio equals to 4 (and 5 if coflows have release times) [1, 19, 20]. Even schedulers relying on relaxation of time-indexed programs face issues related to the number of variables [4, 15, 19]. Several types of schedulers have been proposed [4] in the literature. Clairvoyant schedulers work under perfect information on traffic sources, i.e., engaged ports, flow volumes and flow release times. Semi-clairvoyant schedulers [23] are robust to the lack of information, in particular on exact flow volumes. Non-clairvoyant methods have been studied as well in [8, 10, 11, 22] when prior knowledge is not available, e.g., flow volumes, coflow release times or even the coflow structure per flow is unknown [22].

One popular idea appearing in many research works suggests to equalize flow transfer times per coflow [6] 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 [17]. Varys [7] is a baseline reference for clairvoyant heuristics. Even though recent scheduling algorithms like Sincronia [1] have better performance, 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 *bottleneck* link of a coflow, that is the link 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 alone in the network). Hence, its traffic on the bottleneck link is served in priority, possibly pre-empting lower-priority coflows.

Sincronia [1, 20] resorts to a related primal-dual problem and provides a greedy algorithm which is a $(2 - \frac{2}{n+1})$ approximation for the primal problem. It relies on the results for concurrent open shop scheduling on parallel machines described in [16]. At each iteration, it computes the total load

© 2022 Copyright held by the owner/authors(s). Published in Proceedings of the 10th International Network Optimization Conference (INOC), June 7-10, 2022, Aachen, Germany. ISBN 978-3-89318-090-5 on OpenProceedings.org
Distribution of this paper is permitted under the terms of the Creative Commons license CC-by-nc-nd 4.0.

of each link and finds the link with the heaviest load. It then chooses the coflow using the Smith rule, i.e., the minimum ratio of weight and load on the bottleneck. It iterates on the unsorted coflows until a sorting of all the coflows is obtained. Once the order it decided by Sincronia, it has been shown that any work-conserving rate allocation mechanism achieves a weighted coflow completion time within 4 of the optimal as long as coflows are prioritized respecting the order.

While a number of good heuristics and approximation algorithms have already been proposed for the Weighted Coflow Completion Time Minimization (WCCTM) problem, little has been done on exact methods to attempt to solve optimally the problem at larger scale. In this context, we model the problem as an integer linear program to capture the case of a general network topology where routes can be selected, i.e., multiple routes are possible for flows of a coflow having the same source-destination pairs. Furthermore, we derive a Branch-and-Benders-Cut algorithm (BBC) to decouple the decisions for completion times (Master problem) and individual flow rates (Sub-problem). Based on completion time decisions, we show how the sub-problems can be pre-processed, by removing redundant constraints and variables. Our extensive numerical exploration indicates that the proposed Branch-and-Benders-Cut algorithm accelerates the solving time compared to the compact model.

The paper is organized as follows. Sec. 2 introduces some definitions and notations. Sec 3 presents a compact model for the WCCTM problem. Sec. 4 describes the Branch-and-Benders-Cut algorithm. Sec. 5 reports numerical results and Sec. 6 concludes the paper.

2 NOTATIONS

In this section we give some definitions and notations used throughout this paper. Consider a directed graph $G = (V, \mathcal{L})$ where V is a set of nodes and $\mathcal{L}$ is a set of arcs with capacity $b_l \in \mathbb{R}^+$, for all $l \in \mathcal{L}$. We consider that K coflows are running in parallel $\mathcal{C} = \{C_1, C_2, ..., C_K\}$. Each coflow C_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 4-tuple $(s^{kj}, d^{kj}, \mathcal{P}^{kj}, v^{kj})$ where $s^{kj}, d^{kj} \in V$ are source and destination nodes, respectively. We denote $\mathcal{P}^{kj}$ the set of paths between s^{kj} to d^{kj}, and v^{kj} the flow volume, i.e., the total amount of data to be transferred by flow f^{kj}. A coflow is considered completed only when all its flows are over and the last time when the coflow is active is called *the Coflow Completion Time (CCT) for coflow* C_k. We use w_k to denote the weight, i.e., the importance of each coflow $C_k \in \mathcal{C}$, that is typically given by the application scheduler.

3 COMPACT MODEL

We now present a time-indexed compact model for the WCCTM problem (based on the discretization of a time horizon).

Let T be a time-horizon that we partition into T_s disjoint slots of duration Δ units of time, denoted by u. Let $\mathcal{T} = \{1, \ldots, T_s\}$. The model computes the fraction of the total volume to be transferred by each flow at each time slot together with the completion time of each coflow. We suppose that b_l represents the link capacity associated with $l \in \mathcal{L}$, in Mb per time slot. For this model, three types of variables are required: $x_p^{kj}(t) \in [0,1]$, which represents the fraction of the total volume of flow f^{kj}, associated with coflow $C_k \in \mathcal{C}$ sent during time-slot $t \in \mathcal{T}$ on path $p \in \mathcal{P}^{kj}$; $y^k(t) \in \{0,1\}$, which equals 1 if time-slot $t \in \mathcal{T}$ is the final time-slot used by coflow $C_k \in \mathcal{C}$, 0 otherwise; $\gamma^k(t) \in [0,1]$, which represents the unused percentage of the final time-slot $t \in \mathcal{T}$ (the last where coflow $C_k \in \mathcal{C}$ is active). Note that for a coflow $C_k \in \mathcal{C}$, the completion time CT_k equals $\sum_{t \in \mathcal{T}} \Delta(t y^k(t) - \gamma^k(t))$. Hence the WCCTM problem writes as the following MILP:

$$\min \sum_{C_k \in \mathcal{C}} w_k \sum_{t \in \mathcal{T}} \Delta(t y^k(t) - \gamma^k(t)) \tag{1}$$

$$\sum_{t \in \mathcal{T}} y^k(t) = 1 \qquad \forall C_k \in \mathcal{C}, \tag{2}$$

$$\sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}^{kj}} x_p^{kj}(t) = 1 \qquad \forall C_k \in \mathcal{C}, \forall f^{kj} \in F_k, \tag{3}$$

$$\gamma^k(t) \leq y^k(t) \qquad \forall C_k \in \mathcal{C}, \forall t \in \mathcal{T} \tag{4}$$

$$\sum_{C_k \in \mathcal{C}} \sum_{f^{kj} \in F_k} v^{kj} \sum_{p \in \mathcal{P}^{kj} : l \in p} x^{kj}(t) \leq b_l \qquad \forall l \in \mathcal{L}, \forall t \in \mathcal{T}, \tag{5}$$

$$\sum_{t'=t}^{T_s} \Big(\sum_{p \in \mathcal{P}^{kj}} x_p^{kj}(t') - y^k(t') \Big) \leq 0 \qquad \forall C_k \in \mathcal{C}, \forall f^{kj} \in F_k, \forall t \in \mathcal{T}, \tag{6}$$

$$\sum_{f^{kj} \in F_k} v^{kj} \sum_{p \in \mathcal{P}^{kj} : l \in p} x^{kj}(t) \leq (1 - \gamma^k(t)) b_l \qquad \forall C_k \in \mathcal{C}, \forall l \in \mathcal{L}, \forall t \in \mathcal{T}, \tag{7}$$

$$0 \leq x^{kj}(t) \leq 1 \qquad \forall C_k \in \mathcal{C}, \forall f^{kj} \in F_k, \forall t \in \mathcal{T},$$

$$0 \leq \gamma^k(t) \leq 1 \qquad \forall C_k \in \mathcal{C}, \forall t \in \mathcal{T},$$

$$y^k(t) \in \{0,1\} \qquad \forall C_k \in \mathcal{C}, \forall t \in \mathcal{T}.$$

Constraints (2) select exactly one final time-slot for each coflow. Constraints (3) guarantee that all flows are served. Constraints (4) link y and γ variables. Constraints (5) represent the port capacity constraints. Constraints (6) ensure that, for every coflow, all flows are sent before the final time-slot. Finally, Constraints (7) decreases the port capacity during the final time-slot. This allows to compute the unused part of the final time-slot of each coflow.

Model (1)-(7) is composed of one family of integer variables and two families of continuous variables. The number of continuous variables is huge ($O(|\mathcal{C}| \times n^{\max} \times p^{\max} \times T_s)$ where $n^{\max} = \max_{C_k \in \mathcal{C}} \{n^k\}$ and $p^{\max} = \max_{C_k \in \mathcal{C}} \max_{f^{kj} \in F_k} \{\mathcal{P}^{kj}\}$).

(a) Coflow 0, Time Slot 0 **(b) Coflow 1, Time Slot 0**

(c) Coflow 0, Time Slot 1 **(d) Coflow 1, Time Slot 1**

Figure 1: Example of two coflows where each row represents 1 time slot of duration $1u$.

In time-indexed coflow scheduling models, setting the time slot to a smaller duration Δ provides finer rate control variables. But, since the number of controlled variables used in the MILP increases it would be preferable to set it to a larger value. An example is reported in Fig. 1 with two coflows of weight 1, each one having 1 flow of volume $1Mb$. Their traffic is sent over a single link of capacity $1Mb/u$ and the length of time horizon is 2 units of time ($2u$). Coflow 1 needs to wait until coflow 0 finishes to start sending traffic, due to capacity constraints. Therefore, coflow 0 finishes after $1u$, and coflow 1 finishes after $2u$. Then, the weighted average CCT is 1.5, which is the optimal solution. One may argue that the model should not be necessarily time-indexed discretized and only 1 time slot of duration $2u$ allows to solve the problem optimally. In this case, the total traffic of $2Mb$ (1Mb per coflow) can be sent in $1u$ as the capacity constraints are for the single time slot, leading to a weighted average CCT of 1, for a possibly unfeasible rate allocation. Thus, we obtain just a lower bound of the exact weighted average w.r.t. the CCT under a discretization with steps of 1 time units. In general, the model accuracy increases with the granularity of the time discretization, i.e., the smaller the duration of time slots, the more the model is accurate.

Also, we observe that variables $\gamma^k(t) \in [0,1]$ allows finer tuning on the effective time slot size: if coflow 2 has volume $0.5Mb$, standard models in literature predict again a weighted average CCT of 1.5 under steps of 1 time units, whereas our model correctly reports 0.75.

We illustrate how solutions of Model (1)-(7) look like, Fig. 2 shows a bubble plot where each line represents a flow, each color represents a coflow and each column represents a time slot. The size of each bubble represents the volume of traffic sent by one flow at the associated time slot. We notice that one coflow starts at time slot 0 and finishes at time slot 20. Also, we observe that the optimal solution is obtained in the class of pre-emptive schedulers, where some flows may send part of their volume during the initial time slots, whereas the remaining volume is sent later on thus prioritizing the flows of some other coflow. In the example this is the case of coflows red and orange, which complete at the end of the time horizon.

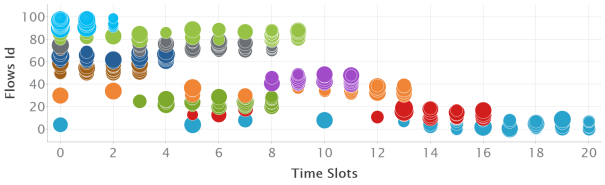

Figure 2: Example with 10 Coflows with 10 flows each. Each line represents a flow, each basic color represents a coflow.

4 BRANCH-AND-BENDERS-CUT

Benders decomposition [3] is a well-known method for solving large-scale combinatorial optimization problems [2]. It consists in decomposing the original problem into one master problem and several sub-problems. The first-stage variables are determined by solving the master problem: the sub-problems check whether they represent a feasible optimal solution. If not, new constraints, called *Benders cuts*, are added to the master problem, and the procedure is repeated. Otherwise, it stops. When sub-problems are linear programs, the approach is guaranteed to converge to an optimal solution.

The Benders decomposition [3] is, generally, effective when the number of integer variables is much smaller than the number of continuous variables. This leads to a master problem with a much smaller dimension than the original one. In the WCCTM problem, the number of coflows is typically much smaller than the number of flow per coflow, i.e., $|C| << n^k$ for all $C_k \in C$. Hence, the gap between the number of integer and continuous variables is often very high.

In this paper we develop a Branch-and-Benders-Cut (BBC) algorithm where, in contrast with the Benders decomposition that solves the master problem at every iteration, a single Branch-and-Cut tree is constructed and the Benders cuts are added during the exploration of the Branch-and-Cut tree.

Developing Branch-and-Benders-Cut algorithm for WCCTM allows to decouple the two types of variables, keeping the integer variables in the master problem, and the continuous variables in the sub-problem (1 sub-problem in our case). The master problem selects the final time-slot for every coflow and the sub-problem checks if all flows can be sent.

Consider an integer solution $y^* \in \{0,1\}^K$. The sub-problem is equivalent to the following linear program

$$\min \quad -\sum_{C_k \in C} w_j \sum_{t \in \mathcal{T}} \Delta \gamma^k(t) \tag{8}$$

$$\sum_{t \in \mathcal{T}} \sum_{p \in \mathcal{P}^{kj}} x_p^{kj}(t) = 1 \qquad \forall C_k \in C, \forall f^{kj} \in F_k, \tag{9}$$

$$\gamma^k(t) \le y^{*k}(t) \qquad \forall t \in \mathcal{T}, \forall C_k \in C. \tag{10}$$

$$\sum_{C_k \in C} \sum_{f^{kj} \in F_k} \sum_{p \in \mathcal{P}^{kj}: l \in p} v^{kj} x_p^{kj}(t) \le b_l \qquad \forall l \in \mathcal{L}, \forall t \in \mathcal{T}. \tag{11}$$

$$\sum_{t'=t}^{T_s} \Big(\sum_{p \in \mathcal{P}^{kj}} x_p^{kj}(t') - y^{*k}(t') \Big) \le 0 \qquad \forall C_k \in C, \forall f^{kj} \in F_k, \forall t \in \mathcal{T}, \tag{12}$$

$$\sum_{f^{kj} \in F_k} \sum_{p \in \mathcal{P}^{kj}: l \in p} v^{kj} x_p^{kj}(t) \le (1 - \gamma^k(t)) b_l \qquad \forall C_k \in C, \forall l \in \mathcal{L}, \forall t \in \mathcal{T}, \tag{13}$$

$$0 \le x^{kj}(t) \le 1 \qquad \forall C_k \in C, \forall f^{kj} \in F_k, \forall t \in \mathcal{T},$$

$$0 \le \gamma^k(t) \le 1 \qquad \forall C_k \in C, \forall t \in \mathcal{T}.$$

Constraints (9)-(13) are exactly the same constraints as (3)-(7) respectively, after fixing the values of y using y^*. The sub-problem can be solved in a polynomial time since all variables are continuous. The goal of the sub-problem is to check if there exists a feasible allocation of traffic that respects the completion times of all coflows given by the master.

Let $\alpha, \zeta, \sigma, \gamma, \theta$ be the dual variable vectors of the sub-problem associated with Constraints (9), (10) (11), (12) and (13), respectively. Let $z \in \mathbb{R}$ be an additional variable representing the objective value of the sub-problem, i.e., $z = -\sum_{C_k \in C} w_j \sum_{t \in \mathcal{T}} \Delta \lambda_j^t$.

The master problem consists in minimizing:

$$\sum_{C_k \in C} w_k \Delta \sum_{t \in \mathcal{T}} t y^k(t) + z$$

under Constraints (2) and the following Benders cuts:

$$\sum_{C_k \in C} \sum_{f^{kj} \in F_k} (\alpha^{kj} - \sum_{t \in \mathcal{T}} \gamma^{kj}(t) \sum_{t'=t}^{T_s} y^k(t')) - \sum_{t \in \mathcal{T}} (\sum_{l \in \mathcal{L}} \sigma_l(t) b_l + \sum_{C_k \in C} (\sum_{l \in \mathcal{L}} \theta_l^k(t) b_l + \zeta^k(t) y^k(t))) \leq z \qquad \forall (\alpha, \gamma, \sigma, \theta, \zeta), \quad (14)$$

$$\sum_{C_k \in C} \sum_{f^{kj} \in F_k} (\hat{\alpha}^{kj} - \sum_{t \in \mathcal{T}} \hat{\gamma}^{kj}(t) \sum_{t'=t}^{T_s} y^k(t')) - \sum_{t \in \mathcal{T}} (\sum_{l \in \mathcal{L}} \hat{\sigma}_l(t) b_l + \sum_{C_k \in C} (\sum_{l \in \mathcal{L}} \hat{\theta}_l^k(t) b_l + \hat{\zeta}^k(t) y^k(t))) \leq 0 \qquad \forall (\hat{\alpha}, \hat{\gamma}, \hat{\sigma}, \hat{\theta}), \quad (15)$$

$$y^k(t) \in \{0,1\} \qquad \forall C_k \in C, \forall t \in \mathcal{T}.$$

where $(\hat{\alpha}, \hat{\gamma}, \hat{\sigma}, \hat{\theta}, \hat{\zeta})$ represents the extreme rays of the sub-problem. Note that Constraints (14) are of the form $g(y) \leq z$ where $g(y)$ is the objective function associated to the dual of the sub-problem (8)-(13) and non-negativity inequalities. Constraints (15) are used in the case where the sub-problem is unfeasible and then no Constraint (14) can be generated.

4.1 Sub-problem pre-processing

Once integer decision variables y^* are fixed for the main problem, several pre-processing operations are possible to reduce the size of the sub-problem, thus reducing its computational cost. Let t_k^* be the *completion time slot* of coflow $C_k \in C$, i.e., $y^{*k}(t_k^*) = 1$. Let $t_{max}^* = \max_{C_k \in C} \{t_k^*\}$. The following results hold.

Proposition 1. *Constraint* (11) *associated with time-slot* $t \in \{t_{max}^* + 1, \ldots, T_s\}$ *and arc* $l \in \mathcal{L}$ *are redundant.*

Proof. By multiplying every constraint (12) associated with t by v^{kj} and summing all resulting constraints, we obtain

$$\sum_{C_k \in C} \sum_{f^{kj} \in F_k} \sum_{t'=t}^{T_s} \sum_{p \in \mathcal{P}^{kj}} v^{kj} x_p^{kj}(t') \leq 0 \leq b_l$$

Every constraint (11) associated with t can be obtained by summing the above inequality with the following trivial constraints

$$-v^{kj} x_p^{kj}(t') \leq 0 \quad \forall C_k \in C, f^{kj} \in F_k, p \in \mathcal{P}^{kj}, t' \in \{t+1, \ldots, T_s\}$$

and the result follows. □

Proposition 2. *For all* $C_k \in C$, *Constraints* (12) *associated with* $f^{kj} \in F_k$ *and time-slot* $t \leq t_k^*$ *are redundant.*

Proof. The upper bound of any Constraint (12) associated with time-slot $t \leq t_k^*$ is equal to 1. It follows that any Constraint (12) associated with time-slot $t \leq t_k^*$ can be obtained by summing Constraint (9) with trivial inequalities

$$-x_p^{kj}(t') \leq 0, \qquad \forall t' \leq t, \forall p \in \mathcal{P}^{kj}$$

and the result follows. □

Proposition 3. *For all* $C_k \in C$, *Constraints* (12) *associated with* $f^{kj} \in F_k$ *and time-slot* $t \in \{t_k^* + 2, \ldots, T_s\}$ *are redundant.*

Proof. Constraint (12) associated with time-slot $t \in \{t_k^* + 2, \ldots, T_s\}$ can be obtained by summing constraint (12) associated with time-slot $t_k^* + 1$ and trivial inequalities

$$-x_p^{kj}(t') \leq 0, \qquad \forall t' \in \{t_k^* + 1, \ldots, t-1\}, \forall p \in \mathcal{P}^{kj}$$

and the result follows. □

Proposition 4. *Constraints* (13) *associated with Coflow* $C_k \in C$, *time-slot* $t \in \mathcal{T} \setminus \{t_k^*\}$ *and arc* $l \in \mathcal{L}$ *are redundant.*

Proof. By constraint (10), $\gamma^k(t) = 0$ for all $t \in \mathcal{T} \setminus \{t_k^*\}$. Then Constraint (13) associated with Coflow $C_k \in C$, time-slot $t \in \mathcal{T} \setminus \{t_k^*\}$ and arc $l \in \mathcal{L}$ can by obtained by summing Constraint (11) associated with time-slot t and arc l with the following trivial constraints

$$-v^{kj} x_p^{kj}(t) \leq 0 \quad \forall C_{k'} \in C \setminus \{C_k\}, f^{kj} \in F_{k'}, p \in \mathcal{P}^{k'j}$$

and the result follows. □

Proposition 5. *For all* $C_k \in C$, *flow* $f^{kj} \in F_k$ *and path* $p \in \mathcal{P}^{kj}$, *variables* $x_p^{kj}(t)$ *associated with* $t \in \{t_{max}^* + 2, \ldots, T_s\}$ *can be removed.*

Proof. By definition, $y^{*k}(t) = 0$ for all $t \in \{t_{max}^* + 2, \ldots, T_s\}$. Therefore, by Constraints (12), $x_p^{kj}(t) = 0$ for all $C_k \in C$, flow $f^{kj} \in F_k$, path $p \leq \mathcal{P}^{kj}$ and $t \in \{t_{max}^* + 2, \ldots, T_s\}$. □

Note that, there must exists an optimal dual solution for the sub-problem where all dual variables associated with redundant constraints are equal to 0.

5 NUMERICAL RESULTS

In this section we present the numerical results obtained from the compact model and BBC algorithm. The algorithms have been implemented in C++ using Cplex 12.6 [12] as MILP-solver on a machine with Intel(R) Xeon(R) CPU E5-4627 v2of 3.30GHz with 504GB RAM, running under Linux 64 bits. A maximum of 1 thread has been used. A time limit is set to 3 hours and a memory limit for the B&B tree is set to 5Gb.

In BBC algorithm, constraints (14) and (15) are exponential in number. They are generated dynamically thanks to the *lazy constraint* callback of Cplex. Each time an integer solution is found for the master problem, the sub-problem is solved. If it is feasible, an optimality Benders cut (14) is added. Otherwise, a feasiblity Benders cut (15) is added. All Benders cuts in the master problems are separated iteratively. In order to avoid starting the Branch-and-Benders-Cut algorithm without Benders cuts, some of them can be added to the first master problem which help the convergence of the algorithm. The procedure we propose consists in supposing that all coflows finish before a tagged time slot $t \in \mathcal{T}$. Therefore, by solving the sub-problem for each $t \in \mathcal{T}$, we may generate T_s Benders cuts for the master problem.

To evaluate our algorithm, we use two types of network instances. We first consider public instances from SNDLib [18] and the Internet Topology Zoo [13] that are a mix of real (e.g., Abilene, BtEurope, Geant) and synthetic networks [1]. We also

[1] All instances with topology and traffic information will be made public here: https://github.com/MagYou/coflow-scheduling-benders

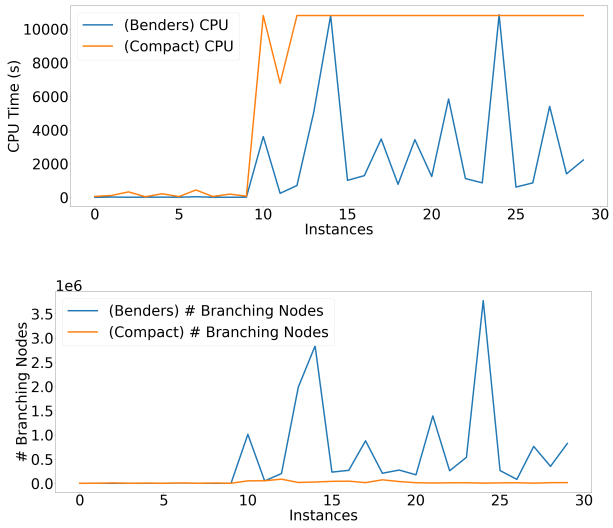

Figure 3: Numerical results on BigSwitch instances for the CPU time and the number of branching nodes.

generate instances following the standard abstraction for data center topologies, called the *Big-Switch* model [7]. This model captures the fact that congestion only occurs at Top-of-Rack (ToR) switches and that the core of the fabric is largely over-provisioned [14].

More precisely, we have considered the three following types of instances in the rest if this paper:

- Big-Switch instances, where the coflow sources are connected (by arcs) to ingress ports of a Big-Switch fabric that connects to all coflow destinations, attached to outgoing ports. We consider several instance sizes:
  - 8 ports and 64 flows : 4 coflows and 16 flows/coflow
  - 12 ports and 144 flows : 8 coflows and 18 flows/coflow
  - 14 ports and 196 flows : 7 and 28 flows/coflow
- SNDLib instances with the following traffic patterns:
  - 30 flows : 3 coflows and 10 flows/coflow
  - 50 flows : 5 coflows and 10 flows/coflow
  - 60 flows: 3 coflows and 20 flows/coflow
  - 100 flows flows: 5 coflows and 20 flows/coflow
- Internet Topology Zoo instances with 150 flows : 6 coflows and 25 flows/coflow.

The length of the horizon time is set to 30 units of times. On SNDLib and Internet Topology Zoo instances, 3 paths are computed for every flow (if they exist). All coflows have been generated randomly (sources, destinations, volumes and paths).

The following figures display the comparison of the CPU time and the number of branching nodes between the BBC algorithm and the compact model.

Fig. 3 reports on the results of Big-Switch instances. We can note that the compact model reaches the time limit on 60% of the instances while the BBC algorithm reaches it on only two instances. However the compact model generates much less branching nodes (around 19000 in average) than the BBC algorithm (around 550000 in average). This can be explained by the fact that the compact model reaches the maximum

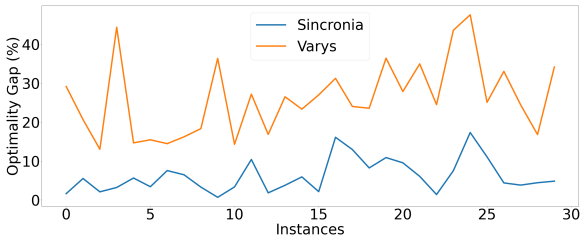

Figure 4: Optimality gaps of Varys [7] and Sincronia [1] (state of the art heuristics) on Big-Switch instances.

time limit on all instances where it has a lower number of branching nodes. Hence, the branching algorithm did not finish generating all nodes. Clearly, the Branch-and-Benders-Cut algorithm performs much better as, in practice, a small master problem is solved at each iteration (around 910 Benders cuts are generated on average at the end of the optimization).

Fig. 4 compares the optimality gaps for state of the art heuristic algorithms such as Varys [7] and Sincronia [1] on Big-Switch instances. As we can see the gap for Varys is quite high, 26.17% in average, while it is much lower for Sincronia, 6.18% in average. While the exact method based on Benders can be used in practice on some of the instances with a reasonable running time, it can also be used to evaluate heuristic algorithms for benchmarking and assess their relative performance.

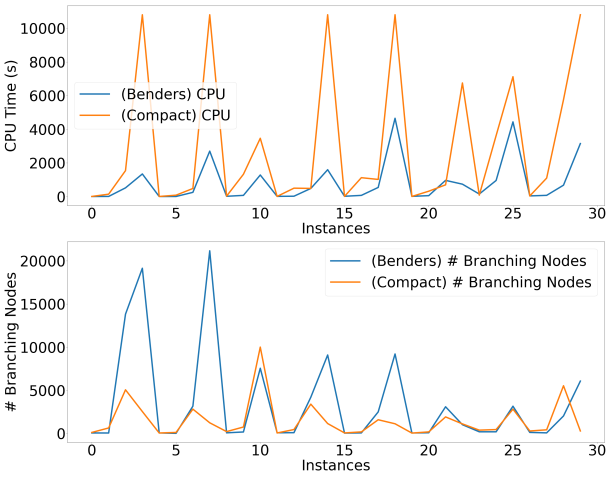

Figure 5: Numerical results on SNDLib instances for the CPU time and the number of branching nodes.

Fig. 5 shows results on SNDLib instances. We can observe that the BBC algorithm did not reach the time limit while the compact model reaches it on 5 instances. The number of branching nodes generated by the BBC algorithm is slightly higher than for the compact model (for the same reason as for Big-Switch instances) since it is around 3500 nodes on average for the BBC algorithm and around 1500 in average for the compact model. The number of Benders cuts generated is

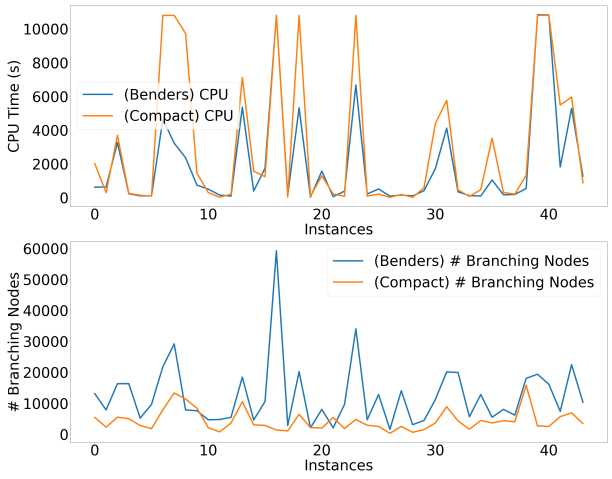

Figure 6: Numerical results on Internet Topology Zoo instances for the CPU time and the number of branching nodes.

also small over all the instances since it is around 600 cuts in average.

Finally, Fig. 6 shows the results for the Internet Topology Zoo instances with 150 flows. We can see that the BBC algorithm gives better results since the compact model reaches the time limit multiple times. Similarly to previous instances, the number of generated nodes in slightly higher for the BBC algorithm. The number of generated Benders cuts is around 2700 cuts on average.

6 CONCLUSION

In this paper, we have addressed exact solution methods for the Weighted Coflow Completion Time Minimization problem, which is NP-hard. The literature on coflows has mostly addressed heuristics and approximation algorithms, and exact solution methods are not discussed in depth to the best of the authors' knowledge. We have proposed a new compact model for the CCTM problem and a Branch-and-Benders-Cut algorithm has been developed which decouples continuous and integer variables. This decomposition permits to pre-process every sub-problem and remove a significant number of useless constraints and variables in advance. Our computational results indicate that the Branch-and-Benders-Cut algorithm for he CCTM improves significantly the CPU time compared to the compact model and thus represents an important contribution to assess the relative performance of coflow schedulers existing in the coflow literature.

REFERENCES

[1] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. 2018. Sincronia: Near-optimal network design for coflows. In Proc. ACM SIGCOMM.

[2] Simon Belieres, Mike Hewitt, Nicolas Jozefowiez, Frédéric Semet, and Tom Van Woensel. 2020. A Benders decomposition-based approach for logistics service network design. European Journal of Operational Research 286, 2 (2020), 523–537.

[3] Jacques F Benders. 1962. Partitioning procedures for solving mixed-variables programming problems. Numerische mathematik 4, 1 (1962), 238–252.

[4] Mosharaf Chowdhury, Samir Khuller, Manish Purohit, Sheng Yang, and Jie You. 2019. Near optimal coflow scheduling in networks. In The 31st ACM Symposium on Parallelism in Algorithms and Architectures. 123–134.

[5] Mosharaf Chowdhury and Ion Stoica. 2012. Coflow: A Networking Abstraction for Cluster Applications. In Proc. ACM HotNets workshop.

[6] Mosharaf Chowdhury, Matei Zaharia, Justin Ma, Michael I Jordan, and Ion Stoica. 2011. Managing data transfers in computer clusters with orchestra. ACM SIGCOMM Computer Communication Review 41, 4 (2011), 98–109.

[7] Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. 2014. Efficient Coflow Scheduling with Varys. In Proc. ACM SIGCOMM.

[8] NM Mosharaf Kabir Chowdhury. 2015. Coflow: A Networking Abstraction for Distributed Data-Parallel Applications. Ph.D. Dissertation. University of California, Berkeley.

[9] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. (2004).

[10] Fahad R. Dogar, Thomas Karagiannis, Hitesh Ballani, and Antony Rowstron. 2014. Decentralized Task-Aware Scheduling for Data Center Networks. In Proc. ACM SIGCOMM.

[11] Yuanxiang Gao, Hongfang Yu, Shouxi Luo, and Shui Yu. 2016. Information-agnostic coflow scheduling with optimal demotion thresholds. In 2016 IEEE International Conference on Communications (ICC).

[12] IBM. [n.d.]. ILOG CPLEX Solver. https://www.ibm.com/analytics/cplex-optimizer

[13] S. Knight, H.X. Nguyen, N. Falkner, R. Bowden, and M. Roughan. 2011. The Internet Topology Zoo. Selected Areas in Communications, IEEE Journal on 29, 9 (october 2011), 1765 –1775.

[14] Y. Liu, J. K. Muppala, M. Veeraraghavan, Dong Lin, and M. Hamdi. 2013. A Survey of Data Center Network Architectures. Springer.

[15] Ruijiu Mao, Vaneet Aggarwal, and Mung Chiang. 2018. Stochastic non-preemptive co-flow scheduling with time-indexed relaxation. In IEEE INFOCOM WKSHPS.

[16] Monaldo Mastrolilli, Maurice Queyranne, Andreas Schulz, Ola Svensson, and Nelson Uhan. 2010. Minimizing the sum of weighted completion times in a concurrent open shop. Oper. Res. Lett. 38 (09 2010), 390–395. https://doi.org/10.1016/j.orl.2010.04.011

[17] Mohammad Noormohammadpour and Cauligi S. Raghavendra. 2018. Datacenter Traffic Control: Understanding Techniques and Tradeoffs. IEEE Communications Surveys Tutorials 20, 2 (2018), 1492–1525.

[18] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. 2007. SNDlib 1.0–Survivable Network Design Library. In Proceedings of the 3rd International Network Optimization Conference (INOC 2007).

[19] Mehrnoosh Shafiee and Javad Ghaderi. 2018. An Improved Bound for Minimizing the Total Weighted Completion Time of Coflows in Datacenters. IEEE/ACM Transactions on Networking 26, 4 (2018), 1674–1687.

[20] Zhiliang Wang, Han Zhang, Xingang Shi, Xia Yin, Yahui Li, Haijun Geng, Qianhong Wu, and Jianwei Liu. 2019. Efficient Scheduling of Weighted Coflows in Data Centers. IEEE Transactions on Parallel and Distributed Systems 30, 9 (2019), 2003–2017.

[21] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, Ion Stoica, et al. 2010. Spark: Cluster computing with working sets. HotCloud 10, 10-10 (2010), 95.

[22] Hong Zhang, Li Chen, Bairen Yi, Kai Chen, Mosharaf Chowdhury, and Yanhui Geng. 2016. CODA: Toward Automatically Identifying and Scheduling Coflows in the Dark. In Proc. ACM SIGCOMM.

[23] Tong Zhang, Fengyuan Ren, Ran Shu, and Bo Wang. 2018. Scheduling Coflows with Incomplete Information. In 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS).