On Scheduling Co-flows*

Saba Ahmadi[1], Samir Khuller[1], Manish Purohit[2], and Sheng Yang[1]

[1] University of Maryland, College Park
{saba,samir,styang}@cs.umd.edu

[2] Google, Mountain View
mpurohit@google.com

Abstract. Applications designed for data-parallel computation frameworks such as MapReduce usually alternate between computation and communication stages. Co-flow 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 co-flow j is a collection of flow demands $\{d^j_{io}\}_{i\in m, o\in m}$ that is said to be complete once *all* of its requisite flows have been scheduled.
We consider the offline co-flow scheduling problem with and without release times to minimize the total weighted completion time. Co-flow scheduling generalizes the well studied concurrent open shop scheduling problem and is thus NP-hard. Recently, Qiu, Stein and Zhong [13] obtained the first constant approximation algorithms for this problem based on LP relaxation with a deterministic $\frac{67}{3}$-approximation and a randomized $(9 + \frac{16\sqrt{2}}{3}) \approx 16.54$-approximation. In this paper, we give a combinatorial algorithm based on LP relaxation which improves significantly upon theirs to yield a deterministic 5-approximation algorithm with release times. For the case without release time, we obtain a deterministic 4-approximation.

Keywords: Co-flow scheduling, Concurrent Open Shop

* This work is supported by NSF grant CCF 1217890.

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 [17] 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" phase 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 co-flows as an effective networking abstraction to represent the collective communication requirements of a job. In this paper, we consider the problem of scheduling co-flows 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 co-flow 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 co-flow 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 co-flow j. Figure 3 shows a single co-flow over a 2×2 switch. For instance, the co-flow 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 co-flow j also has a weight w_j that indicates its relative importance and a release time r_j.

A co-flow 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 co-flow j is defined as the earliest time such that for every input i and output o, d^j_{io} 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 scheduling that minimizes the total, weighted completion time of the co-flows, i.e. minimize $\sum_j w_j C_j$.

1.1 Related Work

Chowdhury and Stoica [4] introduced the co-flow abstraction to describe the prevalent communication patterns in data centers. Since then co-flow scheduling has been a topic of active research [6,5,13,18] in both the systems and theory communities. Although co-flow 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.

For the special case when all co-flows have zero release time, Qiu, Stein and Zhong [13] obtain 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 co-flow scheduling with arbitrary release times, Qiu et al. [13] claim a deterministic $\frac{67}{3}$ approximation and a randomized $(9+\frac{16\sqrt{2}}{3})$ approximation algorithm. However in Appendix E, we demonstrate a subtle error in their proof that deals with non-zero release times. We further show that their techniques in fact only yield a deterministic $\frac{76}{3}$-approximation algorithm for co-flow scheduling with release times.

By exploiting a connection with the well-studied concurrent open shop scheduling problem, Luo et al. [11] claim a 2-approximation algorithm for co-flow scheduling when all the release times are zero. Unfortunately, as we show in Appendix F their proof too is flawed and the result does not hold.

1.2 Our Contributions

The main algorithmic contribution of this paper is a combinatorial algorithm for the offline co-flow scheduling problem with improved approximation guarantees.

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

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

Our results significantly improve upon the approximation algorithms developed by Qiu et al. [13] whose techniques yield an approximation factors of $\frac{76}{3} = 25.33$ (See Appendix E) and $(8+\frac{16\sqrt{2}}{3}) \approx 15.54$ respectively for the two cases. In addition, our algorithm is completely combinatorial and does not require solving a linear program.

We also extend the primal dual algorithm by Mastrolilli et al. [12] to give a 3-approximation algorithm for the concurrent open shop problem when the jobs have arbitrary release times.

1.3 Connection to Concurrent Open Shop

The co-flow scheduling problem as described above generalizes the well-studied concurrent open shop problem [12,3,9,10,16]. 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. Any machine can perform at most one unit of processing at a time. The objective is to find a feasible schedule that minimizes the total weighted completion time of jobs. An LP-relaxation using completion time variables yields a 2-approximation algorithm for concurrent open shop scheduling when all release times are zero [3,9,10] and a 3-approximation algorithm for arbitrary release times [9,10]. Mastrolilli et al. [12] show that a simple greedy algorithm also yields a 2-approximation for concurrent open shop without release times using primal-dual techniques. We prove our combinatorial algorithm yields a 3-approximation for concurrent open shop with release times.

The concurrent open shop problem can be viewed as a special case of co-flow scheduling when the demand matrices D^j for all co-flows j are diagonal [6,13]. At first glance, it appears that co-flow scheduling is much harder than concurrent open shop. For instance, while concurrent open shop always admits an optimal permutation schedule, such a property may not be true for co-flows [6]. In fact, even without release times, the best known approximation algorithm for scheduling co-flows has an approximation factor of ≈ 15.54 [13], in contrast to the many 2-approximations known for the concurrent open shop problem. 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 co-flows such that sequentially scheduling the co-flows after some post-processing in this permutation leads to provably good co-flow schedules.

2 Preliminaries

We first introduce some notation to facilitate the following discussion. For every co-flow 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 co-flow j needs to transmit through port i. Similarly, we define $L_{o,j} = \sum_{i=1}^{m} d_{io}^j$ for every co-flow j and output port o. Equivalently, a co-flow 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 co-flow 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 co-flow j by the corresponding bipartite graph G_j when there is no confusion.

Representing a co-flow as a bipartite graph simplifies some of the notation that we have seen previously. For instance, for any co-flow 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.

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

For any graph G_j, let $\Delta(G_j) = \max_{s \in I \cup O} deg_{G_j}(s) = \max\{\max_i L_i^j, \max_o L_o^j\}$ denote the maximum degree of any node in the graph, i.e., the load on the most heavily loaded port of co-flow j.

In our algorithm we consider co-flows obtained as the union of two or more co-flows. 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 the obvious manner.

2.1 Scheduling a Single Co-flow

Before we present our algorithm for the general co-flow scheduling problem, it is instructive to consider the problem of feasibly scheduling a *single co-flow* subject to the matching constraints. Given a co-flow G_j, the maximum degree of any vertex in the graph $\Delta(G_j) = \max_v deg_G(v)$ is an obvious lower bound on the amount of time required to feasibly schedule co-flow G_j. In fact, the following lemma by Qiu et al. [13] shows that this bound is always achievable for any co-flow. The proof follows by repeated applications of Hall's theorem on the existence of perfect matchings in bipartite graphs.

Lemma 1. *[[13]] There exists a polynomial time algorithm that schedules a single co-flow 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 co-flow schedule for the given time interval by partially scheduling a co-flow if necessary.

Corollary 1. *Given a sequence of co-flows $G_1, G_2, \ldots, G_n$, 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 co-flow schedule for the time interval $(t_s, t_e]$ such that -*

- *co-flows $G_1, G_2, \ldots, G_{j-1}$ are completely scheduled.*
- *co-flow 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 co-flow j that has not yet been scheduled.*
- *co-flows $G_{j+1}, \ldots, G_n$ are not scheduled.*

Proof. We defer the proof to the Appendix C.1.

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 [9,10] to formulate the following linear program as a relaxation of the co-flow scheduling problem. We introduce a variable C_j for every co-flow G_j to denote its completion time. Let $J = \{1, 2, \ldots, n\}$ denote the set of all co-flows and $M = I \cup O$ denote the set of all the ports. For any subset $S \subseteq J$ and each port i, let $f_i(S)$ be as defined below.

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

Figure 1 shows our LP relaxation. The first set of constraints ensure that the completion time of any job j is at least its release time r_j plus the load of co-flow j on any port i. The second set of constraints are standard in parallel scheduling literature (see for example [14]) and are used to effectively lower bound completion time variables.

$$\mathbf{LP_1}: \qquad \min \sum_{j \in J} w_j C_j \tag{3}$$

$$\text{subject to,} \quad \forall j \in J, \text{ and } \forall i \in M, \qquad C_j \geq r_j + L_{i,j} \tag{4}$$

$$\forall i \in M, \text{ and } \forall S \subseteq J, \qquad \sum_{j \in S} L_{i,j} C_j \geq f_i(S) \tag{5}$$

Fig. 1. LP Relaxation for Co-Flow Scheduling

3 High Level Ideas

We use $\mathbf{LP_1}$ and its dual to develop a combinatorial algorithm (Algorithm 1) in Section 4.1 to obtain a good permutation of the co-flows. This primal dual algorithm is inspired by Davis et al. [7] and Mastrolilli et al. [12]. As we show in Lemma 5, once the co-flows are permuted as per this algorithm, we can bound the completion time of a co-flow 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 co-flows in the permutation.

A naïve approach now would be to schedule each co-flow independently and sequentially using Lemma 1 in this permutation. Since all co-flows $k \leq j$ would need to be scheduled before starting to schedule j, the completion time of co-flow j under such a scheme would be $\sum_{k \leq j} \Delta(G_k)$. Unfortunately, for arbitrary co-flows we can have $\sum_{k \leq j} \Delta(G_k) >> \Delta(\bigcup_{k \leq j} G_k)$. For instance, Figure 4 shows three co-flows such that $\Delta(G_1)+\Delta(G_2)+\Delta(G_3) = 300 > \Delta(G_1 \cup G_2 \cup G_3) = 101$.

One key insight is that sequentially scheduling co-flows one after another may waste resources. Since the amount of time required to completely schedule a co-flow 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, then the augmented co-flow can still be scheduled in the same time interval. This observation leads to the natural idea of "shifting" edges from a co-flow j later in the permutation to a co-flow k ($k < j$) as such a shift does not delay co-flow k further but may significantly reduce the requirements of co-flow j. For instance in Figure 4, shifting the edge (c, d) from graph G_2 to G_1 and the edge (e, f) from G_3 to G_1 leaves $\Delta(G_1)$ unchanged but drastically reduces $\Delta(G_2)$ and $\Delta(G_3)$. In Algorithm 3, we formalize this notion of shifting edges and prove that after all such edges have been shifted, sequentially scheduling the augmented co-flows leads to provably good co-flow schedules.

4 Approximation Algorithm for Co-flow Scheduling with Release Times

In this section we present a simple combinatorial 5-approximation algorithm for minimizing the weighted sum of completion times of a set of co-flows 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 co-flows. In the second stage, we show that scheduling the co-flows sequentially in this ordering after some preprocessing steps yields a provably good co-flow schedule.

4.1 Finding a permutation of co-flows

Although our algorithm does not require solving a linear program, we use the linear program in Figure 1 and its dual (Figure 2) in the design and analysis of the algorithm.

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

Fig. 2. Dual of LP_1

Our algorithm works as follows. We build up a permutation of the co-flows in the reverse order iteratively. Let κ be a constant that we optimize later. In any iteration, let j be the unscheduled job with the latest release time, let μ be the machine with the highest load and let L be the load on machine μ. Now if $r_j > \kappa L$, we raise the dual variable $\alpha_{\mu,j}$ until the corresponding dual constraint is tight and place co-flow j to be last in the permutation. But if $r_j \leq \kappa L$, then we raise the dual variable $\beta_{\mu,J}$ until the dual constraint for some job j' becomes tight and place co-flow j' to be last in the permutation. Algorithm 1 gives the formal description of the complete algorithm.

Algorithm 1: Permuting Co-Flows

```
1  $J$ is the set of unscheduled jobs and initially $J = \{1, 2, \cdots, n\}$
2  Initialize $\alpha_{i,j} = 0$ for all $i \in M, j \in J$ and $\beta_{i,S} = 0$ for all $i \in M, S \subseteq J$
3  $L_i = \sum_{j \in J} L_{ij} \; \forall i \in M$                    // load of machine $i$
4  for $k = n, n-1, \cdots, 1$ do
5      $\mu(k) = argmax_{i \in M} L_i$          // determine the machine with highest load
6      $j = argmax_{j \in J} r_j$                      // determine job that released last
7      if $r_j > \kappa \cdot L_{\mu(k)}$ then
8          $\alpha_{\mu(k),j} = (w_j - \sum_{i \in M} \sum_{S \ni j} L_{i,j} \beta_{i,S})$
9          $\sigma(k) \leftarrow j$
10     end
11     else if $r_{\sigma(k)} \leq \kappa \cdot L_{\mu(k)}$ then
12         $j' = argmin_{j \in J} \left( \frac{w_j - \sum_{i \in M} \sum_{S \ni j} L_{i,j} \beta_{i,S}}{L_{\mu(k),j}} \right)$
13         $\beta_{\mu(k),J} = \left( \frac{w_{j'} - \sum_{i \in M} \sum_{S \ni j'} L_{i,j'} \beta_{i,S}}{L_{\mu(k),j'}} \right)$
14         $\sigma(k) \leftarrow j'$
15     end
16     $J \leftarrow J \setminus \sigma(k)$
17     $L_i \leftarrow L_i - L_{i,\sigma(k)}, \forall i \in M$
18 end
19 Output permutation $\sigma(1), \sigma(2), \cdots, \sigma(n)$
```

4.2 Algorithm For Scheduling a Permutation of Co-flows

Algorithm 1 gives a permutation of co-flows. We assume without loss of generality that the co-flows are ordered based on this permutation, i.e. $\sigma(j) = j$.

As we discussed in Section 3, naively scheduling the co-flows sequentially in this order can lead to very bad solutions. On the other hand, by appropriately moving edges from a co-flow j to an earlier co-flow k ($k < j$), we can get a provably good co-flow schedule. The crux of our algorithm lies in the subroutine *MoveEdgesBack* defined in Algorithm 2.

Algorithm 2: The *MoveEdgesBack* subroutine.

```
1  Function MoveEdgesBack($G_k, G_j$)
2      for $e = (u, v) \in G_j$ do
3          $\delta = min(\Delta(G_k) - deg_{G_k}(u), \Delta(G_k) - deg_{G_k}(v), w_{G_j}(e))$;
4          $w_{G_j}(e) = w_{G_j}(e) - \delta$;
5          $w_{G_k}(e) = w_{G_k}(e) + \delta$;
6      end
7      return $G_k, G_j$;
```

Given two bipartite graphs G_k and G_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 the co-flow 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 co-flows have zero release times, then we can safely move edges of a co-flow 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 co-flow G_j are only moved to co-flows G_k such that these edges are only scheduled after time r_j, i.e. after they are released. Algorithm 3 describes the pseudo-code for co-flow scheduling with arbitrary release times. Here q denote the number of distinct values taken by the release times of the n co-flows. 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 co-flow j that has not been scheduled yet. We consider every ordered pair of co-flows $k < j$ such that both the co-flows have been released and *MoveEdgesBack* from graph G'_j to graph G'_k. Finally, we begin to schedule the co-flows sequentially in order using Corollary 1 until all co-flows are scheduled completely or we reach time t_{i+1} when a new co-flow gets released and the process repeats.

Algorithm 3: Co-Flow Scheduling

```
1  for $i = 1, 2, \ldots, q$ do
2      // Find a schedule for time interval $(t_i, t_{i+1}]$
3      for $j = 1, 2, \ldots, n$ do
4          $G'_j \leftarrow Unscheduled(G_j)$;
5      end
6      for $k = 1, 2, \ldots, n-1$ do
7          if $r_k < t_i$ then
8              for $j = k+1, \ldots, n$ do
9                  if $r_j < t_i$ then
10                     $G'_k, G'_j \leftarrow MoveEdgesBack(G'_k, G'_j)$;
11                 end
12             end
13         end
14     end
15     $Schedule(G'_1, G'_2, \ldots, G'_n, t_i, t_{i+1})$ using Corollary 1;
16 end
```

4.3 Analysis

We first analyze Algorithm 3 and upper bound the completion time of a co-flow j in terms of the maximum degree of the cumulative graph obtained by combining the first j co-flows in the given permutation.

Analyzing Co-Flows with Zero Release Times For ease of presentation we first analyze the special case when all co-flows are released at time zero. In this case, we have $q = 1$ in Algorithm 3 and thus the *for* loop in lines 1-16 is only executed once. The following lemma shows that after the *MoveEdgesBack* subroutine has been executed on every ordered pair of co-flows, for any co-flow j, the sum of maximum degrees of graphs G'_k $(k < j)$ is at most twice the maximum degree of the cumulative graph obtained by combining the first j co-flows.

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

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.

$$\Delta(G'_k) = \Delta(G_{k|j}) \text{ for all } k \le j. \tag{6}$$

For any $j \in \{1, 2, \dots, n\}$, consider the set S of graphs $G_{1|j}, G_{2|j}, \dots G_{j|j}$. 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 $G_1, \dots, G_j$ and no edges from graphs G_l for $l > j$, $\hat{G}_j$ is identical to the cumulative graph of the first j co-flows. In particular, we have the following.

$$\Delta(\hat{G}_j) = \Delta(\bigcup_{k \le j} G_k) \tag{7}$$

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

$$\begin{aligned}
\Delta(\hat{G}_j) &\ge \max\left\{deg_{\hat{G}_j}(u), deg_{\hat{G}_j}(v)\right\} &(8)\\
&\ge \max\left\{\sum_{G \in S_u} deg_G(u), \sum_{G \in S_v} deg_G(v)\right\} &(9)\\
&= \max\left\{\sum_{G \in S_u} \Delta(G), \sum_{G \in S_v} \Delta(G)\right\} &(10)
\end{aligned}$$

From Equation (6), we have the following.

$$\sum_{k \le j} \Delta(G'_k) = \sum_{k \le j} \Delta(G_{k|j}) = \sum_{G \in S} \Delta(G) \tag{11}$$

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.

$$\sum_{k\leq j} \Delta(G'_k) \leq 2\max\left\{\sum_{G\in S_u} \Delta(G), \sum_{G\in S_v} \Delta(G)\right\} \tag{12}$$

$$\leq 2\Delta(\hat{G}_j) = 2\Delta(\bigcup_{k\leq j} G_k) \tag{13}$$

where the last equality follows from Equation (7).

Lemma 3. *Consider any co-flow j and let $C_j(alg)$ denote the completion time of co-flow 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 co-flows after all the edges have been moved backward. According to Lemma 1 each co-flow G'_k could be finished at time $\Delta(G'_k)$, thus when the co-flows 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.

Analyzing Co-Flows with Arbitrary Release Times When the co-flows have arbitrary release times, we can bound the completion time of each co-flow j in terms of the maximum degree of the cumulative graph obtained by combining the first j co-flows and the largest release time of all the jobs before j in the permutation.

Lemma 4. *For any co-flow j, let $C_j(alg)$ denote the completion time of co-flow 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. We defer the proof to Appendix C.4.

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., $\sigma(k) = k, \forall k \in [n]$.

Lemma 5. *If there is an algorithm that generates a feasible co-flow schedule such that for any co-flow 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.*

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

Proof Sketch. Algorithm 1 judiciously sets the dual variables such that the dual constraint for an co-flow j is tight. Analyzing the cost of schedule obtained in terms of the dual variables yields the lemma. The formal proof is deferred to Appendix B.

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

Proof Sketch. The theorem follows from lemmas 4 and 5 along with an appropriate choice of the constant κ. The formal proof is deferred to Appendix C.2.

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

Proof Sketch. The theorem follows from lemmas 3 and 5 along with an appropriate choice of the constant κ.

References

1. `https://hadoop.apache.org.`
2. `https://cloud.google.com/dataflow/`.
3. Z.-L. Chen and N. G. Hall. Supply chain scheduling: Conflict and cooperation in assembly systems. *Operations Research*, 55(6):1072–1089, 2007.
4. M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.
5. M. Chowdhury and I. Stoica. Efficient coflow scheduling without prior knowledge. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 393–406. ACM, 2015.
6. M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with varys. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 443–454, New York, NY, USA, 2014. ACM.
7. J. M. Davis, R. Gandhi, and V. H. Kothari. Combinatorial algorithms for minimizing the weighted sum of completion times on a single machine. *Operations Research Letters*, 41(2):121–125, 2013.
8. J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.
9. N. Garg, A. Kumar, and V. Pandit. Order scheduling models: Hardness and algorithms. In *FSTTCS 2007: Foundations of Software Technology and Theoretical Computer Science*, pages 96–107. Springer, 2007.
10. J. Y.-T. Leung, H. Li, and M. Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics*, 155(8):945–970, 2007.
11. S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li. Towards practical and near-optimal coflow scheduling for data center networks. *IEEE Transactions on Parallel and Distributed Systems*, PP(99):1–1, 2016.
12. M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.

13. Z. Qiu, C. Stein, and Y. Zhong. Minimizing the total weighted completion time of coflows in datacenter networks. In *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '15, pages 294–303, New York, NY, USA, 2015. ACM.
14. M. Queyranne. Structure of a simple scheduling polyhedron. *Mathematical Programming*, 58(1-3):263–285, 1993.
15. A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of lp-based heuristics and lower bounds. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 301–315. Springer, 1996.
16. G. Wang and T. E. Cheng. Customer order scheduling to minimize total weighted completion time. *Omega*, 35(5):623–626, 2007.
17. M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica. Spark: cluster computing with working sets. *HotCloud*, 10:10–10, 2010.
18. Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang. Rapier: Integrating routing and scheduling for coflow-aware data center networks. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 424–432. IEEE, 2015.

A Figures

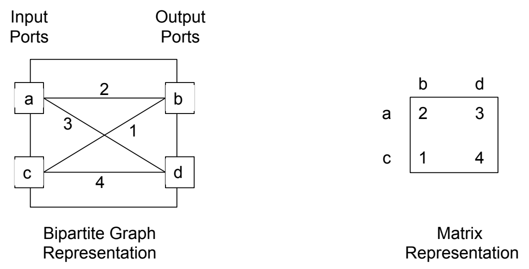

Fig. 3. An example co-flow over a 2 × 2 switch. The figure illustrates two equivalent representations of a co-flow - (i) as a weighted, bipartite graph over the set of ports, and (ii) as a $m \times m$ integer matrix.

B Analysis of Primal Dual Algorithm

We devote this section to prove Lemma 5.

Let S_j be the set of jobs $\{1, \cdots, j\}$. Let $\beta_{i,j} = \beta_{i,S_j}$. Also let $L_{\mu(j)}(S_j) = \sum_{k \le j} L_{\mu(j),k} = \Delta(\bigcup_{k \le j} G_k)$. We will first state a few lemmas for the correctness of primal-dual.

Lemma 6. *The following statements hold.*

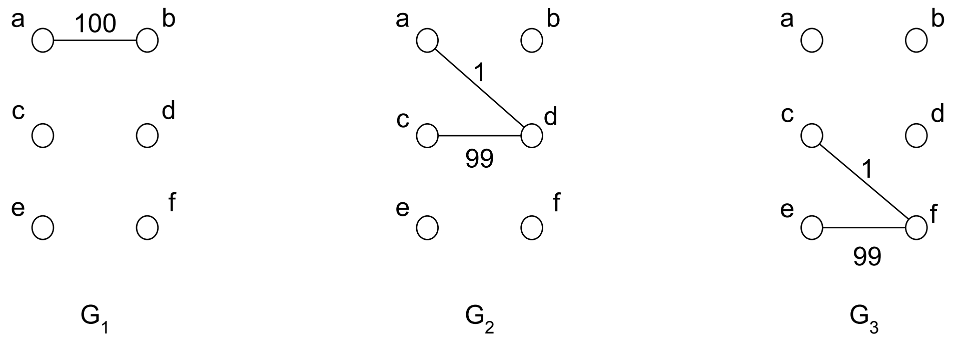

Fig. 4. Example that illustrates sequentially scheduling co-flows independently can lead to bad schedules.

(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 $i \leq j$ then $r_i \leq \kappa \cdot L_{\mu(j)}(S_j)$.
(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 $i \leq j$ then $r_i \leq r_j$.

The correctness of Lemma 6 can be directly obtained from Algorithm 1.

Lemma 7. *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.

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

Lemma 8. *(Proved in [15]). For any $i \in M$ and $S \subseteq J$, we have that $(\sum_{j\in S} L_{i,j})^2 \leq (2 - \frac{2}{n+1}) f_i(S)$.*

Lemma 8 is for the widely used parallel constraints. Now, with everything ready, we will restate and prove Lemma 5.

Lemma 5. *If there is an algorithm that generates a feasible co-flow schedule such that for any co-flow 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.*

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

Proof. By applying Lemma 7:

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

First let's bound $\sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} \cdot C_j$:

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

By applying Lemma 6 parts (c), (d):

$$\leq \sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} \left(a \cdot r_j + b \cdot \frac{r_j}{\kappa} \right)$$
$$\leq \left(a + \frac{b}{k} \right) \sum_{j=1}^{n} \sum_{i \in M} \alpha_{i,j} r_j$$

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

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

By applying Lemma 6 part (b):

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

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

By applying Lemma 8:

$$\leq (a\kappa + b) \sum_{i \in M} \sum_{k=1}^{n} \beta_{i,k} \left(2 - \frac{2}{n+1}\right) f_{\mu(k)}(S_k)$$
$$\leq 2(a\kappa + b) \sum_{i \in M} \sum_{k=1}^{n} \beta_{i,k} f_{\mu(k)}(S_k)$$

By applying Lemma 6 part (a):

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

Therefore,

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

C Missing Proofs

C.1 Proof of Corollary 1

Proof. By scheduling co-flows $G_1, G_2, \ldots, G_{j-1}$ sequentially using Lemma 1, we can completely schedule these co-flows 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 co-flow 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 co-flows have been completed. This partial scheduling of co-flow G_j ends at time t_e as desired. Let $\tilde{G_j} \subset G_j$ denote the partial co-flow that has not yet been scheduled. Inspecting schedule S, we observe that S schedules the partial co-flow $\tilde{G_j}$ from time steps $t_e - (t_s + \sum_{k=1}^{j-1} \Delta(G_k))$ to $\Delta(G_j)$. Hence, we must have $\Delta(\tilde{G_j}) \leq t_s + \sum_{k=1}^{j} \Delta(G_k) - t_e$.

C.2 Proof of Theorem 1

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

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

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

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.

C.3 Proof of Theorem 2

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

Proof. We use algorithm 1 to get a permutation $\{1, 2, \cdots, n\}$ for a set of co-flows J. If we schedule the co-flows using alorithm 3, according to lemma 3, for every co-flow j:

$$C_j \leq \Delta(\bigcup_{k \leq j} G_k)$$

Lemma 5 with $a = 0$ and $b = 2$, imply that:

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

Since all the release times are zero,

$$\sum_j w_j C_j(alg) \leq 4 \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S) \leq 4 \cdot OPT$$

C.4 Proof of Lemma 4

Proof. Consider any co-flow j. Let $t_i = \max_{l \leq j} r_l$ denote the earliest time when all co-flows 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 co-flow k in iteration i before edges have been moved back, i.e., $G_{k,i}$ denotes the state of co-flow k in iteration i after line 5. Since some edges from co-flow 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 co-flow k after the *MoveEdgesBack* subroutines have been executed, i.e. at line 14. We now claim that

$$C_j(alg) \leq t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \tag{14}$$

If $t_{i+1} \geq t_i + \sum_{k \leq j} \Delta(G'_{k,i})$, Corollary 1 guarantees that co-flows $1 \leq k \leq j$ will be completely scheduled sequentially in this iteration. Completion time of co-flow 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 co-flow 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 co-flows $k \leq p-1$ are completely scheduled and co-flow p is partially scheduled so that we have the following.

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

$$\Delta(G'_{k,i+1}) = \Delta(G_{k,i+1}) = 0, \forall k \leq p-1 \tag{16}$$

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

$$\Delta(G'_{k,i+1}) = \Delta(G'_{k,i}), \forall p < k \leq j \tag{17}$$

From equations (15) - (17), we get

$$t_{i+1} + \sum_{k \leq j} \Delta(G'_{k,i+1}) = t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \tag{18}$$

Proceeding this way inductively, we obtain

$$t_{i+x} + \sum_{k \leq j} \Delta(G'_{k,i+x}) = t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \tag{19}$$

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 $i+x$, co-flow 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.

$$C_j(alg) \leq t_i + \sum_{k \leq j} \Delta(G'_{k,i}) \leq t_i + 2\Delta\left(\bigcup_{k \leq j} G_{k,i}\right) \leq t_i + 2\Delta\left(\bigcup_{k \leq j} G_k\right) \tag{20}$$

where the second inequality follows from Lemma 2.

D 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.

E Correction of Algorithm by Qiu et al. [13]

We now give a brief overview of the approximation algorithm given by Qiu, Stein, and Zhong [13].

Interval-Indexed LP Formulation

In the first step we write an interval-indexed linear programming relaxation for the co-flow scheduling problem similar to that for the concurrent open shop problem by Wang and Cheng [16].

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

$$\bar{C_1} \leq \bar{C_2} \ldots \leq \bar{C_n} \tag{21}$$

Let V_j denote the maximum load on any port by the *first* j co-flows 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. [13] prove that these V_j values provide a good approximation for the optimal completion times of the co-flows. In particular, they show the following where C_j^* denotes the completion time of co-flow j in an optimal schedule.

$$\sum_j w_j V_j \leq \frac{16}{3} \sum_j w_j C_j^* \tag{22}$$

Grouping Co-flows

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

Algorithm 1

- For $l = 1, 2, \ldots$
  - Wait until the last co-flow in S_l is released.
  - Group all co-flows in S_l and schedule as per Algorithm 1 in [13]. 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 [13]).

Proposition 1. *For any co-flow j, let $C_j(alg)$ denote the completion time of co-flow 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 (22) together imply the following theorem (Theorem 1 in [13]).

Theorem 2. *There exists a deterministic polynomial time 67/3 approximation algorithm for co-flow 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 [13] 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 [13] 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 [13] can be corrected via a similar modification.

E.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 co-flows are ordered as per Equation (21).

As shown by Leung, Li, and Pinedo (Theorem 13 in [10]), the analysis of Wang and Cheng [16] can be extended to the case of general release times to obtain the following.

$$\sum_j w_j \bar{C}_j \leq \frac{19}{3} \sum_j w_j C_j^* \tag{23}$$

This is analogous to Lemma 3 in [13] 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 co-flows taken together (as per the ordering).

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

$$\bar{C}_j \geq r_j \tag{24}$$

$$\bar{C}_j \geq V_j \tag{25}$$

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

Algorithm

- For $l = 1, 2, \ldots$
  - Wait until the last co-flow in S_l is released AND all co-flows in S_{l-1} have finished. (whichever is later).

- Group all co-flows in S_l and schedule as per Algorithm 1 in [13]. 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 co-flows 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 co-flow 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 co-flows 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 co-flows in group S_{l+1} start executing latest at time 2^l. We know that all these co-flows require at most $V_k \leq \bar{C}_k \leq 2^l$ time units to complete. As a result, all the co-flows in this group are scheduled by time $2^l + 2^l = 2^{l+1}$.

And thus the claim follows by induction.

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

Proof. Consider any co-flow 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 co-flow scheduling with arbitrary release times.*

Proof. Claim E.1 and Equation (23) together imply a $\frac{76}{3}$-approximation algorithm for co-flow scheduling with release times.

F Counterexample to Claim by Luo et al. [11]

Luo et al. [11] claim a 2-approximation algorithm for the co-flow 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 [11].

Claim (Restated from [11]). Given two co-flows G_k and G_l, we can find a feasible schedule such that for both the co-flows such that $C_k + C_l = \min\{\Delta(G_k) + \Delta(G_k + G_l), \Delta(G_l) + \Delta(G_k + G_l)\}$.

Counterexample

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

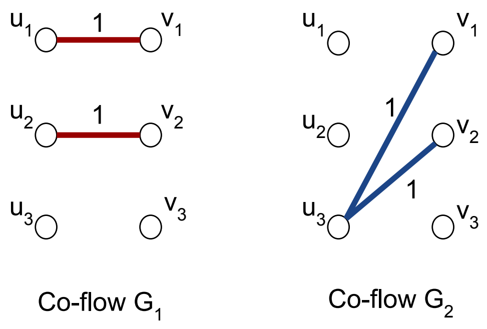

Fig. 5. Simple counterexample to Claim F

On the other hand, as seen in Figure 6, if co-flow G_1 is scheduled so that $C_1 = \Delta(G_1) = 1$, then the matching constraints force co-flow G_2 to have completion time $C_2 = 3$. On the other hand, delaying one edge of co-flow 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.

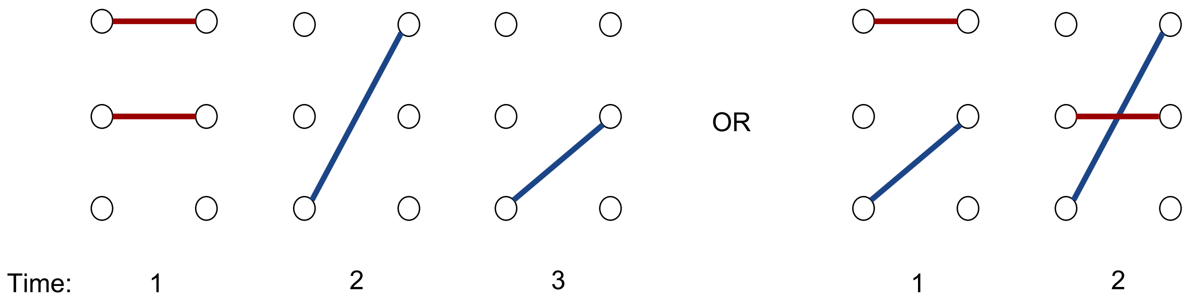

Fig. 6. Simple counterexample to Claim F