On Scheduling Coflows

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

[1] University of Maryland, College Park, USA
{saba,samir,styang}@cs.umd.edu
[2] Google, Mountain View, USA
mpurohit@google.com

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 $\{d^j_{io}\}_{i\in m, o\in m}$ 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 with release times, and a deterministic 4-approximation for the case without release time.

Keywords: Coflow scheduling · Concurrent open shop

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 [9], Hadoop [1], and Spark [19] 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

This work is supported by NSF grant CNS 156019.

F. Eisenbrand and J. Koenemann (Eds.): IPCO 2017, LNCS 10328, pp. 13–24, 2017.
DOI: 10.1007/978-3-319-59250-3_2

[5] 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.

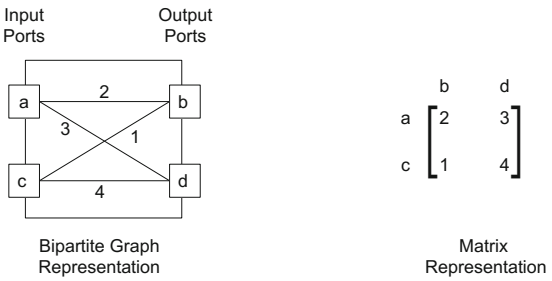

Fig. 1. An example coflow over a 2 × 2 switch. The figure illustrates two equivalent representations of a coflow - (i) as a weighted, bipartite graph over the set of ports, and (ii) as a $m \times m$ integer matrix.

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^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 coflows, i.e. minimize $\sum_j w_j C_j$.

1.1 Related Work

Chowdhury and Stoica [5] introduced the coflow abstraction to describe the prevalent communication patterns in data centers. Since then coflow scheduling has been a topic of active research [6,7,15,20] 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 [7] and online [6] 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 $(2-\epsilon)$ [3,17].

For the special case when all coflows have zero release time, Qiu, Stein and Zhong [15] 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 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 the full version [2], 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 the full version [2], their proof too is flawed and the result does not hold.

In a recent work, Khuller et al. [11] 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 Qiu et al. [15] whose techniques yield an approximation factors of $\frac{76}{3} = 25.33$ and $(8+\frac{16\sqrt{2}}{3}) \approx 15.54$ (see the full version [2]) respectively for the two cases. 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.

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

Due to space constraints, we defer all proofs to the full version [2].

1.3 Connection to Concurrent Open Shop

The coflow scheduling problem generalizes the well-studied concurrent open shop problem [4,10,12,14,18]. 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 [4,10,12] and a 3-approximation algorithm for arbitrary release times [10,12]. Mastrolilli et al. [14] show that a simple greedy 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 [7,15]. 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 is not be true for coflows [7]. In fact, even without release times, the best known approximation algorithm for scheduling coflows has an approximation factor of ≈ 15.54 [15], 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 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 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 $w_{G_j}(e) = d^j_{io}$ represents that the coflow j requires d^j_{io} 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 the 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. We defer the proof to the full version [2].

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.*

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 [10,12] 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 parallel 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 Sect. 4.1 to obtain a good permutation of the coflows. This primal dual algorithm is inspired by Davis et al. [8] 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.

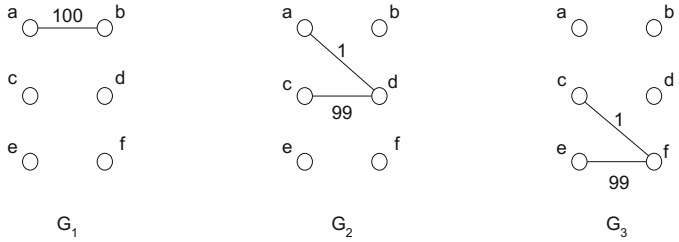

Fig. 3. Example that illustrates sequentially scheduling coflows independently can lead to bad schedules.

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 $\Delta(G_1) + \Delta(G_2) + \Delta(G_3) = 300 > \Delta(G_1 \cup G_2 \cup G_3) = 101$.

One key insight is that sequentially scheduling coflows one after another may waste resources. 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 Fig. 3 when all release times are zero; 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 in Sect. 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 Sect. 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 Theorems 1 and 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 second stage, we show that scheduling the coflows sequentially in this ordering after some postprocessing steps yields a provably good coflow schedule.

$$\max \sum_{j \in J} \sum_{i \in M} \alpha_{i,j}(r_j + L_{i,j}) + \sum_{i \in M} \sum_{S \subseteq J} \beta_{i,S} f_i(S)$$

$$\text{subject to,} \quad \sum_{i \in M} \alpha_{i,j} + \sum_{i \in M} \sum_{S/j \in S} L_{i,j} \beta_{i,S} \leq w_j \qquad \forall j \in J$$

$$\alpha_{i,j} \geq 0 \qquad \forall j \in J, i \in M$$

$$\beta_{i,S} \geq 0 \qquad \forall i \in M, \forall S \subseteq J$$

Fig. 4. Dual of LP_1

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 Fig. 2 and its dual (Fig. 4) in the design and analysis of the algorithm.

Our algorithm works as follows. We build up a permutation of the coflows 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_μ be the load on machine 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 Sect. 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 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

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 machine i
for $k = n, n-1, \cdots, 1$ **do**
  $\mu(k) = \arg\max_{i \in M} L_i$; // determine the machine 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;

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

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* $C_j(alg) \leq 2\Delta(\bigcup_{k \leq j} G_k)$.

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 $C_j(alg) \leq \max_{k \leq j} r_k + 2\Delta(\bigcup_{k \leq j} G_k)$.*

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.*

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

Proof Sketch. Algorithm 1 judiciously sets the dual variables such that the dual constraint for any coflow j is tight. Analyzing the cost of schedule obtained in terms of the dual variables yields the lemma. The formal proof is available in the full version [2].

Lemmas 3 and 4 along with Lemma 5 and an appropriate choice of κ now give the desired theorems. Proof in journal version.

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 coow scheduling without release times.*

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.

$$\overline{C_1} \leq \overline{C_2} \leq \ldots \leq \overline{C_n} \tag{5}$$

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

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

Lemmas 3 and 4 along with Lemma 6 give alternative proofs for Theorems 1 and 2.

References

1. https://hadoop.apache.org
2. http://cs.umd.edu/~samir/ipco17-fullversion.pdf
3. Bansal, N., Khot, S.: Inapproximability of hypergraph vertex cover and applications to scheduling problems. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6198, pp. 250–261. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14165-2_22
4. Chen, Z.-L., Hall, N.G.: Supply chain scheduling: conflict and cooperation in assembly systems. Oper. Res. **55**(6), 1072–1089 (2007)
5. Chowdhury, M., Stoica, I. Coflow: a networking abstraction for cluster applications. In: ACM Workshop on Hot Topics in Networks, pp. 31–36. ACM (2012)
6. Chowdhury, M., Stoica, I.: Efficient coflow scheduling without prior knowledge. In: SIGCOMM, pp. 393–406. ACM (2015)
7. Chowdhury, M., Zhong, Y., Stoica, I.: Efficient coflow scheduling with varys. In: SIGCOMM, SIGCOMM 2014, pp. 443–454. ACM, New York (2014)
8. Davis, J.M., Gandhi, R., Kothari, V.H.: Combinatorial algorithms for minimizing the weighted sum of completion times on a single machine. Oper. Res. Lett. **41**(2), 121–125 (2013)
9. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
10. Garg, N., Kumar, A., Pandit, V.: Order scheduling models: hardness and algorithms. In: Arvind, V., Prasad, S. (eds.) FSTTCS 2007. LNCS, vol. 4855, pp. 96–107. Springer, Heidelberg (2007). doi:10.1007/978-3-540-77050-3_8
11. Khuller, S., Li, J., Sturmfels, P., Sun, K., Venkat, P.: Select, permute: an improved online framework for scheduling to minimize weighted completion time (2016) (Submitted)
12. Leung, J.Y.-T., Li, H., Pinedo, M.: Scheduling orders for multiple product types to minimize total weighted completion time. Discrete Appl. Math. **155**(8), 945–970 (2007)
13. Luo, S., Yu, H., Zhao, Y., Wang, S., Yu, S., Li, L.: Towards practical, near-optimal coflow scheduling for data center networks. IEEE Trans. Parallel Distrib. Syst. PP(99), 1 (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. Oper. Res. Lett. **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 2015, pp. 294–303. ACM, New York (2015)
16. Queyranne, M.: Structure of a simple scheduling polyhedron. Math. Program. **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. Wang, G., Cheng, T.E.: Customer order scheduling to minimize total weighted completion time. Omega **35**(5), 623–626 (2007)
19. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. HotCloud **10**, 10 (2010)
20. 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)

http://www.springer.com/978-3-319-59249-7

Integer Programming and Combinatorial Optimization
19th International Conference, IPCO 2017, Waterloo, ON, Canada, June 26-28, 2017, Proceedings
Eisenbrand, F.; Koenemann, J. (Eds.)
2017, XI, 456 p. 34 illus., Softcover
ISBN: 978-3-319-59249-7