Matroid Coflow Scheduling

Sungjin Im
University of California at Merced, USA
sim3@ucmerced.edu

Benjamin Moseley
Carnegie Mellon University, Pittsburgh, PA, USA
moseleyb@andrew.cmu.edu

Kirk Pruhs
University of Pittsburgh, PA, USA
kirk@cs.pitt.edu

Manish Purohit
Google, Mountain View, CA, USA
mpurohit@google.com

Abstract

We consider the matroid coflow scheduling problem, where each job is comprised of a set of flows and the family of sets that can be scheduled at any time form a matroid. Our main result is a polynomial-time algorithm that yields a 2-approximation for the objective of minimizing the weighted completion time. This result is tight assuming $P \neq NP$. As a by-product we also obtain the first $(2+\epsilon)$-approximation algorithm for the preemptive concurrent open shop scheduling problem.

2012 ACM Subject Classification Theory of computation → Scheduling algorithms

Keywords and phrases Coflow Scheduling, Concurrent Open Shop, Matroid Scheduling

Digital Object Identifier 10.4230/LIPIcs.ICALP.2019.145

Category Track C: Foundations of Networks and Multi-Agent Systems: Models, Algorithms and Information Management

Funding *Sungjin Im*: Supported in part by NSF grants CCF-1409130 and CCF-1617653.
Benjamin Moseley: Supported in part by a Google Research Award and NSF grants CCF-1617724, CCF-1733873 and CCF-1725543.
Kirk Pruhs: Supported in part by NSF grants CCF-1421508 and CCF-1535755, and an IBM Faculty Award.

Acknowledgements We thank the anonymous reviewers for their thorough reviews and many helpful suggestions.

1 Introduction

Coflows were introduced in [5] as: "We propose coflows, a networking abstraction to express the communication requirements of prevalent data parallel programming paradigms. Coflows make it easier for the applications to convey their communication semantics to the network, which in turn enables the network to better optimize common communication patterns." Data parallel application frameworks such as MapReduce [9] and Spark [31] have a unique processing pattern that interleaves local computation with communication across machines. Due to the size of the large data sets processed, communication often tends to be a bottleneck in the performance of these platforms and the coflow model abstracts out this bottleneck. Theoretical work on coflow scheduling has primarily focused on the switch model (also called matching model) where the underlying network is assumed to have full-bisection bandwidth and the set of flows that can be scheduled at any time step is restricted to be form a matching.

While there are several reasonable formulations/models of scheduling coflows, the following will be convenient for our purposes. The input consists of a collection J of jobs, where each job $j \in J$ is comprised of a set U_j of tasks (also called flows), a non-negative integer w_j and a release time r_j. Each task $e \in U_j$ has a processing requirement p_e. For example, in the setting of a network supporting MapReduce [9] computations, each job could be a MapReduce job, and a task/flow could represent a required communication within a shuffle phase of a job. Let $U = \cup_{j \in J} U_j$ be the collection of all tasks. Further the input contains a downward-closed set system $\mathcal{M} = (U, \mathcal{I})$. Here $\mathcal{I} \subseteq 2^U$ and elements of $\mathcal{I}$ are called the *independent sets* of $\mathcal{M}$. Conceptually a collection of tasks is independent (and in $\mathcal{I}$) if they can be simultaneously scheduled by the network. A feasible output is a schedule σ that schedules all the flows. That is for each integer time t, σ specifies a collection σ_t of tasks processed/scheduled at time t. In order to be feasible, σ must satisfy the conditions that:

- every task $e \in U$ is scheduled for p_e time steps, and
- at each time t, the scheduled tasks/flows σ_t are in $\mathcal{I}$.

A job j completes at the first time C_j such that every task in U_j has been scheduled fully. The objective is to minimize the total weighted completion time of the jobs. That is, to minimize $\sum_j w_j C_j$.

In this paper, we consider coflow scheduling when the set system $\mathcal{M}$ forms a matroid. The starting point for our investigations is the question whether there is an algorithm to effectively schedule coflows that involve aggregating information, stored at various locations in a network, to a common sink location. Such gathering communication patterns were identified as common in [5]. We model aggregation communications by assuming that for each job j, U_j is a collection of locations in the network where the units of information needed for job j are stored. It is natural to define the independent sets to be locations that can simultaneously be routed to the sink without violating any capacity constraint of the network. In this case, $\mathcal{M}$ is a matroid, and more specifically, a gammoid. Note that the symmetric problem, of disseminating data from a fixed location to various locations in the network, is also common, and essentially equivalent to the aggregation problem.

The matroid coflow scheduling problem as defined here also naturally captures a number of well-studied scheduling problems.

- Parallel Identical Machines Scheduling: Each job j has a single task. The matroid $\mathcal{M} = (U, \mathcal{I})$ is the uniform matroid of rank m, i.e., any set of m jobs can be scheduled in parallel.
- (Preemptive) Concurrent Open Shop Scheduling: In the concurrent open shop scheduling problem, each job j comprises of m tasks, one on each machine, i.e. $U_j = \{t_{ij}\}_{i=1}^{m}$. Task t_{ij} needs to be scheduled for time p_{ij} and the job is completed when all its tasks are completed. To model this setting, consider $T_i = \{t_{ij}\}_{j=1}^{n}$ to be set of all tasks that need to be scheduled on machine i. $\mathcal{M}$ is a partition matroid that ensures that a set S of tasks is independent if and only if $|S \cap T_i| \leq 1$ for each machine i.

1.1 Our Contributions

We first consider coflow scheduling on unit length tasks when $\mathcal{M}$ is a matroid. Our main result is:

▶ **Theorem 1.** *There is a deterministic polynomial-time algorithm for coflow scheduling with unit length tasks, when $\mathcal{M}$ is a matroid, that is 2-approximate with respect to the objective of minimizing total weighted completion time.*

We note that Theorem 1 can be extended to the case that tasks may have arbitrary processing times, albeit at a slight loss in the approximation factor.

▶ **Theorem 2.** *There is a deterministic polynomial-time algorithm for coflow scheduling with arbitrary length tasks, when $\mathcal{M}$ is a matroid, that is $(2+\epsilon)$-approximate with respect to the objective of minimizing total weighted completion time, for any constant $\epsilon > 0$.*

As with all the approximation results for coflow scheduling in the literature, our algorithm is based on rounding a natural time-indexed linear program. Intuitively the rounding extracts a deadline C_j^* for each job j. This time is roughly $1/\lambda$ times later than the first time when every task in U_j has been scheduled at least to the extent λ in the solution to LP. Here the value of λ is randomly chosen. The expected value of C_j^* is shown to be at most twice the fractional completion time for j in the solution to LP; this "stretching" (also called slow-motion) idea has been used in other scheduling contexts [12, 22, 27]. This can be viewed as deriving from the LP a fractional schedule where each job j is fully completed by time C_j^*. Then, we observe that the problem of scheduling tasks to meet the C_j^* deadlines can be expressed as a matroid intersection problem. As the matroid intersection polytope is integral [26], one can find an integral schedule meeting these deadlines. Finally, by derandomizing the random choice of λ, we derive our main theorem.

The approximation guarantee in Theorem 1 is tight assuming $P \neq NP$. This is because it is NP-hard to approximate the total weighted completion time for concurrent open shop (even with unit sized tasks) within a factor of $2-\epsilon$ [23], and this problem is a special case of matroid coflow scheduling, where the matroid is a partition matroid. Somewhat surprisingly, even for the concurrent open shop scheduling with release times, the previous best known approximation factor was 3 [10, 17]. (See also additional discussion in [2].) Thus, Theorem 2 immediately yields an improved approximation algorithm for preemptive concurrent open shop with arbitrary release times.

▶ **Corollary 3.** *There is a deterministic, polynomial-time $(2+\epsilon)$ approximation algorithm for the preemptive concurrent open shop scheduling problem when jobs have arbitrary release times, for any constant $\epsilon > 0$. If all the release times and processing requirements are polynomially bounded, then the approximation guarantee improves to 2.*

We believe our primary technical contribution is the high-level approach to reduce a weighted completion time scheduling problem to a deadline-constrained scheduling problem. Our approach to first extract a deadline for each job from the LP solution and then finding an integer schedule that meets those deadlines can be viewed as a strict generalization of processing jobs in increasing order of their completion time derived from the LP, which has been a very common rounding tool in scheduling literature; e.g. [21, 28, 2]. Our novel approach allows us to handle the matroid constraint, which we believe is natural and quite general.

1.2 Related Results

Most of the theoretical/algorithmic work on coflows has been on matching coflows [20, 16, 15, 2, 1]. These results essentially abstract out the network by modeling the network as an n-by-n switch, or equivalently a complete bipartite graph, and by modeling supportable flows by matchings in the graph. This is well motivated in practice as the networks in many data centers are hierarchical, with higher network elements having higher capacities. Thus a matching between servers at leaves of the network is a not unreasonable approximation of a communication supportable by the network. We note that matching coflows correspond to coflows in our framework when the set system $\mathcal{M}$ is an intersection of two partition matroids. The first constant (16.54) approximation for coflow scheduling in this model was given in [20]. Currently the best known approximation ratios are 5 for when jobs may have variable release times, and 4 when all jobs arrive at time 0 [2, 29], respectively. Note that the 2-approximation algorithms claimed in [18] and [11] are both flawed; see [2] and [11] for the discussion of the flaws.

Jahanjou et al. [14] consider several problems where there is an underlying network with capacities on the edges. If the tasks are paths in the network, and $\mathcal{I}$ consists of collections of paths that don't collectively violate any edge capacity, then their work gives a algorithm for producing a fractional schedule (which is equivalent to time being continuous) that is $O(1)$-approximate with respect to total weighted completion time. If the tasks are (source, sink) pairs in the network, and $\mathcal{I}$ consists of collections of (source, sink) pairs that can be simultaneously routed without violating any edge capacity, then their work gives a algorithm for producing a fractional unsplittable schedule that is $O(\log E/\log\log E)$-approximate with respect to total weighted completion time, together with a matching hardness result; here, E is the number of edges. Our work is not comparable to theirs since different constraints are addressed and our focus is on integer schedules in contrast to theirs on fractional schedules.

Coflow scheduling is a generalization of the classical concurrent open shop scheduling problem [3, 4, 10, 17, 19, 23, 30]. Several 2-approximation algorithms were shown [4, 10, 17] via LP rounding. Matching hardness results were shown in [3, 23]. When jobs have different release times, the same LP relaxations yielded 3-approximations [10, 17]. Later, [19] gave a simple greedy algorithm that matches the best approximation ratio when all jobs arrive at time 0. Recently, [2] gave a combinatorial 3-approximation via a primal-dual analysis when jobs have non-uniform release times.

Coflow scheduling has been actively studied within the networking community; some examples include [5, 6, 7, 18, 32].

1.3 Organization

The rest of the paper is organized as follows. In Section 2 we give some basic definitions and notation. In Section 3 we give the linear programming formulation. In Section 4 we explain how to round a solution to the linear program. In Section 5 we discuss the derandomization. In Section 6, we discuss the extension to tasks with variable processing times.

2 Definitions and Notations

We first consider the matroid coflow scheduling problem with unit length tasks. We will discuss three types of schedules, and two types of objectives. In a discrete-time schedule, we consider that time is divided into unit length intervals (also called time slots), and the schedule specifies the set of jobs processed during each time slot. We let time slot t refer to the interval of time $(t-1, t]$. In an *integer* discrete-time schedule, at each time slot t, an independent set in the matroid is scheduled. In a *fractional* discrete-time schedule, at each time slot t, a convex combination of independent sets from the matroid are scheduled. In other words, in such a fractional schedule, the set of tasks scheduled at time slot t can be expressed as $\sum_{S\in\mathcal{I}}\alpha_S 1_S$, where $\sum_{S\in\mathcal{I}}\alpha_S = 1$, and 1_S is the characteristic vector corresponding to independent set $S\in\mathcal{I}$. A valid feasible solution is restricted to be an integer discrete-time schedule. On the other hand, during our analysis, we will also consider *continuous* schedules. A continuous schedule specifies an independent set of tasks to be scheduled at each instantaneous time τ (as opposed to during a unit-length time slot).

The completion time C_j of a job j is the first time when all tasks in U_j have been completed. We let $q_e(t) : [0,T] \to \{0,1\}$ denote an indicator function defined for each task $e \in U$, where $q_e(t) = 1$ if and only if task e (more precisely an independent set including e) is scheduled at time t in σ. We let $Q_e(t) = \int_{\tau=0}^{t} q_e(\tau) d\tau$ denote the extent to which task e is scheduled by time t. Let $\tilde{C}_j(v)$ denote the first time when every task in U_j has been scheduled by extent at least v. The fractional completion time of job j is then $\tilde{C}_j = \int_{v=0}^{\infty} \tilde{C}_j(v) dv$. We will use $\textsc{cost}(\mathsf{LP})$ to denote the optimum objective of the LP, which we will describe soon.

3 Linear Program

In this section we give a linear programming formulation LP of our matroid coflow problem when tasks have unit lengths. Let $x_{j,t}$ be an indicator variable that specifies whether job j completes at time t. For a task $e \in U_j$, let $y_{e,t}$ be an indicator variable that specifies whether task e is assigned to time slot t. Let $\rho(S)$ be the rank function of the matroid.[1] Let $T = |U|$ be an upper bound on the time by which all tasks can be completed. The formulation of LP is then:

$$\begin{array}{lll} \mathsf{LP}: & \min \sum_{j \in J} w_j \sum_{t \in [T]} t \cdot x_{j,t} & \\ \text{s.t.} \quad \forall j \in J, & \sum_t x_{j,t} = 1 & (1) \\ \forall j \in J \text{ and } \forall e \in U_j \text{ and } \forall t \in [T], & \sum_{s \le t} y_{e,s} \ge \sum_{s \le t} x_{j,s} & (2) \\ \forall S \subseteq U \text{ and } \forall t \in [T], & \sum_{e \in S} y_{e,t} \le \rho(S) & (3) \\ \forall j \in J \text{ and } \forall e \in U_j \text{ and } \forall t \in [r_j - 1], & y_{e,t} = 0 & (4) \\ & \mathbf{x}, \mathbf{y} \ge 0 & (5) \end{array}$$

Constraint (1) ensures that every job is scheduled. Constraint (2) ensures that all tasks of a job j are scheduled to at least the extent that j is completed by time t. Constraint (3) ensures that at any time step t, the set of tasks assigned to t form an independent set in the given matroid. Constraint (3) is the only constraint set that can potentially have a super-polynomial size. However, for each fixed time t, the constraint is just a polymatroid, and therefore, admits an efficient separation oracle [8, 24, 13]. In case that there are arrival/release times, constraint (4) ensures that no tasks in U_j are processed before j's release time r_j. The objective of LP is fractional weighted completion time.

Note that a solution to LP can be viewed as a fractional discrete schedule. We will use $X_{j,t} := \sum_{s \le t} x_{j,s}$ to denote the extent to which job j has been processed by time t, and use $Y_{e,t} := \sum_{s \le t} y_{e,s}$ to denote the extent to which task e has been processed by time t.

4 Rounding

In this section, we show how to round an optimal solution to LP to obtain a 2-approximate integral (discrete) schedule. For each job j and $v \in (0,1]$, define $\bar{C}_j(v) = \frac{1}{x_{j,t}}(v - X_{j,t-1}) + (t-1)$ if $v \in (X_{j,t-1}, X_{j,t}], t \in [T]$. Intuitively, $\bar{C}_j(v)$ is a linear interpolation of the discrete

[1] $\rho(S)$ is defined as $\max_{S' \subseteq S : S' \in \mathcal{I}} |S'|$.

times when job j is partially completed. We set a deadline $C_j^* = \lceil \frac{1}{\lambda}\bar{C}_j(\lambda)\rceil$ for each job j, where $\lambda \in (0,1]$ is randomly drawn according to the probability density function $f(v) = 2v$. A key portion of the analysis is to show that the expected value of each $w_j C_j^*$ is at most twice the contribution of job j to the LP objective.

To analyze the expected value of C_j^*, we construct several schedules from the LP solution. In Subsection 4.1, we will show how to convert a solution of LP to a continuous schedule σ. In Subsection 4.2 we show how to convert σ into a stretched schedule σ^λ, which is another continuous schedule parameterized by $\lambda \in (0,1]$. Finally, in Subsection 4.3 we will show how to convert this continuous schedule into (discrete-time) integer schedule with the same cost. We note that we construct schedules in Subsection 4.1 and 4.2 only for the sake of analysis. That is, we can obtain a 2-approximate integral discrete schedule only using the rounding algorithm in Subsection 4.3 with the deadlines $\{C_j^*\}_j$.

4.1 Constructing the Continuous Schedule σ

We construct a continuous schedule σ from the solution to LP. For each time t, we first decompose $\{y_{e,t}\}_{e\in U}$ into a convex combination $\sum_{S\in\mathcal{I}} \alpha_S 1_S$ of independent sets.[2] To create σ this convex combination is "smeared" across all instantaneous times during $(t-1,t]$. That is, in σ each independent set S is scheduled for $\alpha_i(\tau_2 - \tau_1)$ time units during each infinitesimal time interval $(\tau_1, \tau_2] \in (t-1,t]$. This is formalized in Proposition 4. In Lemma 5 we show that the first time when a job j is scheduled to extent v in σ is at most $\bar{C}_j(v)$. In Lemma 6 we show that the fractional weighted completion time of σ is a bit less than the objective value of the solution to LP. This is because any processing of job j done during $(t-1,t]$ has no effect until time t on the LP objective, whereas it can have effect on j's fractional weighted completion time of σ during $(t-1,t]$, before time t.

▶ **Proposition 4.** *Consider the schedule σ. For any integer $t \in [T]$ and $(\tau_1,\tau_2] \in (t-1,t]$, we have, $\int_{\tau=\tau_1}^{\tau_2} q_e(\tau)d\tau = y_{e,t}(\tau_2 - \tau_1)$.*

▶ **Lemma 5.** *Consider the schedule σ. For any j and $v \in (0,1]$,*

$$\tilde{C}_j(v) \le \bar{C}_j(v) =: \frac{1}{x_{j,t}}(v - X_{j,t-1}) + (t-1) \text{ if } v \in (X_{j,t-1}, X_{j,t}], t \in [T],$$

and $\tilde{C}_j(0) = 0$.

Proof. By definition, we have $\tilde{C}_j(0) = 0$, so let us assume that $v > 0$. We first show that $\tilde{C}_j(X_{j,t}) = t$. Due to constraint (2), $Y_{e,t} \ge X_{j,t}$ for all $e \in U_j$. Thus, by construction of σ, all tasks in U_j are processed by at least $X_{j,t}$ by time t, i.e., $Q_e(t) \ge X_{j,t}$, meaning that $\tilde{C}_j(X_{j,t}) \le t$. We also have that $\tilde{C}_j(X_{j,t}) \ge t$ since we know by the optimality of the LP solution that $Y_{e,t} = X_{j,t}$ for some $e \in U_j$, therefore, $Q_e(t) = X_{j,t}$. Thus, we have $\tilde{C}_j(X_{j,t}) = t = \bar{C}_j(X_{j,t})$.

Now consider an arbitrary $v \in (0,1]$. Let $t \in [T]$ be such that $v \in (X_{j,t-1}, X_{j,t}]$. Then, it follows that $x_{j,t} \ne 0$. Thus, from the above argument, we have $\tilde{C}_j(X_{j,t}) = t$. Let $t_v := \bar{C}_j(v)$ for notational convenience. We want to show $\tilde{C}_j(v) \le t_v$. By Proposition 4 and construction of σ, we know that the extend to which e is processed by time t_v,

$$Q_e(t_v) = Y_{e,t-1} + y_{e,t}(t_v - (t-1)) = Y_{e,t-1} + \frac{y_{e,t}}{x_{j,t}}(v - X_{j,t-1})$$

[2] This is possible because $\{y_{e,t}\}_e$ lies in the polymatroid associated with the matroid rank function ρ due to constraint (3). It is well-known that this polymatroid is equivalent to the independence set polytope of the matroid, meaning that $\{y_{e,t}\}_e$ can be expressed as a convex combination of characteristic vectors of some independent sets. For more details, see Chapter 44 of [25].

First, if $y_{e,t} \geq x_{j,t}$, we immediately have $Q_e(t_v) \geq v + Y_{e,t-1} - X_{j,t-1} \geq v$ due to constraint (2). Otherwise, since $\frac{1}{x_{j,t}}(v - X_{j,t-1}) \leq 1$, fixing the value of $Y_{e,t} = Y_{e,t-1} + y_{e,t}$, the right-hand-side decreases when we increase $y_{e,t}$. Therefore, we have, $Q_e(t) \geq Y_{e,t-1} - (x_{j,t} - y_{e,t}) + \frac{x_{j,t}}{x_{j,t}}(v - X_{j,t-1}) = v + Y_{e,t} - X_{e,t} \geq v$, again due to constraint (2). Hence, we have $Q_e(t_v) \geq v$ for all $e \in U_j$, which immediately yields $\tilde{C}_j(v) \leq t_v$. ◀

▶ **Lemma 6.** $\sum_{j \in J} w_j \int_{v=0}^{1} \bar{C}_j(v)dv = \textsc{cost}(\mathsf{LP}) - \sum_{j \in J} w_j/2$

Proof. It suffices to show that $\int_{v=0}^{1} \bar{C}_j(v)dv = \sum_{t \in [T]} t \cdot x_{j,t} - 1/2$, since summing this equation over all $j \in J$ multiplied by their weight w_j yields the lemma.

$$\int_{v=0}^{1} \bar{C}_j(v)dv = \sum_{t \in [T]} \int_{v=X_{j,t-1}}^{X_{j,t}} \bar{C}_j(v)dv = \sum_{t \in [T]: x_{j,t} \neq 0} \int_{v=X_{j,t-1}}^{X_{j,t}} \bar{C}_j(v)dv$$
$$= \sum_{t \in [T]: x_{j,t} \neq 0} \int_{v=X_{j,t-1}}^{X_{j,t}} \left(\frac{1}{x_{j,t}}(v - X_{j,t-1}) + (t-1) \right) dv$$
$$= \sum_{t \in [T]: x_{j,t} \neq 0} \left[\frac{1}{2} x_{j,t} + (t-1)x_{j,t} \right] = -\frac{1}{2} + \sum_{t \in [T]: x_{j,t} \neq 0} t \cdot x_{j,t},$$

where the last equality follows from constraint (1). ◀

4.2 Constructing the Stretched Schedule σ^λ

To construct σ^λ from σ we "stretch" the schedule σ by a factor of $1/\lambda$. More precisely, if an independent set S is scheduled in σ during an infinitesimal interval $(\tau_1, \tau_2]$, the same independent set is scheduled in σ^λ during $(\tau_1/\lambda, \tau_2/\lambda]$. In Lemma 7 we show that σ^λ completes job j by time $C_j^* = \lceil \frac{\bar{C}_j(\lambda)}{\lambda} \rceil$. In Lemma 8 we upper bound the expected cost of $\sum_j w_j C_j^*$ by twice $\textsc{cost}(\mathsf{LP})$.

▶ **Lemma 7.** *The schedule σ^λ completes every job j by time C_j^*.*

Proof. Lemma 5 shows that $\tilde{C}_j(v) \leq \bar{C}_j(v)$ for all $v \in (0,1]$, meaning that every task in U_j is completed by v units by time $\bar{C}_j(v)$ in σ. Thus, in the stretched schedule σ^λ, every job j completes by time $\bar{C}_j(\lambda)/\lambda$, for any value of $\lambda \in (0,1]$. ◀

▶ **Lemma 8.** $\mathbb{E}[\sum_{j \in J} w_j C_j^*] \leq 2\ \textsc{cost}(\mathsf{LP})$.

Proof. First note that

$$\sum_{j \in J} w_j \mathbb{E}[\bar{C}_j(\lambda)/\lambda] = \sum_{j \in J} w_j \int_{v=0}^{1} \bar{C}_j(v)/v \cdot (2v)dv = 2 \sum_{j \in J} w_j \int_{v=0}^{1} \bar{C}_j(v)dv \tag{6}$$

Thus, we have,

$$\mathbb{E}\Big[\sum_{j \in J} w_j C_j^*\Big] = \mathbb{E}\Big[\sum_{j \in J} w_j \lceil \frac{1}{\lambda} \bar{C}_j(\lambda) \rceil\Big] \leq \Big(\mathbb{E}\Big[\sum_j w_j \frac{1}{\lambda} \bar{C}_j(\lambda)\Big]\Big) + \sum_j w_j$$
$$= 2 \sum_j w_j \int_{v=0}^{1} \bar{C}_j(v)dv + \sum_j w_j \qquad \text{[Eqn. (6)]}$$
$$= 2\Big(\textsc{cost}(\mathsf{LP}) - \sum_j w_j/2\Big) + \sum_j w_j \qquad \text{[Lemma 6]}$$
$$= 2\ \textsc{cost}(\mathsf{LP})$$
◀

4.3 Constructing a Discrete Integer Schedule

Let $y^*_{e,t}$ denote how much task e is processed during time interval $(t-1, t]$. In other words, task e appears in $y^*_{e,t}$ units of independents sets scheduled in σ^λ during the time interval. Then, $\{y^*_{e,t}\}_{e \in U, t \in [T]}$ satisfies the following:

1. For all $j \in J$ and $e \in U_j$, $\sum_{t \in [C^*_j] \setminus [r_j - 1]} y^*_{e,t} = 1$; and .
2. For all $S \subseteq U$ and for all $t \in [T]$, $\sum_{e \in S} y^*_{e,t} \leq \rho(S)$,

where the second holds true since $\{y^*_{e,t}\}_{e \in U}$ can be expressed as a convex combination of independent sets scheduled during time interval $(t-1, t]$, and therefore, lies in the matroid polytope. We now interpret $\{y^*_{e,t}\}$ as a fractional point in the intersection of two matroid polytopes. We create the following two matroids. The new universe U' is defined as $U' := \{(e, t) \mid t \in [T], j \in J, e \in U_j \text{ s.t. } r_j \leq t \leq C^*_j\}$. The first matroid M_1 is a partition matroid that forces to choose at most one element out of $\{(e, t)\}_t$, for each $e \in U$. Intuitively, this ensures that no task is scheduled more than once across times. The second matroid ensures that elements scheduled at each time t forms an independent set in $\mathcal{I}$. The following lemma formally defines the second matroid and shows that it is indeed a matroid.

▶ **Lemma 9.** *Define $\mathcal{I}_2 \subseteq 2^{U'}$ such that $S' \subseteq U'$ is in $\mathcal{I}_2$ if and only if for any $t \in [T]$, $\{e \mid (e, t) \in S'\} \in \mathcal{I}$. Then, $M_2 = (U', \mathcal{I}_2)$ is a matroid.*

Proof. Let $\mathcal{I}_2$ denote the family of independent sets of M_2. It is straightforward to see that $\mathcal{I}_2$ is downward closed. Thus, it suffices to show that for any $A', B' \in \mathcal{I}_2$ such that $|A'| < |B'|$, there exists $(e, t) \in B' \setminus A'$ such that $A' \cup \{(e, t)\} \in \mathcal{I}_2$. Let $U'_t := \{(e, t) \mid j \in J, e \in U_j \text{ s.t. } r_j \leq t \leq C^*_j\}$ denote the subset of U' restricted to time t. Consider any fixed $A', B' \in \mathcal{I}_2$ such that $|A'| < |B'|$. Then, consider any fixed time t^* such that $|A' \cap U'_{t^*}| < |B' \cap U'_{t^*}|$; such a time t^* must exist since $\{U'_t\}_t$ partitions U'. Then, for some $(e^*, t^*) \in (B' \cap U'_{t^*}) \setminus (A' \cap U'_{t^*})$, it must be the case that $\{e^*\} \cup \{e \mid (e, t^*) \in A' \cap U'_{t^*}\} \in \mathcal{I}$. This is because B' has more elements than A' that are paired up with the fixed time t^*, and therefore, the set of elements appearing in $A' \cap U'_{t^*}$ remains independent with some e^* added. Further, for any other time t, the elements appearing in the pairs of A' associated with t remain unchanged, and therefore, is in $\mathcal{I}$. ◀

Then, it is easy to see that $\{y^*_{e,t}\}$ is a point that lies in the intersection of the polymatroids that are defined by M_1 and M_2. Further, $\{y^*_{e,t}\}$ belongs to the base polymatroid of M_1; so we have $\sum_{(e,t) \in U'} y^*_{e,t} = |U|$. Since the matroid intersection polytope is well-known to be integral [26], meaning that every vertex is an integer point, a maximum independent set in the intersection of M_1 and M_2 must have $|U|$ elements. Further, we can find such a maximum independent set in polynomial time. To recap, we have found $S' \in U'$ that is a base of M_1 and is independent in M_2. This set S' immediately gives the desired integer schedule where $\{e \mid (e, t) \in S'\}$ is scheduled at each time t. Indeed, due to S' being a base of M_1, every task in U_j is scheduled exactly once during time interval $[r_j, C^*_j]$. Further, S' being independent in M_2 ensures that the set of tasks scheduled at each time forms an independent set in $\mathcal{I}$.

5 Derandomization

In this section, we discuss how to derandomize the choice of $\lambda \in (0, 1]$, which was used to compute the deadlines for the jobs. This will complete the proof of Theorem 1. Let us first define *step* values. We say that $v \in (0, 1]$ is a step value if $\sum_{s \leq t} x_{j,s} = v$ for some $j \in J$ and integer $t \in [T]$ – in other words, exactly v fraction of some job j is completed by some integer time in the LP solution. Let V denote the set of all step values; $1 \in V$ by definition. Note that that $|V|$ is polynomially bounded in the input size, as the number of variables $x_{j,t}$ we consider in LP is at most $|J| \cdot |U|$.

Recall that in Lemma 8 we showed $\mathbb{E}[\sum_j w_j C_j^*] \leq 2\,\textsc{cost}(\mathsf{LP})$ when $C_j^* := \lceil \frac{1}{\lambda} \bar{C}_j(\lambda) \rceil$. This implies there exists a certain value of $\lambda \in (0, 1]$ such that $\sum_j w_j C_j^* \leq 2\textsc{cost}(\mathsf{LP})$. For the purpose of derandomization, it suffices to find λ such that $\sum_j w_j \bar{C}_j(\lambda)/\lambda \leq 2 \sum_j w_j \int_{v=0}^{1} \bar{C}_j(v) dv$; the equality is shown in equation (6) in expectation.

Towards this end, we aim to find $\lambda \in (0, 1]$ that minimizes $\sum_j w_j \bar{C}_j(\lambda)/\lambda$. Suppose λ was set to a value $v \in (v_1, v_2]$, where v_1 and v_2 are two adjacent step values in V. Consider any fixed job j. Let $t \in [T]$ be such that $v \in (X_{j,t-1}, X_{j,t}]$. By definition of step values, we have $(v_1, v_2] \subseteq (X_{j,t-1}, X_{j,t}]$. Thus, we have $\bar{C}_j(v)/v = \frac{1}{x_{j,t}}(1 - \frac{X_{j,t-1}}{v}) + \frac{t-1}{v}$. This becomes a linear function in z over $[1/v_2, 1/v_1)$ if we set $z = 1/v$. Therefore, we get a piece-wise linear function $g(z)$ by summing over all jobs multiplied by their weight and considering all pairs of two adjacent step values in V. We set λ to the the inverse of z's value that achieves the global minimum, which can be found in polynomial time.

6 Arbitrary Processing Times

In this section we show how to extend Theorem 1 to allow tasks with arbitrary processing times with a loss of $(1 + \epsilon)$ factor in the approximation ratio for any arbitrary constant $\epsilon > 0$. In this setting, each task e has an arbitrary integer size p_e and the task e completes when p_e independent sets including e are scheduled. As before, at each time we can schedule a set of tasks that is independent in the given matroid and a job completes when all its tasks complete.

6.1 Compact Linear Program

We first describe our new compact LP relaxation. Let $T := \sum_e p_e + \max_j r_j$, which is clearly an upper bound on the maximum time we need to consider. We define a set of times $\mathcal{T}$ that consists of polynomially many time steps. First, let $\mathcal{T}$ include every job's arrival time. Next, let $\mathcal{T}$ include all times appearing in $\{\lfloor (1 + \epsilon)^i \rfloor\}_{0 \leq i \leq \lceil \log_{1+\epsilon} T \rceil + 1}$. In words, $\mathcal{T}$ includes exponentially increasing time steps by a factor of $(1 + \epsilon)$ starting from 1 but includes no times greater than $(1 + \epsilon)^2 T$. Let $t_1 = 1, t_2, \ldots, t_k, \ldots, t_{K+1}$ denote the (integer) times in $\mathcal{T}$ in increasing order. Let $I_i := [t_i, t_{i+1})$ where $i \in [K]$. The idea is to rewrite LP compactly as follows by replacing time-indexed variables with interval-indexed variables.

$$\min \sum_{j \in J} w_j \sum_{i \in [K]} (t_{i+1} - 1) \cdot x_{j,i}$$

$$
\begin{aligned}
\text{s.t.} \quad & \forall e \in U, & \sum_{i \in [K]} (t_{i+1} - t_i) y_{e,i} = p_e & \qquad (7)\\
& \forall j \in J\ \forall e \in U_j\ \forall i \in [K], & \sum_{i' \leq i} y_{e,i'}/p_e \geq \sum_{i' \leq i} x_{j,i'} & \qquad (8)\\
& \forall S \subseteq U\ \forall i \in [K], & \sum_{e \in S} y_{e,i} \leq \rho(S) & \qquad (9)\\
& \forall j \in J\ \forall e \in U_j\ \forall i \in [K] \text{ s.t. } t_{i+1} \leq r_j, & y_{e,i} = 0 & \qquad (10)\\
& & \mathbf{x}, \mathbf{y} \geq 0 & \qquad (11)
\end{aligned}
$$

Here, variable $x_{j,i}$ can be viewed as the average fraction of job j that completes per unit time during I_i; so, when the job j completes during I_i for the first time, we have $\sum_{i' \leq i} x_{j,i'} = 1$. Likewise, $y_{e,i}$ has an analogous meaning for each task e but it denotes the average *unit* of task e that is processed per unit time during I_i. Constraint (7) ensures that all tasks complete eventually. Constraint (9) ensures that the average vector representing how much each task is processed per unit time during I_t lies in the polymatroid. Constraint (10) enforces that no tasks in U_j are processed before j's arrival time; this is possible since $\mathcal{T}$ includes all jobs arrival times. Before explaining constraint (8), we explain the objective. If all intervals, $\{I_i\}$ were of unit length, the objective would be exactly the fractional total weighted completion time. However, to make the LP compact, when job j completes by $x_{j,i}$ fraction during interval I_i, we pretend that the fraction completes at the end of I_i, i.e., $t_{i+1} - 1$. Thus, we overestimate the fractional objective; but since times in I_i differ by at most $(1 + \epsilon)$ factor, our overestimate is by a factor of at most $(1 + \epsilon)$. Finally, we discuss constraint (8), which caps each job's (cumulative) processed fraction at the analogous quantity of each task of the job, which is measured as how much the task has been processed divided by its processing time. We also note that this compact LP admits the same separation oracle as the one for LP.

6.2 Rounding

As before, we seek to round the optimal LP solution. Recall that we first obtained $C^*_j := \lceil \frac{1}{\lambda}\bar{C}_j \rceil$ and found an integer schedule that completes every job j before C^*_j. We observe that the first procedure is no issue. This is because we can interpret the solution to our compact LP as a solution to LP. To see this, when a task e is processed by δ amount, pretend that there exist p_e different tasks of unit size and they are processed equally by δ/p_e amount. Thus, we can compute $\bar{C}_j(v)$ efficiently for any value of $v \in (0, 1]$. The derandomization can be done similarly.

6.3 Finding An Integer Schedule

It now remains to find an integer schedule meeting the discovered deadlines, $\{C^*_j\}_{j \in J}$. We use essentially the same idea of reducing the problem to finding an integer solution to the intersection of two matroids. However, this reduction requires some careful modifications to be implemented in polynomial time. Also, we will aim to complete every job j by $(1+O(\epsilon))C^*_j$ meeting the deadline slightly loosely.

The main idea is to use the fact that the continuous schedule σ^λ meeting the deadlines $\{C^*_j\}$ only changes polynomially many times. This is because the continuous schedule σ before the stretching is identical at all times during each of the intervals $(0, t_1 - 1], (t_1 - 1, t_2], \ldots, (t_{K-1} - 1, t_K]$ – these intervals are stretched into $(0, (t_1 - 1)/\lambda], ((t_1 - 1)/\lambda, t_2/\lambda], \ldots, ((t_{K-1} - 1)/\lambda, t_K/\lambda]$, respectively. We split the interval including the time $T' = |U|^2/\epsilon^2$ into two, the left one ending at $|U|^2/\epsilon^2$ and the right one starting at $|U|^2/\epsilon^2$. Here, assume that $1/\epsilon$ is an integer. We also add time $\bar{C}_j(\lambda)/\lambda$ for every $j \in J$ and split the intervals accordingly. To simplify the notation, we recycle the notations I_i. By reindexing the resulting intervals and merging some initial intervals, we have $I_0 := (0, T']$, I_1, I_2, ..., $I_{K'}$. We say that an interval is small if its starting time or ending time is not a power of $(1 + \epsilon)$ divided by λ; more precisely, $((t_{i-1} - 1)/\lambda, t_i/\lambda]$ is small if t_{i-1} or t_i is not a power of $(1 + \epsilon)$ divided by λ. Note that there are at most $4|J| + 4 \leq 8|J| \leq 8|U|$ small intervals since each job's arrival time and deadline together can create at most 4 small intervals; the extra four come from time 0, the final time, and T'.

For each interval I_i, let $Q_e(I_i)$ denote the amount of task e processed during I_i, which can be easily computed in polynomial time. For each interval, we will construct an integer schedule that schedules each task as much as the continuous schedule σ^λ does without using too many time steps compared to the interval's length; more precisely, the integer schedule will process at least $\lceil Q_e(I_i) \rceil$ units of task e. We categorize the intervals into three groups. Depending on the category where each interval belongs, we construct an integer schedule differently or give a different upper bound on the length of the integer schedule. At the end, we will concatenate the constructed integer intervals in increasing order of times. In the following, $|I|$ denotes I's length.

The first interval, $I_0 = (0, T']$. Using the same idea we used for handling unit-sized tasks, we find an integer schedule that processes at least $\lfloor Q_e(I_i) \rfloor$, meeting all job deadlines no greater than T'. Note that I_0 has a polynomial length; thus, the desired integer schedule can be computed in polynomial time. Then, we can greedily schedule each task e per unit time such that $Q_e(I_i)$ is not an integer. Note that such a task e hasn't completed by time T', so the task (more precisely, the job to which the task belongs) has deadline at least T'. Therefore, we will be able to charge the extra delay of at most $|U|$ to the corresponding job's deadline directly.

I_i that is not small, for $i \geq 1$. We seek to construct an integer schedule of length $(1 + O(\epsilon))|I_i|$. Towards this end, we do the following. Suppose we divide the interval into $\lceil \frac{|I_i|}{|U|/\epsilon} \rceil$ subintervals of length $|U|/\epsilon$; there can be at most one subinterval of a smaller length and we will handle it later. Next, for each subinterval of length $|U|/\epsilon$, we try to schedule $\lceil \frac{|U|/\epsilon}{|I_i|} Q_e(I_i) \rceil$ units of each task e. Since the length is polynomial in $|U|$, we can find an integer schedule of length $|U|/\epsilon + 1$ that schedules $\lfloor \frac{|U|/\epsilon}{|I_i|} Q_e(I_i) \rfloor$ units of each task e. By scheduling one task per unit time, we can schedule $\lceil \frac{|U|/\epsilon}{|I_i|} Q_e(I_i) \rceil$ units of each task e for $|U|/\epsilon + 1 + |U| \leq (|U|/\epsilon) \cdot (1 + 2\epsilon)$ time steps. Here, our integer schedule's length is at most $(1 + 2\epsilon)$ times the subinterval's length, $|U|/\epsilon$. This integer schedule is repeated $\lfloor \frac{|I_i|}{|U|/\epsilon} \rfloor$ times. We now handle the smaller subinterval of length less than $|U|/\epsilon$. Using a similar argument, we can process more units of each task than the continuous schedule, using at most $|U|/\epsilon + 1 + |U| \leq 2|U|/\epsilon$ time steps. Here we use the fact that I_i has length significantly greater than $|U|$. To see this, suppose we had not added jobs arrival times, deadlines or T' in the process of creating the intervals. Then the intervals preceding I_i have exponentially decreasing lengths by a factor of $(1 + \epsilon)$. Using this observation, we can argue that I_i's length is at least $\epsilon/2$ times I_i's starting time. Since I_i's starting time is greater than T', we have that I_i's length is at least $(\epsilon/2) \cdot T' = (\epsilon/2) \cdot (|U|^2/\epsilon^2) = |U|^2/(2\epsilon)$. So, we can charge the number of time steps spent to handle the smaller subinterval, which is at most $2|U|/\epsilon$, to the length of I_i. From all these arguments, we can construct an integer schedule of length at most $(1 + 6\epsilon)|I_i|$.

I_i that is small, for $i \geq 1$. We seek to construct an integer schedule of length $(1+O(\epsilon))|I_i| + 2|U|/\epsilon$. The whole idea is the same for the intervals that are not small. The only difference is that we cannot charge the extra time steps we spend to handle the smaller subinterval, which is at most $2|U|/\epsilon$, to the length of I_i. Thus, we just use the upper bound on the length of our integer schedule.

As mentioned before, we concatenate the integer schedules originating from $I_0, I_1, \ldots, I_K$ in this order to obtain the final schedule. It now remains to show that each job completes by time $(1+O(\epsilon))C_j^*$. We already showed that our integer schedule completes every job j before its deadline C_j^* if it is smaller than T'. For any other job j, it must be the case that $\bar{C}_j(\lambda)/\lambda$ is greater than T'. Let I_i be the interval including $\bar{C}_j(\lambda)/\lambda$. Due to the way the intervals are constructed, $\bar{C}_j(\lambda)/\lambda$ must be equal to I_i's finish time. Our goal is to show that we complete j not too late compared to I_i's finish time. That is, we want to show that the total length of the integer schedules originating from $I_0, I_1, \ldots, I_i$ is at most $(1+O(\epsilon))\sum_{i' \leq i} |I_{i'}|$. Indeed, the total length is at most,

$$|I_0| + |U| + \sum_{i'=[i]:I_{i'} \text{ is small}} ((1+O(\epsilon))|I_i| + 2|U|/\epsilon) + \sum_{i'=[i]:I_{i'} \text{ is not small}} (1+O(\epsilon))|I_i|$$

$$\leq \sum_{i'=0}^{i}(1+O(\epsilon))|I_{i'}| + |U| + (2|U|/\epsilon)\cdot(8|U|) \leq \sum_{i'=0}^{i}(1+O(\epsilon))|I_{i'}| + O(\epsilon)|I_0|$$

Here, the first inequality follows from the fact that there are at most $8|U|$ small intervals, as argued above. The second inequality is immediate from $|I_0| = T' = |U|^2/\epsilon^2$. Therefore, we have shown that each job completes by time $(1+O(\epsilon))C_j^*$, which establishes that our final schedule's objective is at most $(1+O(\epsilon))$ times the compact LP's optimum. Since we showed the compact LP lower bounds the optimum times $(1+\epsilon)$, we obtain a $2(1+\epsilon)$-approximate schedule for arbitrary $\epsilon > 0$ by scaling ϵ appropriately.

References

1 Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. Sincronia: Near-optimal Network Design for Coflows. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, pages 16–29. ACM, 2018.

2 Saba Ahmadi, Samir Khuller, Manish Purohit, and Sheng Yang. On Scheduling Coflows. In *IPCO*, pages 13–24. Springer, 2017.

3 Nikhil Bansal and Subhash Khot. Inapproximability of hypergraph vertex cover and applications to scheduling problems. In *ICALP*, pages 250–261. Springer, 2010.

4 Zhi-Long Chen and Nicholas G Hall. Supply chain scheduling: Conflict and cooperation in assembly systems. *Operations Research*, 55(6):1072–1089, 2007.

5 Mosharaf Chowdhury and Ion Stoica. Coflow: A networking abstraction for cluster applications. In *ACM Workshop on Hot Topics in Networks*, pages 31–36. ACM, 2012.

6 Mosharaf Chowdhury and Ion Stoica. Efficient coflow scheduling without prior knowledge. In *SIGCOMM*, pages 393–406. ACM, 2015.

7 Mosharaf Chowdhury, Yuan Zhong, and Ion Stoica. Efficient Coflow Scheduling with Varys. In *SIGCOMM*, SIGCOMM '14, pages 443–454, New York, NY, USA, 2014. ACM.

8 William H Cunningham. Testing membership in matroid polyhedra. *Journal of Combinatorial Theory, Series B*, 36(2):161–188, 1984.

9 Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

10 Naveen Garg, Amit Kumar, and Vinayaka Pandit. Order scheduling models: Hardness and algorithms. In *FSTTCS*, pages 96–107. Springer, 2007.

11 Sungjin Im and Manish Purohit. A Tight Approximation for Co-flow Scheduling for Minimizing Total Weighted Completion Time. *CoRR*, abs/1707.04331, 2017. `arXiv:1707.04331`.

12 Sungjin Im, Maxim Sviridenko, and Ruben Van Der Zwaan. Preemptive and non-preemptive generalized min sum set cover. *Mathematical Programming*, 145(1-2):377–401, 2014.

13 Satoru Iwata, Lisa Fleischer, and Satoru Fujishige. A combinatorial strongly polynomial algorithm for minimizing submodular functions. *JACM*, 48(4):761–777, 2001.

14 Hamidreza Jahanjou, Erez Kantor, and Rajmohan Rajaraman. Asymptotically Optimal Approximation Algorithms for Coflow Scheduling. In *SPAA*, pages 45–54. ACM, 2017.

15 Samir Khuller, Jingling Li, Pascal Sturmfels, Kevin Sun, and Prayaag Venkat. Select and permute: An improved online framework for scheduling to minimize weighted completion time. In *LATIN*, pages 669–682. Springer, 2018.

16 Samir Khuller and Manish Purohit. Brief announcement: Improved approximation algorithms for scheduling co-flows. In *SPAA*, pages 239–240. ACM, 2016.

17 Joseph Y-T Leung, Haibing Li, and Michael Pinedo. Scheduling orders for multiple product types to minimize total weighted completion time. *Discrete Applied Mathematics*, 155(8):945–970, 2007.

18 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*, 27(11):3366–3380, 2016.

19 Monaldo Mastrolilli, Maurice Queyranne, Andreas S Schulz, Ola Svensson, and Nelson A Uhan. Minimizing the sum of weighted completion times in a concurrent open shop. *Operations Research Letters*, 38(5):390–395, 2010.

20 Zhen Qiu, Cliff Stein, and Yuan Zhong. Minimizing the Total Weighted Completion Time of Coflows in Datacenter Networks. In *Symposium on Parallel Algorithms and Architectures*, pages 294–303. ACM, 2015.

21 Maurice Queyranne and Andreas S Schulz. Approximation bounds for a general class of precedence constrained parallel machine scheduling problems. *SIAM Journal on Computing*, 35(5):1241–1253, 2006.

22 Maurice Queyranne and Maxim Sviridenko. A (2+ ε)-approximation algorithm for the generalized preemptive open shop problem with minsum objective. *Journal of Algorithms*, 45(2):202–212, 2002.

23 Sushant Sachdeva and Rishi Saket. Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover. In *IEEE Conference on Computational Complexity*, pages 219–229. IEEE, 2013.

24 Alexander Schrijver. A combinatorial algorithm minimizing submodular functions in strongly polynomial time. *Journal of Combinatorial Theory, Series B*, 80(2):346–355, 2000.

25 Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

26 Alexander Schrijver. Matroid Intersection. In *Combinatorial Optimization: Polyhedra and Efficiency*, volume 24, chapter 41. Springer Science & Business Media, 2003.

27 Andreas S Schulz and Martin Skutella. Random-based scheduling new approximations and LP lower bounds. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 119–133. Springer, 1997.

28 Andreas S Schulz and Martin Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal on Discrete Mathematics*, 15(4):450–469, 2002.

29 Mehrnoosh Shafiee and Javad Ghaderi. An Improved Bound for Minimizing the Total Weighted Completion Time of Coflows in Datacenters. *arXiv preprint*, 2017. `arXiv:1704.08357`.

30 Guoqing Wang and TC Edwin Cheng. Customer order scheduling to minimize total weighted completion time. *Omega*, 35(5):623–626, 2007.

31 Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

32 Yangming Zhao, Kai Chen, Wei Bai, Minlan Yu, Chen Tian, Yanhui Geng, Yiming Zhang, Dan Li, and Sheng Wang. RAPIER: Integrating routing and scheduling for coflow-aware data center networks. In *INFOCOM*, pages 424–432. IEEE, 2015.