Noname manuscript No.
(will be inserted by the editor)

On Scheduling Coflows***

Saba Ahmadi · Samir Khuller · Manish Purohit · Sheng Yang

Received: date / Accepted: date

Abstract Applications designed for data-parallel computation frameworks such as MapReduce usually alternate between computation and communication stages. Coflow scheduling is a recent popular networking abstraction introduced to capture such application-level communication patterns in datacenters. In this framework, a datacenter is modeled as a single non-blocking switch with m input ports and m output ports. A coflow j is a collection of flow demands $\{d^j_{io}\}_{i\in\{1,\cdots,m\},o\in\{1,\cdots,m\}}$ that is said to be complete once *all* of its requisite flows have been scheduled.

We consider the offline coflow scheduling problem with and without release times to minimize the total weighted completion time. Coflow scheduling generalizes the well studied concurrent open shop scheduling problem and is thus NP-hard. Qiu, Stein and Zhong [15] obtain the first constant approximation algorithms for this problem via LP rounding and give a deterministic $\frac{67}{3}$-approximation and a randomized $(9+\frac{16\sqrt{2}}{3}) \approx 16.54$-approximation algorithm. In this paper, we give a combinatorial algorithm that yields a deterministic 5-approximation algorithm for coflow scheduling with release times, and a deterministic 4-approximation for the case without release time. As for concurrent open shop problem with release time, we give a combinatorial 3-approximation algorithm.

Keywords Coflow scheduling · Concurrent Open Shop

* A preliminary version of this paper appears in Proceedings of the 19th IPCO.

** This work is supported by NSF grants CNS 156019 and CCF 1655073 (Eager).

Saba Ahmadi, Samir Khuller, Sheng Yang
University of Maryland, College Park
E-mail: {saba,samir,styang}@cs.umd.edu

Google, Mountain View
E-mail: mpurohit@google.com

1 Introduction

Large scale data centers have emerged as the dominant form of computing infrastructure over the last decade. The success of data-parallel computing frameworks such as MapReduce [8], Hadoop [1], and Spark [20] has led to a proliferation of applications that are designed to alternate between computation and communication stages. Typically, the intermediate data generated by a computation stage needs to be transferred across different machines during a communication stage for further processing. For example, there is a "Shuffle" phase between every consecutive "Map" and "Reduce" phases in MapReduce. With an increasing reliance on parallelization, these communication stages are responsible for a large amount of data transfer in a datacenter. Chowdhury and Stoica [4] introduced coflows as an effective networking abstraction to represent the collective communication requirements of a job. In this paper, we consider the problem of scheduling coflows to minimize weighted completion time and give improved approximation algorithms for this basic problem.

The communication phase for a typical application in a modern data center may contain hundreds of individual flow requests, and the phase ends only when all of these flow requests are satisfied. A coflow is defined as the collection of these individual flow requests that all share a common performance goal. The underlying data center is modeled as a single $m \times m$ *non-blocking switch* that consists of m input ports and m output ports. We assume that each port has unit capacity, i.e., it can handle at most one unit of data per unit time. Modeling the data center itself as a simple switch allows us to focus solely on the scheduling task instead of the problem of *routing* flows through the network. Each coflow j is represented as a $m \times m$ integer matrix $D^j = [d^j_{io}]$ where the entry d^j_{io} indicates the number of data units that must be transferred from input port i to output port o for coflow j. Figure 1 shows a single coflow over a 2×2 switch. For instance, the coflow depicted needs to transfer 2 units of data from input a to output b and 3 units of data from input a to output d. Each coflow j also has a weight w_j that indicates its relative importance and a release time r_j.

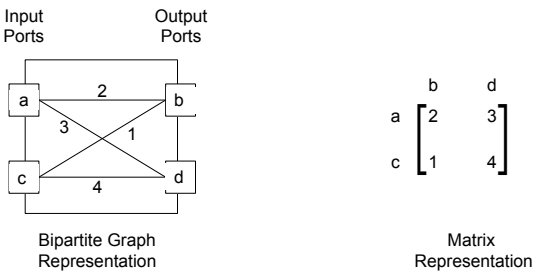

Fig. 1 An example coflow over a 2×2 switch. The figure illustrates two equivalent representations of a coflow - (i) as a weighted, bipartite graph over the set of ports, and (ii) as a $m \times m$ integer matrix.

A coflow j is available to be scheduled at its release time r_j and is said to be completed when all the flows in the matrix D^j have been scheduled. More formally, the completion time C_j of coflow j is defined as the earliest time such that for every input i and output o, d_{io}^j units of its data have been transferred from port i to port o . We assume that time is slotted and data transfer within the switch is instantaneous. Since each input port i can transmit at most one unit of data and each output port o can receive at most one unit of data in each time slot, a feasible schedule for a single time slot can be described as a matching. Our goal is to find a feasible schedule that minimizes the total weighted completion time of the coflows, i.e., minimize $\sum_j w_j C_j$.

1.1 Related Work

Chowdhury and Stoica [4] introduced the coflow abstraction to describe the prevalent communication patterns in data centers. Since then coflow scheduling has been a topic of active research [5,6,15,21] in both the systems and theory communities. Although coflow aware network schedulers have been found to perform very well in practice in both the offline [6] and online [5] settings, no $O(1)$ approximation algorithms were known even in the offline setting until recently. Since the coflow scheduling problem generalizes the well-studied concurrent open shop scheduling problem, it is NP-hard to approximate within a factor better than $(2-\epsilon)$ [2,17].

For the concurrent open shop scheduling problem an LP-relaxation yields a 2-approximation algorithm when all release times are zero [3,9,12] and a 3-approximation algorithm for arbitrary release times [9,12]. Mastrolilli et al. [14] showed a 2-approximation for concurrent open shop without release times.

For the special case when all coflows have zero release time, Qiu, Stein and Zhong [15] established the first polynomial-time constant approximation for this problem. They obtained a deterministic $\frac{64}{3}$ approximation and a randomized $(8+\frac{16\sqrt{2}}{3})$ approximation algorithm for the problem of minimizing the weighted completion time. For coflow scheduling with arbitrary release times, Qiu et al. [15] claim a deterministic $\frac{67}{3}$ approximation and a randomized $(9+\frac{16\sqrt{2}}{3})$ approximation algorithm. However in Appendix B, we demonstrate a subtle error in their proof that deals with non-zero release times. We show that their techniques in fact only yield a deterministic $\frac{76}{3}$-approximation algorithm for coflow scheduling with release times. However their result holds for the case with equal release times.

By exploiting a connection with the well-studied concurrent open shop scheduling problem, Luo et al. [13] claim a 2-approximation algorithm for coflow scheduling when all the release times are zero. Unfortunately, as we show in Appendix C, their proof is flawed and the result does not hold.

Khuller et al. [11] obtained a deterministic 12-approximation algorithm for coflow scheduling with arbitrary release times. For the special case when all release times are zero they obtained a deterministic 8-approximation and

a randomized $3 + 2\sqrt{2} \approx 5.83$-approximation. Their approach is based on reducing coflow scheduling to the concurrent open shop scheduling problem.

In independent recent work, Shafiee and Ghaderi [18] obtained the same approximation ratio, i.e. a deterministic 5-approximation algorithm with arbitrary release times, and a 4-approximation without release time. This is the same as our LP rounding results, which involves solving an LP with exponential many constraints. We go further and get a more practical primal-dual based algorithm that achieves the same approximation bound for both cases.

In recent work, Khuller et al. [10] study coflow scheduling in the online setting where the coflows arrive online over time. Using the results of this paper (Theorem 2), they obtain an exponential time 7-competitive algorithm and a polynomial time 14-competitive algorithm.

1.2 Our Contributions

The main algorithmic contribution of this paper is a deterministic, primal-dual algorithm for the offline coflow scheduling problem with improved approximation guarantees.

Theorem 1 *There exists a deterministic, combinatorial, polynomial time 5-approximation algorithm for coflow scheduling with release times.*

Theorem 2 *There exists a deterministic, combinatorial, polynomial time 4-approximation algorithm for coflow scheduling without release times.*

Our results significantly improve upon the approximation algorithms developed by Khuller and Purohit [11] whose techniques yield a 12-approximation algorithm for the case with release time, and an 8-approximation algorithm without release time. In addition, our algorithm is completely combinatorial and does not require solving a linear program. A LP-based version is also provided together with its proof, to help show the intuition behind the primal-dual one.

We also extend the primal dual algorithm by Mastrolilli et al. [14] to give a 3-approximation algorithm for the concurrent open shop problem when the jobs have arbitrary release times. Leung et al. [12] have a LP based algorithm which gives a 3-approximation as well, however our approach is the first combinatorial algorithm which achieves this bound.

Theorem 3 *There exists a deterministic, combinatorial, polynomial time 3-approximation algorithm for concurrent open shop scheduling with release times.*

1.3 Connection to Concurrent Open Shop

The coflow scheduling problem generalizes the well-studied concurrent open shop problem [3,9,12,14,19]. In the concurrent open shop problem, we have a set of m machines and each job j with weight w_j is composed of m tasks $\{t_i^j\}_{i=1}^m$, one on each machine. Let p_i^j denote the processing requirement of

task t_i^j. A job j is said to be completed once all its tasks have completed. A machine can perform at most one unit of processing at a time. The goal is to find a feasible schedule that minimizes the total weighted completion time of jobs. An LP-relaxation yields a 2-approximation algorithm for concurrent open shop scheduling when all release times are zero [3,9,12] and a 3-approximation algorithm for arbitrary release times [9,12]. Mastrolilli et al. [14] show that a simple primal-dual algorithm also yields a 2-approximation for concurrent open shop without release times. We develop a primal-dual algorithm that yields a 3-approximation for concurrent open shop with release times.

The concurrent open shop problem can be viewed as a special case of coflow scheduling when the demand matrices D^j for all coflows j are diagonal [6,15]. We use our algorithm to get a schedule for coflows and then schedule the concurrent open shop jobs in the same order. Since coflow is preemptive by definition, our algorithm gives a preemptive schedule. One might think by reducing the concurrent open shop problem to coflow scheduling we get a preemptive schedule. However for the case when all release times are zero, the schedule is automatically non-preemptive.

At first glance, it appears that coflow scheduling is much harder than concurrent open shop. For instance, while concurrent open shop always admits an optimal permutation schedule, such a property does not hold for coflows [6]. Surprisingly, we show that using a similar LP relaxation as for the concurrent open shop problem, we can design a primal dual algorithm to obtain a permutation of coflows such that sequentially scheduling the coflows after some post-processing in this permutation leads to provably good coflow schedules.

2 Preliminaries

We first introduce some notation to facilitate the following discussion. For every coflow j and input port i, we define the load $L_{i,j} = \sum_{o=1}^{m} d_{io}^j$ to be the total amount of data that coflow j needs to transmit through input port i. Similarly, we define $L_{o,j} = \sum_{i=1}^{m} d_{io}^j$ for every coflow j and output port o. Equivalently, a coflow j can be represented by a weighted, bipartite graph $G_j = (I, O, E_j)$ where the set of input ports (I) and the set of output ports (O) form the two sides of the bipartition and an edge $e = (i, o)$ with weight $w_{G_j}(e) = d_{io}^j$ represents that the coflow j requires d_{io}^j units of data to be transferred from input port i to output port o. We will abuse notation slightly and refer to a coflow j by the corresponding bipartite graph G_j when there is no confusion.

Representing a coflow as a bipartite graph simplifies some of the notation that we have seen previously. For instance, for any coflow j, the load of j on port i is simply the weighted degree of vertex i in graph G_j, i.e., if $\mathbb{N}_{G_j}(i)$ denotes the set of neighbors of node i in the graph G_j.

$$L_{i,j} = \mathsf{deg}_{G_j}(i) = \sum_{o \in \mathbb{N}_{G_j}(i)} w_{G_j}(i, o) \tag{1}$$

For any graph G_j, let $\Delta(G_j) = \max_{s \in I \cup O} \mathsf{deg}_{G_j}(s) = \max\{\max_i L_{i,j}, \max_o L_{o,j}\}$ denote the maximum degree of any node in the graph, i.e., the load on the most heavily loaded port of coflow j.

In our algorithm, we consider coflows obtained as the union of two or more coflows. Given two weighted bipartite graphs $G_j = (I, O, E_j)$ and $G_k = (I, O, E_k)$, we define the cumulative graph $G_j \cup G_k = (I, O, E_j \cup E_k)$ to be a weighted bipartite graph such that $w_{G_j \cup G_k}(e) = w_{G_j}(e) + w_{G_k}(e)$. We extend this notation to the union of multiple graphs in an obvious manner.

2.1 Scheduling a Single Coflow

Before we present our algorithm for the general coflow scheduling problem, it is instructive to consider the problem of feasibly scheduling a *single coflow* subject to the matching constraints. Given a coflow G_j, the maximum degree of any vertex in the graph $\Delta(G_j) = \max_v \mathsf{deg}_G(v)$ is an obvious lower bound on the amount of time required to feasibly schedule coflow G_j. In fact, the following lemma by Qiu et al. [15] shows that this bound is always achievable for any coflow. The proof follows by repeated applications of Hall's Theorem on the existence of perfect matchings in bipartite graphs.

Lemma 1 *[15] There exists a polynomial time algorithm that schedules a single coflow G_j in $\Delta(G_j)$ time steps.*

Lemma 1 also implicitly provides a way to decompose a bipartite graph G into two graphs G_1 and G_2 such that $\Delta(G) = \Delta(G_1) + \Delta(G_2)$. Given a time interval $(t_s, t_e]$, the following corollary uses such a decomposition to obtain a feasible coflow schedule for the given time interval by partially scheduling a coflow if necessary.

Corollary 1 *Given a sequence of coflows $G_1, G_2, \ldots, G_n$, a start time t_s, and an end time t_e such that $t_e \geq t_s + \sum_{k=1}^{j-1} \Delta(G_k)$ and $t_e < t_s + \sum_{k=1}^{j} \Delta(G_k)$, there exists a polynomial time algorithm that finds a feasible coflow schedule for the time interval $(t_s, t_e]$ such that -*

- *coflows $G_1, G_2, \ldots, G_{j-1}$ are completely scheduled.*
- *coflow G_j is partially scheduled so that $\Delta(\tilde{G}_j) = t_s + \sum_{k=1}^{j} \Delta(G_k) - t_e$ where $\tilde{G}_j$ denotes the subset of coflow j that has not yet been scheduled.*
- *coflows $G_{j+1}, \ldots, G_n$ are not scheduled.*

Proof By scheduling coflows $G_1, G_2, \ldots, G_{j-1}$ sequentially using Lemma 1, we can completely schedule these coflows by time $t_s + \sum_{k=1}^{j-1} \Delta(G_k) \leq t_e$. Similarly using Lemma 1, we find a schedule S for coflow G_j that requires $\Delta(G_j)$ time steps. We schedule only the first $t_e - (t_s + \sum_{k=1}^{j-1} \Delta(G_k))$ matchings from S after all the previous coflows have been completed. This partial scheduling of coflow G_j ends at time t_e as desired. Let $\tilde{G}_j \subset G_j$ denote the partial coflow that has not yet been scheduled. Inspecting schedule S, we observe that S schedules the partial coflow $\tilde{G}_j$ from time steps $t_e - (t_s + \sum_{k=1}^{j-1} \Delta(G_k))$ to $\Delta(G_j)$. Hence, we must have $\Delta(\tilde{G}_j) \leq t_s + \sum_{k=1}^{j} \Delta(G_k) - t_e$.

2.2 Linear Programming Relaxation

By exploiting the connection with concurrent open-shop scheduling, we adapt the LP relaxation used for the concurrent open-shop problem [9,12] to formulate the following linear program as a relaxation of the coflow scheduling problem. We introduce a variable C_j for every coflow j to denote its completion time. Let $J = \{1, 2, \ldots, n\}$ denote the set of all coflows and $M = I \cup O$ denote the set of all the ports. Figure 2 shows our LP relaxation.

$$\min \sum_{j \in J} w_j C_j$$

$$\text{subject to,} \quad C_j \geq r_j + L_{i,j} \qquad \forall j \in J, \forall i \in M \qquad (2)$$

$$\sum_{j \in S} L_{i,j} C_j \geq \frac{1}{2} \left(\sum_{j \in S} L_{i,j}^2 + (\sum_{j \in S} L_{i,j})^2 \right) \qquad \forall i \in M, \forall S \subseteq J \qquad (3)$$

Fig. 2 LP_1 for Coflow Scheduling

The first set of constraints (2) ensure that the completion time of any job j is at least its release time r_j plus the load of coflow j on any port i. The second set of constraints (3) are standard in scheduling literature (e.g. [16]) and are used to effectively lower bound completion time variables. For simplicity, we define $f_i(S)$ for any subset $S \subseteq J$ and each port i as follow

$$f_i(S) = \frac{\sum_{j \in S} L_{i,j}^2 + (\sum_{j \in S} L_{i,j})^2}{2} \qquad (4)$$

3 High Level Ideas

We use the LP above in Fig 2 and its dual to develop a combinatorial algorithm (Algorithm 1) in Section 4.1 to obtain a good permutation of the coflows. This primal dual algorithm is inspired by Davis et al. [7] and Mastrolilli et al. [14]. As we show in Lemma 5, once the coflows are permuted as per this algorithm, we can bound the completion time of a coflow j in an optimal schedule in terms of $\Delta(\bigcup_{k \leq j} G_k)$, the maximum degree of the union of the first j coflows in the permutation.

A naïve approach now would be to schedule each coflow independently and sequentially using Lemma 1 in this permutation. Since all coflows $k \leq j$ would need to be scheduled before starting to schedule j, the completion time of coflow j under such a scheme would be $\sum_{k \leq j} \Delta(G_k)$. Unfortunately, for arbitrary coflows we can have $\sum_{k \leq j} \Delta(G_k) \gg \Delta(\bigcup_{k \leq j} G_k)$. For instance, Fig 3 shows three coflows such that $\Delta(G_1) + \Delta(G_2) + \Delta(G_3) = 300 > \Delta(G_1 \cup G_2 \cup G_3) = 101$.

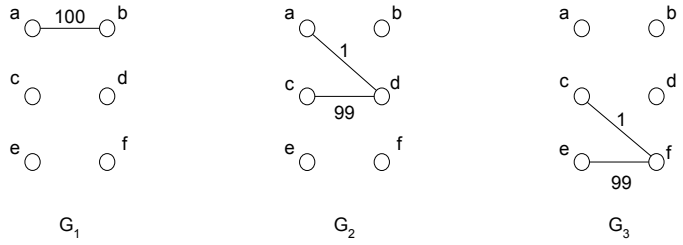

Fig. 3 Example that illustrates sequentially scheduling coflows independently can lead to bad schedules.

One key insight is that sequentially scheduling coflows one after another may waste resources, such as in Fig 3. Since the amount of time required to completely schedule a single coflow k only depends on the maximum degree of the graph G_k, if we augment graph G_k by adding edges such that its maximum degree does not increase, the augmented coflow can still be scheduled in the same time interval. This observation leads to the natural idea of "shifting" edges from a coflow j later in the permutation to a coflow k ($k < j$), so long as the release time of j is still respected, as such a shift does not delay coflow k further but may significantly reduce the requirements of coflow j. Consider for instance the coflows in Figure 3 when all release times are zero; shifting the edge (c, d) from graph G_2 to G_1 and the edges (e, f) and (c, f) from G_3 to G_1 leaves $\Delta(G_1)$ unchanged but drastically reduces $\Delta(G_2)$ and $\Delta(G_3)$. Let's call the coflows after shifting edges G_1', G_2', G_3'. After moving edges, $\Delta(G_1') = 100$, $\Delta(G_2') = 1$ and $\Delta(G_3') = 0$. If we schedule G_1', G_2', G_3' sequentially, completion time of G_1 (C_1) will be $\Delta(G_1') = 100$, $C_2 = \Delta(G_1') + \Delta(G_2') = 101$ and $C_3 = \Delta(G_1') = 100$. However before shifting edges completion times were $C_1 = 100, C_2 = 200, C_3 = 300$. Thus it could be seen that shifting edges reduces completion times.

In Algorithm 3 in Section 4.2, we formalize this notion of shifting edges and prove that after all such edges have been shifted, sequentially scheduling the augmented coflows leads to provably good coflow schedules.

In Section 6 we present an alternative approach using LP Rounding for finding a good permutation of coflows. Then we schedule the coflows using Algorithm 3 and give alternative proofs for Theorem 1 and Theorem 2.

4 Approximation Algorithm for Coflow Scheduling with Release Times

In this section we present a combinatorial 5-approximation algorithm for minimizing the weighted sum of completion times of a set of coflows with release times. Our algorithm consists of two stages. In the first stage, we design a primal-dual algorithm to find a good permutation of the coflows. In the sec-

ond stage, we show that scheduling the coflows sequentially in this ordering after some postprocessing steps yields a provably good coflow schedule.

4.1 Finding a Permutation of Coflows Using a Primal Dual Algorithm

Although our algorithm does not require solving a linear program, we use the linear program in Figure 2 and its dual (Figure 4) in the design and analysis of the algorithm.

$$
\begin{aligned}
& \max \sum_{j \in J} \sum_{i \in M} \alpha_{i,j}(r_j + L_{i,j}) + \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S) && \\
\text{subject to,} \quad & \sum_{i \in M} \alpha_{i,j} + \sum_{i \in M} \sum_{S/j \in S} L_{i,j} \beta_{i,S} \leq w_j && \forall j \in J \\
& \alpha_{i,j} \geq 0 && \forall j \in J, i \in M \\
& \beta_{i,S} \geq 0 && \forall i \in M, \forall S \subseteq J
\end{aligned}
$$

Fig. 4 Dual of LP_1

Our algorithm works as follows. We build up a permutation of the coflows in reverse order iteratively. Let κ be a constant that we specify later. Let J be the set of unscheduled jobs, initially $J = \{1, 2, \cdots, n\}$. In any iteration, let j be the unscheduled job with the latest release time, let μ be the port with the highest load and let L_μ be the load on port μ. Now if $r_j > \kappa L_\mu$, we raise the dual variable $\alpha_{\mu,j}$ until the corresponding dual constraint is tight and place coflow j to be last in the permutation. But if $r_j \leq \kappa L_\mu$, then we raise the dual variable $\beta_{\mu,J}$ until the dual constraint for some job j' becomes tight and place coflow j' to be last in the permutation. Algorithm 1 gives the formal description of the complete algorithm.

4.2 Scheduling Coflows According to a Permutation

We assume without loss of generality that the coflows are ordered based on the permutation given by Algorithm 1, i.e. $\sigma(j) = j$.

As we discussed in Section 3, naïvely scheduling the coflows sequentially in this order may not be a good idea. However, by appropriately moving edges from a coflow j to an earlier coflow k ($k < j$), we can get a provably good schedule. The crux of our algorithm lies in the subroutine MoveEdgesBack defined in Algorithm 2.

Given two bipartite graphs G_k and G_j ($k < j$), MoveEdgesBack greedily moves weighted edges from graph G_j to G_k so long as the maximum degree of graph G_k does not increase. The key idea behind this subroutine is that since

Algorithm 1: Permuting Coflows

J is the set of unscheduled jobs and initially $J = \{1, 2, \cdots, n\}$;
Initialize $\alpha_{i,j} = 0$ for all $i \in M, j \in J$ and $\beta_{i,S} = 0$ for all $i \in M, S \subseteq J$;
$L_i = \sum_{j \in J} L_{ij}\ \forall i \in M$; `// load of port` i
for $k = n, n-1, \cdots, 1$ **do**
  $\mu(k) = \arg\max_{i \in M} L_i$; `// determine the port with highest load`
  $j = \arg\max_{\ell \in J} r_\ell$; `// determine job that released last`
  if $r_j > \kappa \cdot L_{\mu(k)}$ **then**
    $\alpha_{\mu(k),j} = (w_j - \sum_{i \in M} \sum_{S \ni j} L_{i,j} \beta_{i,S})$;
    $\sigma(k) \leftarrow j$;
  end
  else if $r_{\sigma(k)} \leq \kappa \cdot L_{\mu(k)}$ **then**
    $j' = \arg\min_{j \in J} \left(\frac{w_j - \sum_{i \in M} \sum_{S \ni j} L_{i,j} \beta_{i,S}}{L_{\mu(k),j}} \right)$;
    $\beta_{\mu(k),J} = \left(\frac{w_{j'} - \sum_{i \in M} \sum_{S \ni j'} L_{i,j'} \beta_{i,S}}{L_{\mu(k),j'}} \right)$;
    $\sigma(k) \leftarrow j'$;
  end
  $J \leftarrow J \setminus \sigma(k)$;
  $L_i \leftarrow L_i - L_{i,\sigma(k)}, \forall i \in M$;
end
Output permutation $\sigma(1), \sigma(2), \cdots, \sigma(n)$;

Algorithm 2: The MoveEdgesBack subroutine.

Function *MoveEdgesBack*(G_k, G_j)
  for $e = (u, v) \in G_j$ **do**
    $\delta = \min(\Delta(G_k) - \deg_{G_k}(u), \Delta(G_k) - \deg_{G_k}(v), w_{G_j}(e))$;
    $w_{G_j}(e) = w_{G_j}(e) - \delta$;
    $w_{G_k}(e) = w_{G_k}(e) + \delta$;
  end
  return G_k, G_j;

the coflow k requires $\Delta(G_k)$ time units to be scheduled feasibly, the edges moved back can now also be scheduled in those $\Delta(G_k)$ time units for "free".

If all coflows have zero release times, then we can safely move edges of a coflow G_j to any G_k such that $k < j$. However, with the presence of arbitrary release times, we need to ensure that edges of coflow G_j do not violate their release time, i.e. they are scheduled only after they are released. Algorithm 3 describes the pseudo-code for coflow scheduling with arbitrary release times. Here q denote the number of distinct values taken by the release times of the n coflows. Further, let $t_1 < t_2 < \ldots < t_q$ be the ordered set of the release times. For simplicity, we define $t_{q+1} = T$ as a sufficiently large time horizon.

At any time step t_i, let $G'_j \subseteq G_j$ denote the subgraph of coflow j that has not been scheduled yet. We consider every ordered pair of coflows $k < j$ such that both the coflows have been released and MoveEdgesBack from graph G'_j to graph G'_k. Finally, we begin to schedule the coflows sequentially in order

using Corollary 1 until all coflows are scheduled completely or we reach time t_{i+1} when a new set of coflows gets released and the process repeats.

Algorithm 3: Coflow Scheduling

```
1  q ← number of distinct release times;   t_{q+1} ← T;
2  t_1, t_2, ..., t_q ← distinct release time in increasing order ;
3  for i = 1, 2, ..., q do
4  |  // Each loop finds a schedule for time interval (t_i, t_{i+1}]
5  |  for j = 1, 2, ..., n do
6  |  |  G'_j ← unscheduled part of G_j;
7  |  end
8  |  for k = 1, 2, ..., n − 1 do
9  |  |  if r_k ≤ t_i then
10 |  |  |  for j = k + 1, ..., n do
11 |  |  |  |  if r_j ≤ t_i then G'_k, G'_j ← MoveEdgesBack(G'_k, G'_j) ;
12 |  |  |  end
13 |  |  end
14 |  end
15 |  Schedule (G'_1, G'_2, ..., G'_n) in (t_i, t_{i+1}] using Corollary 1;
16 end
```

5 Analysis

We first analyze Algorithm 3 and upper bound the completion time of a coflow j in terms of the maximum degree of the cumulative graph obtained by combining the first j coflows in the given permutation. For simplicity, we first state the proof when all release times are zero, then proceed to the case with non-zero release time

5.1 Coflows with Zero Release Times

For ease of presentation we first analyze the special case when all coflows are released at time zero. In this case, we have $q = 1$ in Algorithm 3 and thus the outer *for* loop is only executed once. The following lemma shows that after the MoveEdgesBack subroutine has been executed on every ordered pair of coflows, for any coflow j, the sum of maximum degrees of graphs G'_k $(k \leq j)$ is at most twice the maximum degree of the cumulative graph obtained by combining the first j coflows.

Lemma 2 *For all* $j \in \{1, 2, \ldots n\}$, $\sum_{k \leq j} \Delta(G'_k) \leq 2\Delta(\bigcup_{k \leq j} G_k)$.

Proof Since the graphs G'_k keep changing during the course of the algorithm, for the sake of analysis, let $G_{k|j}$ where $k < j$ be the state of the graph G'_k immediately after we have transferred all possible edges from G'_j to G'_k. Let

$G_{j|j}$ denote the graph G'_j after all possible edges have been moved to G'_{j-1}. Since we move edges back to a graph G'_k only if it does not increase the maximum degree, we have the following.

$$\Delta(G'_k) = \Delta(G_{k|j}) \text{ for all } k \leq j. \tag{5}$$

For any $j \in \{1, 2, \ldots, n\}$, consider the set S of graphs $G_{1|j}, G_{2|j}, \ldots G_{j|j}$. Let u be a vertex of maximum degree in $G_{j|j}$, i.e. $deg_{G_{j|j}}(u) = \Delta(G_{j|j})$ and consider any edge $e = (u, v)$ incident on u in $G_{j|j}$. Since edge (u, v) was not moved to any of the graphs $G_{k|j}$ for $k < j$, we must have that either u or v had maximum degree in $G_{k|j}$. Let $S_u = \{G_{k|j} \mid deg_{G_{k|j}}(u) = \Delta(G_{k|j})\}$ and $S_v = \{G_{k|j} \mid deg_{G_{k|j}}(v) = \Delta(G_{k|j})\}$ denote the subsets of the graph where vertex u or v has the maximum degree respectively.

Now, let $\hat{G}_j = \bigcup_{k=1}^{j} G_{k|j}$ be the union of the graphs $G_{k|j}$. Since $\hat{G}_j$ contains all edges from the graphs $G_1, \ldots, G_j$ and no edges from graphs G_l for $l > j$, $\hat{G}_j$ is identical to the cumulative graph of the first j coflows. In particular, we have the following.

$$\Delta(\hat{G}_j) = \Delta(\bigcup_{k \leq j} G_k) \tag{6}$$

Let us now consider the maximum degree of the graph $\hat{G}_j$.

$$\Delta(\hat{G}_j) \geq \max\left\{\mathsf{deg}_{\hat{G}_j}(u), deg_{\hat{G}_j}(v)\right\} \tag{7}$$

$$\geq \max\left\{\sum_{G \in S_u} \mathsf{deg}_G(u), \sum_{G \in S_v} \mathsf{deg}_G(v)\right\} \tag{8}$$

$$= \max\left\{\sum_{G \in S_u} \Delta(G), \sum_{G \in S_v} \Delta(G)\right\} \tag{9}$$

From Equation (5), we have the following.

$$\sum_{k \leq j} \Delta(G'_k) = \sum_{k \leq j} \Delta(G_{k|j}) = \sum_{G \in S} \Delta(G) \tag{10}$$

However, since $S_u \cup S_v = S$ as either u or v has maximum degree in every graph in S, we get the following.

$$\sum_{k \leq j} \Delta(G'_k) \leq 2 \max\left\{\sum_{G \in S_u} \Delta(G), \sum_{G \in S_v} \Delta(G)\right\} \leq 2\Delta(\hat{G}_j) = 2\Delta(\bigcup_{k \leq j} G_k)$$

where the last equality follows from Equation (6).

Lemma 3 *Consider any coflow j and let $C_j(alg)$ denote the completion time of coflow j when scheduled as per Algorithm 3. Then $C_j(alg) \leq 2\Delta(\bigcup_{k \leq j} G_k)$.*

Proof Let $G'_1, \ldots, G'_n$ denote the coflows after all the edges have been moved backward. According to Lemma 1 each coflow G'_k could be finished at time $\Delta(G'_k)$, thus when the coflows are scheduled sequentially, we get the following.

$$C_j(alg) = \sum_{k \leq j} \Delta(G'_k) \leq 2\Delta(\bigcup_{k \leq j} G_k)$$

where the last inequality follows from Lemma 2.

5.2 Coflows with Arbitrary Release Times

When the coflows have arbitrary release times, we can bound the completion time of each coflow j in terms of the maximum degree of the cumulative graph obtained by combining the first j coflows and the largest release time of all the jobs before j in the permutation.

Lemma 4 *For any coflow j, let $C_j(alg)$ denote the completion time of coflow j when scheduled as per Algorithm 3. Then $C_j(alg) \leq \max_{k \leq j} r_k + 2\Delta(\bigcup_{k \leq j} G_k)$*

Proof Consider any coflow j. Let $t_i = \max_{l \leq j} r_l$ denote the earliest time when all coflows in the set $\{1, 2, \ldots, j\}$ have been released. In Algorithm 3, consider the i^{th} iteration of the for loop. Let $G_{k,i}$ denote the graph corresponding to coflow k in iteration i before edges have been moved back, i.e., $G_{k,i}$ denotes the state of coflow k in iteration i after line 7. Since some edges from coflow k may have already been scheduled in earlier iterations, we have $G_{k,i} \subseteq G_k$. Let $G'_{k,i}$ denote the graph corresponding to coflow k after the MoveEdgesBack subroutines have been executed, i.e. at line 14. We now claim that

$$C_j(alg) \leq t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \tag{11}$$

If $t_{i+1} \geq t_i + \sum_{k \leq j} \Delta(G'_{k,i})$, Corollary 1 guarantees that coflows $1 \leq k \leq j$ will be completely scheduled sequentially in this iteration. Completion time of coflow j is thus $t_i + \sum_{k \leq j} \Delta(G'_{k,i})$ as desired.

On the other hand, if $t_{i+1} < t_i + \sum_{k \leq j} \Delta(G'_{k,i})$, let p denote the first coflow such that $t_{i+1} < t_i + \sum_{k \leq p} \Delta(G'_{k,i})$. Corollary 1 now finds feasible schedules for time slots t_i to t_{i+1} such that all coflows $k \leq p-1$ are completely scheduled and coflow p is partially scheduled so that we have the following.

$$\Delta(G'_{p,i+1}) = \Delta(G_{p,i+1}) = t_i + \sum_{k \leq p} \Delta(G'_{k,i}) - t_{i+1} \tag{12}$$

$$\Delta(G'_{k,i+1}) = \Delta(G_{k,i+1}) = 0, \forall k \leq p-1 \tag{13}$$

Also, since all the coflows $1 \leq k \leq j$ had already been released at time t_i, any new coflows that get released do not affect the movement of edges from graphs corresponding to coflows $1 \leq k \leq j$. Hence, we have -

$$\Delta(G'_{k,i+1}) = \Delta(G'_{k,i}), \forall p < k \leq j \tag{14}$$

From equations (12) - (14), we get

$$t_{i+1} + \sum_{k \leq j} \Delta(G'_{k,i+1}) = t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \tag{15}$$

Proceeding this way inductively, we obtain

$$t_{i+x} + \sum_{k \leq j} \Delta(G'_{k,i+x}) = t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \tag{16}$$

where $i + x$ is the last iteration such that $t_{i+x} < t_i + \sum_{k \leq j} \Delta(G'_{k,i})$. By Corollary 1 at the end of iteration $i + x$, coflow j is completely scheduled at time $t_{i+x} + \sum_{k \leq j} \Delta(G'_{k,i+x}) = t_i + \sum_{k \leq j} \Delta(G'_{k,i})$ as desired, thus completing the proof of the claim.

We can now bound $C_j(alg)$ as follows.

$$C_j(alg) \leq t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \leq t_i + 2\Delta\left(\bigcup_{k \leq j} G_{k,i}\right) \leq t_i + 2\Delta\left(\bigcup_{k \leq j} G_k\right) \tag{17}$$

where the second inequality follows from Lemma 2.

5.3 Analyzing the Primal-Dual Algorithm

We are now in a position to analyze Algorithm 1. Recall that we assume that the jobs are sorted as per the permutation obtained by Algorithm 1, i.e., $\sigma(k) = k, \forall k \in [n]$. We first give a lemma,

Lemma 5 *If there is an algorithm that generates a feasible coflow schedule such that for any coflow j, $C_j(alg) \leq a \max_{k \leq j} r_k + b\Delta(\bigcup_{k \leq j} G_k)$ for some constants a and b, then the total cost of the schedule is bounded as follows.*

$$\sum_j w_j C_j(alg) \leq (a + \frac{b}{\kappa}) \sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j + 2(a\kappa + b) \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S)$$

Theorem 1 *There exists a deterministic, combinatorial, polynomial time 5-approximation algorithm for coflow scheduling with release times.*

Proof For scheduling coflows with arbitrary release times, Lemmas 4 and 5 (with $a = 1$ and $b = 2$) together imply that -

$$\sum_j w_j C_j(alg) \leq \left(1 + \frac{2}{\kappa}\right) \sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j + 2(\kappa + 2) \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S)$$

To minimize the approximation ratio, we substitute $\kappa = \frac{1}{2}$ and obtain

$$\sum_j w_j C_j(alg) \leq 5 \left(\sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j + \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S) \right) \leq 5 \cdot OPT$$

where the last inequality follows from weak duality as α and β constitute a feasible dual solution.

Theorem 2 *There exists a deterministic, combinatorial, polynomial time 4-approximation algorithm for coflow scheduling without release times.*

Proof Lemmas 4 and 5 (with $a = 0$ and $b = 2$) together imply that -

$$\sum_j w_j C_j(alg) \leq \frac{2}{\kappa} \sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j + 4 \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S)$$

To minimize the approximation ratio, we substitute $\kappa = \frac{1}{2}$ and obtain

$$\sum_j w_j C_j(alg) \leq 4 \left(\sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j + \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S) \right) \leq 4 \cdot OPT$$

where the last inequality follows from weak duality as α and β constitute a feasible dual solution.

5.4 Primal Dual Analysis

We devote this section to prove Lemma 5.

Recall that we assume that the jobs are sorted as per the permutation obtained by Algorithm 1, i.e., $\sigma(k) = k, \forall k \in [n]$.

Let S_j be the set of jobs $\{1, \cdots, j\}$. Let $\beta_{i,j} = \beta_{i,S_j}$ and $L_i(S_j) = \sum_{k \leq j} L_{i,k}$. Also let $\mu(j)$ define the port with highest load in S_j, therefore $L_{\mu(j)}(S_j) = \sum_{k \leq j} L_{\mu(j),k} = \Delta(\bigcup_{k \leq j} G_k)$. We will first state a few observations regarding the primal-dual algorithm.

Observation 1 *The following statements hold.*

(a) Every nonzero $\beta_{i,S}$ can be written as $\beta_{\mu(j),j}$ for some job j.
(b) For every set S_j that has a nonzero $\beta_{\mu(j),j}$ variable, if $k \leq j$ then $r_k \leq \kappa \cdot L_{\mu(j)}(S_j)$.
(c) For every job j that has a nonzero $\alpha_{\mu(j),j}$, $r_j > \kappa \cdot L_{\mu(j)}(S_j)$.
(d) For every job j that has a nonzero $\alpha_{\mu(j),j}$, if $k \leq j$ then $r_k \leq r_j$.

The correctness of Observation 1 can be directly obtained from Algorithm 1.

Lemma 6 *For every job j, $\sum_{i\in M}\alpha_{i,j}+\sum_{i\in M}\sum_{k\geq j}L_{i,j}\beta_{i,k}=w_j$.*

Proof A job j is added to the permutation in Algorithm 1 only if the constraint $\sum_{i\in M}\alpha_{i,j}+\sum_{i\in M}\sum_{S/j\in S}L_{i,j}\beta_{i,S}\leq w_j$ gets tight for this job, thus:

$$\sum_{i\in M}\alpha_{i,j}+\sum_{i\in M}\sum_{S/j\in S}L_{i,j}\beta_{i,S}=w_j$$
$$\sum_{i\in M}\alpha_{i,j}+\sum_{i\in M}\sum_{k\geq j}L_{i,j}\beta_{i,k}=w_j$$

Observation 2 *For any $i\in M$ and $S\subseteq J$, we have that $(\sum_{j\in S}L_{i,j})^2\leq 2f_i(S)$.*

Lemma 5 *If there is an algorithm that generates a feasible coflow schedule such that for any coflow j, $C_j(alg)\leq a\max_{k\leq j}r_k+b\Delta(\bigcup_{k\leq j}G_k)$ for some constants a and b, then the total cost of the schedule is bounded as follows.*

$$\sum_j w_jC_j(alg)\leq(a+\frac{b}{\kappa})\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,j}r_j+2(a\kappa+b)\sum_{i\in M}\sum_{S\subseteq J}\beta_{i,S}f_i(S)$$

Proof In the following we denote $C_j(alg)$ as C_j for ease of notation. By applying Lemma 6:

$$\sum_{j=1}^{n}w_j\cdot C_j=\sum_{j=1}^{n}\left(\sum_{i\in M}\alpha_{i,j}+\sum_{i\in M}\sum_{k\geq j}L_{i,j}\beta_{i,k}\right)\cdot C_j$$
$$=\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,j}\cdot C_j+\sum_{j=1}^{n}\sum_{i\in M}\sum_{k\geq j}L_{i,j}\beta_{i,k}\cdot C_j$$

First let's bound $\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,j}\cdot C_j$. Since $\Delta(\bigcup_{k\leq j}G_k)=L_{\mu(j)}(S_j)$:, by applying Observation 1 parts (c), (d), we get

$$\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,j}\cdot C_j$$
$$\leq\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,j}\left\{a\cdot\max_{\ell\leq j}r_\ell+b\cdot L_{\mu(j)}(S_j)\right\}$$
$$\leq\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,j}\left(a\cdot r_j+b\cdot\frac{r_j}{\kappa}\right)$$
$$\leq\left(a+\frac{b}{\kappa}\right)\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,j}r_j$$

Now we bound $\sum_{j=1}^{n}\sum_{i\in M}\sum_{k\geq j} L_{i,j}\beta_{i,k}C_j$:

$$\begin{aligned}
&\sum_{j=1}^{n}\sum_{i\in M}\sum_{k\geq j} L_{i,j}\beta_{i,k}C_j \\
\leq &\sum_{j=1}^{n}\sum_{i\in M}\sum_{k\geq j} L_{i,j}\beta_{i,k}\cdot\left\{a\cdot max_{\ell\leq j}r_\ell + b\cdot L_{\mu(j)}(S_j)\right\} \\
\leq &\sum_{j=1}^{n}\sum_{i\in M}\sum_{k\geq j} L_{i,j}\beta_{i,k}\cdot\left\{a\cdot max_{\ell\leq k}r_\ell + b\cdot L_{\mu(j)}(S_j)\right\}
\end{aligned}$$

By applying Observation 1 part (b):

$$\begin{aligned}
&\leq \sum_{j=1}^{n}\sum_{i\in M}\sum_{k\geq j} L_{i,j}\beta_{i,k}\cdot\left\{a\kappa\cdot L_{\mu(k)}(S_k) + b\cdot L_{\mu(j)}(S_j)\right\} \\
&\leq (a\kappa+b)\sum_{k=1}^{n}\sum_{i\in M}\sum_{j\leq k} L_{i,j}\beta_{i,k}\cdot L_{\mu(k)}(S_k) \\
&\leq (a\kappa+b)\sum_{k=1}^{n}\sum_{i\in M}\beta_{i,k}\sum_{j\leq k} L_{i,j}\cdot L_{\mu(k)}(S_k) \\
&= (a\kappa+b)\sum_{k=1}^{n}\sum_{i\in M}\beta_{i,k}\left(L_i(S_k)\right)\cdot L_{\mu(k)}(S_k) \\
&\leq (a\kappa+b)\sum_{i\in M}\sum_{k=1}^{n}\beta_{i,k}\left(L_{\mu(k)}(S_k)\right)^2
\end{aligned}$$

By sequentially applying Observation 2 and Observation 1 part (a), this is upper bounded by

$$\begin{aligned}
&2(a\kappa+b)\sum_{i\in M}\sum_{k=1}^{n}\beta_{i,k}f_{\mu(k)}(S_k) \\
=&2(a\kappa+b)\sum_{k=1}^{n}\beta_{\mu(k),k}f_{\mu(k)}(S_k) \\
\leq&2(a\kappa+b)\sum_{i\in M}\sum_{S\subseteq J}\beta_{i,S}f_i(S)
\end{aligned}$$

Therefore,

$$\sum_{j\in J} w_jC_j \leq \left(a+\frac{b}{\kappa}\right)\sum_{j=1}^{n}\sum_{i\in M}\alpha_{i,\sigma(j)}r_j + 2(a\kappa+b)\sum_{i\in M}\sum_{S\subseteq J}\beta_{i,S}f_i(S)$$

6 An Alternative Approach Using LP Rounding

This alternative approach also consists of two stages. First, we find a good permutation of coflows and after that we schedule the coflows sequentially in this ordering using Algorithm 3.

Let $\overline{C_j}$ denote the completion time of job j in an optimal $\mathbf{LP_1}$ solution. We assume without loss of generality that the coflows are ordered so that the following holds.

$$\overline{C_1} \leq \overline{C_2} \leq \ldots \leq \overline{C_n} \tag{18}$$

We can use the LP-constraints to provide a lower bound on $\overline{C_j}$ in terms of the maximum degree of the cumulative graph obtained by combining the first j coflows. In particular, the following lemma follows from the constraints of $\mathbf{LP_1}$.

Lemma 7 *For each coflow $j = 1, 2, \ldots, n$, the following inequality holds.*

$$\overline{C_j} \geq \frac{1}{2} \max_i \left\{ \sum_{k=1}^{j} L_{i,k} \right\} = \frac{1}{2} \Delta(\bigcup_{k \leq j} G_k)$$

Proof Let $S = \{1, 2, \ldots, j\}$. The LP constraint (3) implies that

$$\max_i \left\{ \sum_{k=1}^{j} L_{i,k} \cdot \overline{C_k} \right\} \geq \max_i f_i(S) \geq \max_i \left\{ \frac{(\sum_{k=1}^{j} L_{i,k})^2}{2} \right\}$$

Since $\overline{C_k} \leq \overline{C_j}$, for each $k = 1, 2, \ldots j$ we have

$$\overline{C_j} \cdot \max_i \{ \sum_{k=1}^{j} L_{i,k} \} = \max_i \left\{ \sum_{k=1}^{j} L_{i,k} \overline{C_j} \right\} \geq \max_i \left\{ \sum_{k=1}^{j} L_{i,k} \overline{C_k} \right\} \geq \max_i \left\{ \frac{(\sum_{k=1}^{j} L_{i,k})^2}{2} \right\}$$

which is equivalent to

$$\overline{C_j} \geq \frac{1}{2} \max_i \left\{ \sum_{k=1}^{j} L_{i,k} \right\} = \frac{1}{2} \Delta(\bigcup_{k \leq j} G_k)$$

6.1 Proof of the Main Theorems Using the LP Rounding Approach

Theorem 2 *There exists a deterministic, combinatorial, polynomial time 4-approximation algorithm for coflow scheduling without release times.*

Proof Consider any coflow j and let $C_j(alg)$ denote the completion time of coflow j when scheduled as per Algorithm 3. Since all coflows have zero release times, at time $t_1 = 0$ all the coflows are arrived. Let $G'_1, \ldots, G'_n$ denote the coflows after all the edges have been moved backward. According to Lemma

1 each coflow G'_k could be finished at time $\Delta(G'_k)$, thus when the coflows are scheduled sequentially, we get the following.

$$C_j(alg) = \sum_{k \leq j} \Delta(G'_k)$$

Applying Lemma 2 and Lemma 7:

$$C_j(alg) = \sum_{k \leq j} \Delta(G'_k) \leq 2\Delta(\bigcup_{k \leq j} G_k) \leq 4\overline{C_j}$$

Hence, the total weighted completion time of our schedule can be bounded by the objective of the optimal LP solution.

$$\sum_{j=1}^{n} w_j C_j(alg) \leq 4 \sum_{j=1}^{n} w_j \overline{C_j} \leq 4OPT$$

Theorem 1 *There exists a deterministic, combinatorial, polynomial time 5-approximation algorithm for coflow scheduling with release times.*

Proof

$$C_j(alg) \leq \max_{k \leq j} r_k + 2\Delta(\bigcup_{k \leq j} G_k) = \max_{k \leq j} r_k + 4\overline{C_j} \leq 5\overline{C_j}$$

The first inequality follows from Lemma 4 and the second equality follows from Lemma 7. The last inequality holds since $\overline{C_j} \geq \overline{C_k}$ for all $1 \leq k \leq j$ and $\overline{C_k} \geq r_k$.

The cost of obtained coflow schedule is

$$\sum_{j=1}^{n} w_j C_j(alg) \leq 5 \sum_{j=1}^{n} w_j \overline{C_j} \leq 5OPT.$$

References

1. `https://hadoop.apache.org.`
2. Bansal, N., Khot, S.: Inapproximability of hypergraph vertex cover and applications to scheduling problems. In: ICALP, pp. 250–261. Springer (2010)
3. Chen, Z.L., Hall, N.G.: Supply chain scheduling: Conflict and cooperation in assembly systems. Operations Research **55**(6), 1072–1089 (2007)
4. Chowdhury, M., Stoica, I.: Coflow: A networking abstraction for cluster applications. In: ACM Workshop on Hot Topics in Networks, pp. 31–36. ACM (2012)
5. Chowdhury, M., Stoica, I.: Efficient coflow scheduling without prior knowledge. In: SIGCOMM, pp. 393–406. ACM (2015)
6. Chowdhury, M., Zhong, Y., Stoica, I.: Efficient coflow scheduling with varys. In: SIGCOMM, SIGCOMM '14, pp. 443–454. ACM, New York, NY, USA (2014). DOI 10.1145/2619239.2626315. URL `http://doi.acm.org/10.1145/2619239.2626315`
7. Davis, J.M., Gandhi, R., Kothari, V.H.: Combinatorial algorithms for minimizing the weighted sum of completion times on a single machine. Operations Research Letters **41**(2), 121–125 (2013)
8. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Communications of the ACM **51**(1), 107–113 (2008)

9. Garg, N., Kumar, A., Pandit, V.: Order scheduling models: Hardness and algorithms. In: FSTTCS, pp. 96–107. Springer (2007)
10. Khuller, S., Li, J., Sturmfels, P., Sun, K., Venkat, P.: Select and permute: An improved online framework for scheduling to minimize weighted completion time. CoRR **abs/1704.06677** (2017). URL `http://arxiv.org/abs/1704.06677`
11. Khuller, S., Purohit, M.: Brief announcement: Improved approximation algorithms for scheduling co-flows. In: Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '16, pp. 239–240. ACM, New York, NY, USA (2016). DOI 10.1145/2935764.2935809. URL `http://doi.acm.org/10.1145/2935764.2935809`
12. Leung, J.Y.T., Li, H., Pinedo, M.: Scheduling orders for multiple product types to minimize total weighted completion time. Discrete Applied Mathematics **155**(8), 945–970 (2007)
13. Luo, S., Yu, H., Zhao, Y., Wang, S., Yu, S., Li, L.: Towards practical and near-optimal coflow scheduling for data center networks. IEEE Transactions on Parallel and Distributed Systems **27**(11), 3366–3380 (2016)
14. Mastrolilli, M., Queyranne, M., Schulz, A.S., Svensson, O., Uhan, N.A.: Minimizing the sum of weighted completion times in a concurrent open shop. Operations Research Letters **38**(5), 390–395 (2010)
15. Qiu, Z., Stein, C., Zhong, Y.: Minimizing the total weighted completion time of coflows in datacenter networks. In: SPAA, SPAA '15, pp. 294–303. ACM, New York, NY, USA (2015). DOI 10.1145/2755573.2755592. URL `http://doi.acm.org/10.1145/2755573.2755592`
16. Queyranne, M.: Structure of a simple scheduling polyhedron. Mathematical Programming **58**(1-3), 263–285 (1993)
17. Sachdeva, S., Saket, R.: Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. In: IEEE Conference on Computational Complexity, pp. 219–229. IEEE (2013)
18. Shafiee, M., Ghaderi, J.: An improved bound for minimizing the total weighted completion time of coflows in datacenters. CoRR **abs/1704.08357** (2017). URL `http://arxiv.org/abs/1704.08357`
19. Wang, G., Cheng, T.E.: Customer order scheduling to minimize total weighted completion time. Omega **35**(5), 623–626 (2007)
20. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. HotCloud **10**, 10–10 (2010)
21. Zhao, Y., Chen, K., Bai, W., Yu, M., Tian, C., Geng, Y., Zhang, Y., Li, D., Wang, S.: Rapier: Integrating routing and scheduling for coflow-aware data center networks. In: INFOCOM, pp. 424–432. IEEE (2015)

A A Combinatorial 3-approximation Algorithm For Concurrent Open Shop with Release Times

Theorem 1 *Algorithm 1 gives a 3-approximation for concurrent open shop scheduling with release times.*

Proof We use algorithm 1 to get a permutation $\{1, 2, \cdots, n\}$ for a set of jobs J. If we schedule the jobs according to this permutation sequentially, we'll get:

$$C_j \leq \max_{i' \leq j} r_{i'} + \sum_{k \leq j} L_{\mu(j),k}$$

Lemma 5 with $a = 1$ and $b = 1$, imply that:

$$\sum_j w_j C_j(alg) \leq (1 + \frac{1}{\kappa}) \sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j + 2(\kappa + 1) \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S)$$

To minimize the approximation ratio, we substitute $\kappa = \frac{1}{2}$ and obtain

$$\sum_j w_j C_j(alg) \leq 3 \left(\sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j + \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S) \right) \leq 3 \cdot OPT$$

where the last inequality follows from weak duality as α and β constitute a feasible dual solution.

B Correction of Algorithm by Qiu et al. [15]

We now give a brief overview of the approximation algorithm given by Qiu, Stein, and Zhong [15].

Interval-Indexed LP Formulation

In the first step we write an interval-indexed linear programming relaxation for the coflow scheduling problem similar to that for the concurrent open shop problem by Wang and Cheng [19].

Let $\bar{C_j}$ denote the approximated completion time of coflow j obtained by an optimal feasible solution to this LP relaxation. We first order the coflows in non-decreasing order of these approximated completion times, i.e. we have the following.

$$\bar{C_1} \leq \bar{C_2} \ldots \leq \bar{C_n} \tag{19}$$

Let V_j denote the maximum load on any port by the *first* j coflows taken together in the above ordering, i.e.

$$V_j = \max \left[\max_i \left\{ \sum_{k=1}^{j} \sum_o d_{io}^k \right\}, \max_o \left\{ \sum_{k=1}^{j} \sum_i d_{io}^k \right\} \right].$$

Qiu et al. [15] prove that these V_j values provide a good approximation for the optimal completion times of the coflows. In particular, they show the following where C_j^* denotes the completion time of coflow j in an optimal schedule.

$$\sum_j w_j V_j \leq \frac{16}{3} \sum_j w_j C_j^* \tag{20}$$

Grouping Coflows

Divide time into geometrically increasing intervals as follows - $[1], [2], [3, 4], [5, 8], [9, 16], \ldots$. Let $I_l = (2^{l-2}, 2^{l-1}]$ denote the l^{th} interval.

Now group the coflows based on the interval where their V values lie and let S_l denote the set of coflows assigned to interval I_l. So for all coflows $j \in S_l$, we have $2^{l-2} < V_j \leq 2^{l-1}$.

Algorithm 1

- For $l = 1, 2, \ldots$
  - Wait until the last coflow in S_l is released.
  - Group all coflows in S_l and schedule as per Algorithm 1 in [15]. This would take time at most $V_k \leq 2^{l-1}$ where k is the last job in the group.

Analysis

Qiu et al. claim the following (Proposition 1 in [15]).

Proposition 1 *For any coflow j, let $C_j(alg)$ denote the completion time of coflow j as per Algorithm 1. Then we have*

$$C_j(alg) \leq \max_{1 \leq g \leq j} \{r_g\} + 4V_j.$$

Since $C_j^* \geq \max_{1 \leq g \leq j} \{r_g\}$, Proposition 1 and Equation (20) together imply the following theorem (Theorem 1 in [15]).

Theorem 2 *There exists a deterministic polynomial time 67/3 approximation algorithm for coflow scheduling, i.e.*

$$\sum_j w_j C_j(alg) \leq \frac{67}{3} \sum_j w_j C_j^*.$$

Error

We now show that the Proposition 1 stated above is incorrect. Consequently, Theorem 1 no longer holds. Recall that Algorithm 1 groups jobs based on their V values alone and does not consider their release times.

Consider a simple case where $m = 1$ and we have just one input port and one output port. Say we have two jobs j_1 and j_2 such that j_1 needs to send 3 units of data and j_2 needs to send 1 unit of data. Also say $r_{j_1} = 0$ and $r_{j_2} = 100$. By definition, we have $V_{j_1} = 3$ and $V_{j_2} = 4$; note that both the jobs belong to the same interval $I_3 = (2, 4]$. Now since both jobs belong to the same interval, Algorithm 1 waits for both the jobs to be released and then schedules them together (after time 100). In this case, the claim in Proposition 1 clearly does not hold for job j_1.

Proposition 2 in [15] makes a similar claim for a grouping algorithm using randomized intervals. Again, the above instance serves as a counterexample to the claim. Consequently, Theorem 2 in [15] does not hold.

In the following section, we show that the deterministic grouping algorithm can be modified to yield a $\frac{76}{3}$-approximation algorithm. Note that this is worse than the $\frac{67}{3}$ factor claimed earlier. It is not immediately clear whether the randomized algorithm from [15] can be corrected via a similar modification.

B.1 Corrected Grouping Algorithm

We first solve the interval-indexed LP formulation to obtain approximated completion times $\bar{C}_j$. Without loss of generality, we assume that the coflows are ordered as per Equation (19).

As shown by Leung, Li, and Pinedo (Theorem 13 in [12]), the analysis of Wang and Cheng [19] can be extended to the case of general release times to obtain the following.

$$\sum_j w_j \bar{C}_j \leq \frac{19}{3} \sum_j w_j C_j^* \tag{21}$$

This is analogous to Lemma 3 in [15] that shows that $\sum_j w_j V_j \leq \frac{16}{3} \sum_j w_j C_j^*$ where V_j is the maximum load on any port by the *first* j coflows taken together (as per the ordering).

Since $\bar{C}_j$ denotes the approximation completion time of coflow j as computed by the valid LP relaxation, we also have the following where r_j denotes the release time of coflow j.

$$\bar{C}_j \geq r_j \tag{22}$$

$$\bar{C}_j \geq V_j \tag{23}$$

B.1.1 Algorithm

Divide time into geometrically increasing intervals as follows - $[1], [2], [3, 4], [5, 8], [9, 16], \ldots$. Let $I_l = (2^{l-2}, 2^{l-1}]$ denote the l^{th} interval.

Now group the coflows based on the interval where their $\bar{C}$ values lie and let S_l denote the set of coflows assigned to interval I_l. So for all coflows $j \in S_l$, we have $2^{l-2} < \bar{C}_j \leq 2^{l-1}$.

Algorithm

- For $l = 1, 2, \ldots$
  - Wait until the last coflow in S_l is released AND all coflows in S_{l-1} have finished. (whichever is later).
  - Group all coflows in S_l and schedule as per Algorithm 1 in [15]. This would take time at most $V_k \leq 2^{l-1}$ where k is the last job in the group.

Analysis

Let $\tilde{C}_l$ denote the time by which all coflows in S_l have been scheduled by the above algorithm.

Claim $\tilde{C}_l \leq 2 \times 2^{l-1} = 2^l$ for every group S_l.

Proof We prove by induction. For group S_1, we start executing the schedule at $\max_{j \in S_1} r_j \leq \max_{j \in S_1} \bar{C}_j \leq 2^{1-1} = 1$ and the schedule takes time at most $V_k \leq 2^{1-1} = 1$ where k is the last coflow in the group. So the base case is true.

Now assume that the claim is true for some group S_l. As per the algorithm, the coflows in group S_{l+1} start executing at $\tilde{C}_l$ or $\max_{j \in S_{l+1}} r_j$ whichever is later. By induction, we are guaranteed that $\tilde{C}_l \leq 2^l$. Also $\max_{j \in S_{l+1}} r_j \leq \max_{j \in S_{l+1}} \bar{C}_j \leq 2^l$. Thus the coflows in group S_{l+1} start executing latest at time 2^l. We know that all these coflows require at most $V_k \leq \bar{C}_k \leq 2^l$ time units to complete. As a result, all the coflows in this group are scheduled by time $2^l + 2^l = 2^{l+1}$.

And thus the claim follows by induction.

Claim For any coflow j, let $C_j(alg)$ denote the completion time of coflow j as per the algorithm. Then $C_j(alg) < 4\bar{C}_j$.

Proof Consider any coflow j, and let l be such that $j \in S_l$. Hence we have $\bar{C}_j > 2^{l-2}$. By the previous claim, we have

$$C_j(alg) \leq \tilde{C}_l \leq 2^l = 4 \times 2^{l-2} < 4\bar{C}_j$$

Corollary 2 *There is a deterministic $\frac{76}{3}$-approximation for coflow scheduling with arbitrary release times.*

Proof Claim B.1.1 and Equation (21) together imply a $\frac{76}{3}$-approximation algorithm for coflow scheduling with release times.

C Counterexample to Claim by Luo et al. [13]

Luo et al. [13] claim a 2-approximation algorithm for the coflow scheduling problem by proving that it is equivalent to concurrent open shop scheduling. One of the key ingredients of their proof is the following claim that is implicit in Lemma 3 in [13].

Claim (Restated from [13]) Given two coflows G_k and G_l, we can find a feasible schedule for both the coflows such that $C_k + C_l = \min\{\Delta(G_k) + \Delta(G_k \bigcup G_l), \Delta(G_l) + \Delta(G_k \bigcup G_l)\}$.

Counterexample

We show that Claim C is erroneous via a simple counterexample. Consider two coflows on a 3×3 datacenter as shown in Figure 5. Note that while coflows G_1 and G_2 have $\Delta(G_1) = 1$ and $\Delta(G_2) = 2$, the combined coflow $G_1 \bigcup G_2$ also has $\Delta(G_1 \bigcup G_2) = 2$. Consequently, the RHS in Claim C equals $\Delta(G_1) + \Delta(G_1 \bigcup G_2) = 3$.

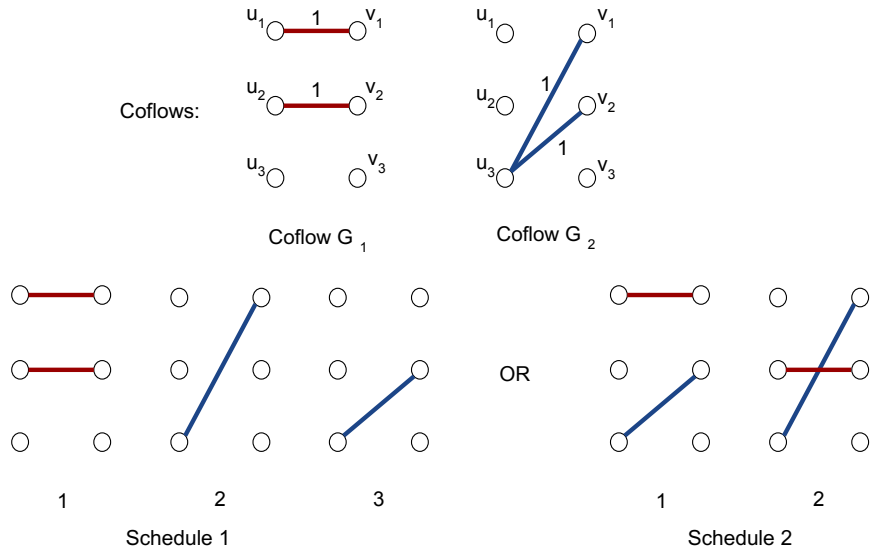

Fig. 5 Simple counterexample to Claim C

On the other hand, as seen in Figure 5, if coflow G_1 is scheduled so that $C_1 = \Delta(G_1) = 1$, then the matching constraints force coflow G_2 to have completion time $C_2 = 3$. On the other hand, delaying one edge of coflow G_1, leads to a schedule with $C_1 = C_2 = 2$. In both cases, we have $C_1 + C_2 = 4$ (instead of 3) leading to a contradiction to the claim.