Coflow Deadline Scheduling via Network-Aware Optimization

Shih-Hao Tseng
School of Electrical and Computer Engineering
Cornell University
Ithaca, New York 14853, U.S.A.
st688@cornell.edu

Ao Tang
School of Electrical and Computer Engineering
Cornell University
Ithaca, New York 14853, U.S.A.
atang@ece.cornell.edu

Abstract—**Several cloud or data-parallel applications involve coflow scheduling, which controls a set of flows under the same semantic meaning with a common goal. Due to such common goal, optimizing each flow using the standard flow scheduling approaches does not necessarily lead to good coflow performance. Therefore, numerous methods and systems are proposed to focus on coflow scheduling problem.**

However, the current coflow scheduling designs are based on simple heuristics or observations. The absence of the optimum makes it hard to adjudge their absolute effectiveness. In this work, we derive the optimal solution to the coflow deadline satisfaction problem (CDS), which maximizes the number of satisfied coflow deadlines, from a mixed integer linear program formulation.

We further show that CDS is not only NP-hard to solve but also intractable to approximate with a fixed approximation ratio (unless P=NP). As such, the use of heuristics is justified. We then develop optimization-based methods to approach the problem offline and online. The proposed methods are simulated and compared against the optimum, along with some state-of-the-art designs, and the results suggest that our methods are much closer to the optimum than the existing ones, especially when we have more room to schedule.

I. Introduction

Coflows are collections of flows that are related semantically with a common objective [1], which can be observed in several applications, including Dryad [2], MapReduce [3], Pig [4], Hive [5], Spark [6], and CIEL [7]. Such collective objectives differentiate coflow scheduling problems from traditional flow scheduling problems, which allocate bandwidth according to per-flow objectives. As pointed out in the literature [1], [8]–[10], flow-based scheduling policies may not suffice to achieve good performance for coflows. Therefore, several methods are proposed to specifically emphasize on coflow scheduling.

Existing coflow scheduling techniques vary in the ways of approaching their objectives: Orchestra employs weighted shuffle scheduling which assigns the rates of flows proportional to their demands [8]; Varys only admits a coflow when it can be completed before its deadline, and the admitted coflows are scheduled according to the bottlenecks they encounter [11]; Baraat processes colfows locally on a first-come-first-serve basis with a variable level of multiplexing [9]; Aalo schedules the coflows without prior knowledge by multi-level priority queues, and the coflow that is least served gets the highest priority [12]; D-CAS is another priority-based scheduling strategy, which relies on the senders and the receivers to negotiate the priority [13]; Stream also leverages the priority queues at each commodity switch to enforce a priority-based scheduling, but unlike Aalo, the priority of a coflow is determined by the receiver [14]; [15] models the sensitivity of completion time by utility functions and schedules the coflows to iteratively maximize the minimum utility; RAPIER is, to the best of our knowledge, the first work that takes network topology into account, and it tries to maneuver both routing and scheduling of coflows to shorten the completion time [10]; OMCoflow manipulates both routing and scheduling as well. Nevertheless, it adopts a randomized rounding-based approximation instead of a deterministic strategy to steer the coflows online [16].

In spite of the methods that have been proposed to schedule coflows, the lack of a comprehensive framework makes all previous attempts fall short in answering the following question: How far away is the proposed heuristic from the optimal schedule? As a result, the coflow scheduling methods tend to compare against each other without knowing the margin to the optimum. We would remark that such question has also been explored for the standard flow scheduling problems (cf. [17]), and our work can be deemed a followup of such optimization-based approach for the coflows. In this paper, we address the coflow scheduling problem in a principled approach by first establishing the optimal solution in an offline setting.

The optimal solution can be established if we have specified the objective. In this work, we focus on maximizing the *coflow deadline satisfaction (CDS)*, i.e., how many coflows can meet their deadlines? The importance of meeting flow deadlines has been illustrated in the literature [18], [19], and their arguments apply to coflows as well. For instance, a coflow originating from an online game match may need to be processed soon, or the players would be upset if the critical timing is missing due to the network delay.

Upon establishing the optimal solution, we can then ask whether the optimal solution can be obtained within reasonable time and resources. It is shown in the paper that CDS

DOI: 10.1109/ALLERTON.2018.8635974

is not only NP-hard to solve, but also hard to approximate. As such, we justify the use of heuristics in the literature. Meanwhile, we derive our heuristics via linear relaxation and careful rounding techniques that cover the offline and the online scenarios.

We organize the paper as follows. Some backgrounds of the coflow scheduling problems are given in the next section. In Section III, we introduce our model and notations to formulate CDS as an optimization problem, followed by the NP-hardness results. Then we propose our relaxation-based methods to approach CDS in both offline and online settings in Section IV. Section V compares the proposed algorithms against some state-of-the-art algorithms through simulations, and we conclude in Section VI.

II. Background

Most network operators handle their traffic according to some flow-level metrics, such as flow-completion time [20] or flow deadline satisfaction [18]. In the presence of coflows, optimizing those metrics may not translate to good coflow performance. Each coflow usually represents a task, and it is deemed finished upon the completion of all its flows. As such, when dealing with coflows, one should focus on coflow-level metrics rather than the flow-level ones. Two common coflow-level metrics are coflow completion time (CCT) and coflow deadline satisfaction (CDS). Under capacity constraints, minimizing CCT aims to serve coflows, as a whole or in average, as fast as possible, while maximizing CDS tries to meet as many deadlines as possible. These two metrics are correlated but different, as demonstrated in [18].

Although most of the work in the literature focuses on CCT minimization, we would argue that maximizing CDS is also important. Following the arguments in [18], [20], a coflow is useful when its flows can be done before their deadlines. For instance, an user-initiated MapReduce job may require its mappers provide results to the reducers prior to some deadline. As such, the reducers can complete their work and respond to the user in time.

In this work, we adopt an optimization-based approach to schedule coflows to meet their deadlines. Optimization depends on the underlying model of the network, and there are three major models: network-oblivious, non-blocking switch, and network-aware. The network-oblivious model does not consider the connectivity within a network. Instead, it focuses on each router and tries to schedule locally. We can also view it as a decentralized approach. The non-blocking switch model, however, takes into account the capacity constraints at the input and the output ports. The assumption behind the model is that sufficient bisection bandwidth is given within the network, and thus the bottlenecks are at the input/output ports. The network-aware model captures the whole network topology along with the link capacity constraints. Such a model can reflect possible in-network congestion, and we leverage it to explore the limit of coflow scheduling and derive our optimization-based methods in the following sections.

TABLE I
Summary of State-of-the-Art Methods

		Network Model		
		Network-Oblivious	Non-Blocking Switch	Network-Aware
Information Availability	Myopic	Baraat [9] Stream [14]	Orchestra [8] Aalo [12]	RAPIER [10]
	Online	D-CAS [13]	Varys [11]	OMCoflow [16] OLPA†
	Offline		max-min utility [15]	LPA† ILPA†

†LPA, ILPA, and OLPA are introduced in Section IV.

On the other hand, scheduling problems can be categorized into different scenarios according to the availability of the coflow information. We elaborate on three scenarios: offline, online, and myopic. Under the offline scenario, the information of all the flows is available for the scheduler before they are scheduled. Such a case happens when the coflows can be precisely forecasted, e.g., scheduled tasks. In general, coflows can show up spontaneously and their per-flow information is available only upon arrival, which is quite common when users issue some searching requests. For those unplanned coflows, we can further group them based on their information availability. Under the online scenario, all flow properties, including the deadline and the size, are revealed when the flow arrives. However, the available information for the myopic case is more limited: no prior information is revealed unless it happens. As a result, the deadline and the size are unknown even upon the arrival of a flow.

We summarize in Table I the state-of-the-art coflow scheduling methods along with their underlying network model and information availability. Most work targets myopic information scenario. Such least information disclosure nature of the myopic scenario confines the analytic conclusions that can be derived. As a result, all those papers propose various heuristics to schedule coflows. In this work, we aim to schedule coflows in a principled way that would be connected with the optimal solution to CDS. We argue that the coflows can be intentionally scheduled to satisfy their deadlines only when the deadlines are known before they occur. Otherwise, the scheduler might try to approach some proxy metric, such as CCT minimization, in the hope of reaching a good CDS solution, which is not necessarily the case [18]. Therefore, we develop our setup and methods under offline and online environments.

III. Formulation and Analysis

In this section, we introduce our model to formulate the coflow deadline satisfaction (CDS) problem, which maximizes the number of satisfied deadlines by scheduling each flow through its predetermined path. CDS is then shown NP-hard, which justifies the use of approximation heuristics that will be proposed in the following section.

A. Notations

The network is modeled as a directed graph $G = (V, E)$ where V is the set of nodes and E is the set of directed edges. Each edge $e \in E$ is associated with a capacity $c_e \geq 0$.

N coflows are generated to go through the network. Each coflow F^n, $n \in N$, is a collection of correlated flows f_j indexed by $j \in J^n$, and each flow f_j is defined by its lifespan τ_j, its predetermined acyclic path p_j, and its size s_j. The lifespan τ_j is a time interval $[a_j, d_j]$ where a_j is the arrival time and d_j is the deadline of the flow f_j. The flow is routed through a determined path p_j between some nodes in V. We say an edge $e \in p_j$ if the path p_j goes through the edge e.

Although our model allows τ_j to vary among the flows, in practice, each coflow represents a task and hence the lifespan is defined on a per-coflow instead of a per-flow basis. Therefore, we consider $\tau_j = \tau^n = [a^n, d^n]$ for all $j \in J^n$ in the following context, where a^n and d^n are the arrival time and the deadline of the coflow F^n.

Given the time horizon $T > 0$, we consider a finite horizon scheduling problem, i.e., $0 \leq a_j < d_j \leq T$ for all $j \in J = \bigcup\limits_{n \in N} J^n$.

B. Coflow Deadline Satisfaction (CDS) Problem

A CDS aims to maximize the number of satisfied coflows within a given horizon T. Within the horizon, CDS finds a schedule for each flow to go through its predetermined path. While such a problem can be written as a continuous time control problem, following Proposition 1 in [17], we can also write CDS in the form of a mixed integer linear program (MILP). To do so, we partition $[0, T]$ into M disjoint subintervals Δ_m such that their endpoints are either a_j or d_j for some $j \in J$ if not 0 or T. We denote by $|\Delta_m|$ the length of the interval Δ_m.

Let $x_j(\Delta_m) \geq 0$ denote the sending rate of flow f_j through its path p_j during $\Delta_m \subseteq \tau_j$. Also, indicator $z^n \in \{0, 1\}$ corresponds to whether the coflow F^n can be satisfied during $[0, T]$. F^n is deemed satisfied if all flows f_j, $j \in J^n$, can transmit data s_j within its lifespan τ_j. $z^n = 1$ if F^n is satisfied, and $z^n = 0$ otherwise.

Using the above notations, we formulate CDS as follows

$$
\begin{aligned}
\max \quad & \sum_{n \in N} z^n \\
\text{s.t.} \quad & \sum_{\Delta_m \subseteq \tau^n} x_j(\Delta_m)\,|\Delta_m| = s_j z^n \\
& \qquad\qquad\qquad\qquad \forall n \in N, j \in J^n && (1) \\
& z^n \in \{0, 1\} \qquad\qquad \forall n \in N && (2) \\
& \sum_{j \in J : e \in p_j} x_j(\Delta_m) \leq c_e \qquad \forall e \in E, \Delta_m \subseteq [0, T] && (3) \\
& x_j(\Delta_m) \geq 0 \qquad\qquad \forall j \in J, \Delta_m \subseteq \tau_j && (4) \\
& x_j(\Delta_m) = 0 \qquad\qquad \forall j \in J, \Delta_m \nsubseteq \tau_j && (5)
\end{aligned}
$$

Condition (1) is the demand constraint for each flow: each flow f_j sends s_j when it is satisfied ($z^n = 1$). z^n is introduced in condition (2). Condition (3) is the capacity constraint. Conditions (4) and (5) ensure that $x_j(\Delta_m)$ can only be non-zero during its lifespan τ_j.

Solving the MILP above gives the optimal schedule, which allows us to compare the performance of different scheduling algorithms.

C. NP-hardness of CDS

An important question regarding CDS is whether we can solve CDS efficiently. Proposition 1 shows that CDS is not only hard to solve but also hard to be approximated.

Proposition 1. *CDS is NP-hard and there exists no constant factor polynomial-time approximation algorithm for CDS unless P=NP.*

Proof. We prove the proposition by reducing the offline rate control problem in [17] (problem (4), abbreviated as ORC in the following context) to a CDS. Given an ORC, we construct the corresponding CDS by mapping each flow in ORC to a coflow F^n with singleton J^n in CDS. As such, ORC can be reduced to CDS in polynomial time, which implies the proposition. □

IV. Proposed Algorithms

Although Proposition 1 obviates the attempts of searching for an approximation algorithm with fixed ratio performance guarantee, a good approximation algorithm is still essential for the system operators. Below we propose some optimization-based approximation algorithms, which include both offline and online versions. Through simulations in Section V, we demonstrate that our algorithms outperform existing coflow scheduling heuristics.

A. Linear Programming Approximation (LPA)

The most straightforward scheduling method is the linear programming approximation (LPA), which relaxes the integer constraints (2) in CDS and solves the resulting linear program (LP) as below.

$$
\begin{aligned}
\max \quad & \sum_{n \in N} z^n \\
\text{s.t.} \quad & 0 \leq z^n \leq 1 \qquad\qquad \forall n \in N \\
& \text{Conditions (1), (3), (4), and (5).}
\end{aligned}
$$

Upon obtaining the result of the LP, LPA schedules the flows according to $x_j(\Delta_m)$, and the coflows with $z^n = 1$ will be satisfied.

B. Iterative Linear Programming Approximation (ILPA)

LPA satisfies the coflows corresponding to $z^n = 1$. For those coflows with $z^n < 1$, LPA also allocates bandwidth to them, which is a waste of bandwidth. To prevent the drawback, we can omit a coflow whenever it is no longer possible to be satisfied. As such, the other satisfiable coflows have more bandwidth to share.

Intuitively, with more bandwidth, we can find new schedules that satisfy at least the same number of coflows, and hopefully, LPA will discover those schedules after omitting the unsatisfiable coflows. That leads to the design of interactive LPA (ILPA), which iterates through the subintervals Δ_m, applies LPA on the rest of the horizon to obtain the sending rate of the current subinterval $x_j(\Delta_m)$, and drops the unsatisfiable coflows.

However, LPA does not necessarily find a solution satisfying more coflows, i.e., with more $z^n = 1$. Instead, it only reaches a higher objective value, which may consist of more fractional z^n. Therefore, we should adopt a new schedule only when it could satisfy at least as many coflows as the old schedule. Also, usually known as the *work conservation criterion*, we would prefer the schedule that can be finished earlier if both old and new schedules satisfy the same number of coflows. Incorporating the two principles into the design, we propose ILPA in Algorithm 1.

Algorithm 1: Iterative Linear Programming Approximation (ILPA)

```
for $\Delta_m$ from earliest to the last do
    Remove the coflows that cannot be satisfied anymore.
    Apply LPA to solve for new
    $x_j(\Delta_m), x_j(\Delta_{m+1}), \ldots$.
    Adopt the new LPA schedule if
      1) more coflows can be satisfied, or
      2) the same number of coflows can be satisfied
         strictly earlier.
end for
```

C. Online Linear Programming Approximation (OLPA)

ILPA is an offline algorithm – it requires the coflow information before starting scheduling. Offline methods work for scheduled tasks, while coflows can also be spawned spontaneously in practice. Such a scenario leads to an online environment, and we can easily handle online scheduling using ILPA as a building block.

The idea is to refresh the schedule whenever a flow arrives, expires, or finishes. Those events account for a_j and d_j in the offline setting. Similar to the iterative procedure in ILPA, we propose online LPA (OLPA) in Algorithm 2 that reevaluates the schedule when those pivot events occur.

Algorithm 2: Online Linear Programming Approximation (OLPA)

```
for whenever a flow arrives, expires, or finishes do
    Apply ILPA to schedule the satisfiable coflows.
    Adopt the new ILPA schedule if
      1) more coflows can be satisfied, or
      2) the same number of coflows can be satisfied
         strictly earlier.
end for
```

V. Simulation

We evaluate our proposed algorithms along with some existing coflow schedulers: Varys [11], Aalo [12], and RAPIER [10][1]. By comparing those methods against the optimal solution to CDS, we would get a sense of how close the methods are to the optimum.

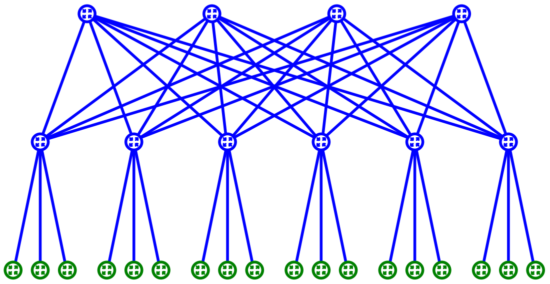


Fig. 1. We conduct simulations based on the fat-tree topology. Each link has capacity 10 Gbps, and coflows are formed from randomly generated leaf-to-leaf flows.

The evaluations are done through simulations. We conduct simulations on ns-3. Within the horizon $T = 100$ ms, we generate coflows according to a Poisson process with different means of inter-arrival time. Each coflow is a MapReduce job consisting of 1 to 3 mappers and reducers, which are selected from leaf nodes of the fat-tree network in Fig. 1. Each reducer requires a data size uniformly distributed over $[1, 100]$ MB from every mapper through one of the shortest paths. Such a path is predetermined for each mapper/reducer pair.

The lifespan of each flow f_j is set according to the tightness parameter q, which is defined as

$$\tau_j = q \times \text{minimum possible lifespan of the flow},$$

where the minimum possible lifespan of the flow is estimated under the assumption that the flow is scheduled through an empty network. Therefore, $q = 1$ means that the flow can only be satisfied if it is assigned full available bandwidth. Essentially, a larger q creates more room for scheduling.

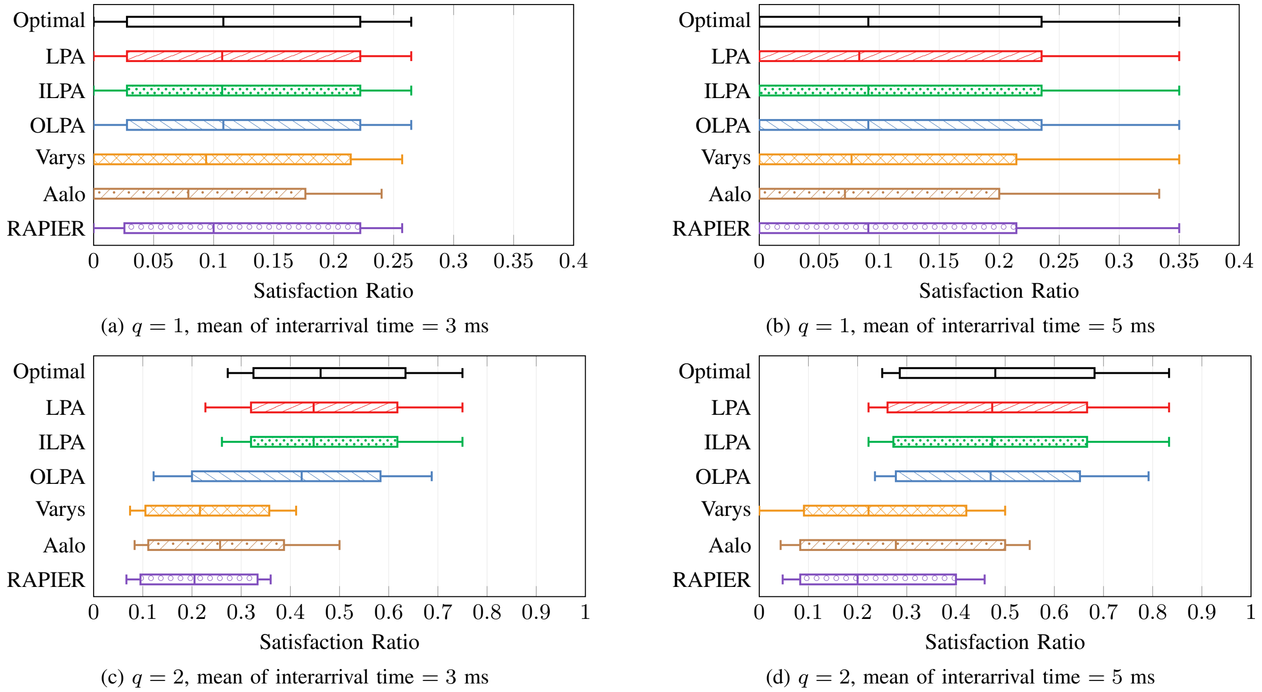
We express the simulation results in *satisfaction ratio*. The satisfaction ratio of a schedule is given by

$$\text{satisfaction ratio} = \frac{\text{number of satisfied coflows}}{\text{total number of coflows}}.$$

It is straightforward to see that a larger satisfaction ratio implies more satisfied coflows.

The simulation results are in Fig. 2. We consider two different tightness parameters $q = 1$ and $q = 2$ under two different inter-arrival time 3 ms and 5 ms. As expected, smaller tightness parameter leads to smaller satisfaction ratio. On the other hand, shorter inter-arrival time implies a more congested network. As a result, fewer coflows can be satisfied and the satisfaction ratio is smaller. Within all the simulation scenarios, ILPA and OLPA outperform the existing scheduling methods as they have more deadline information and better network perception. RAPIER works better than Varys and Aalo when q is small, but Aalo can satisfy more coflows when there is more room for scheduling ($q = 2$).

[1] OMCoflow [16] is not chosen as it involves randomization.



(a) $q = 1$, mean of interarrival time $= 3$ ms

(b) $q = 1$, mean of interarrival time $= 5$ ms

(c) $q = 2$, mean of interarrival time $= 3$ ms

(d) $q = 2$, mean of interarrival time $= 5$ ms

Fig. 2. The 1^{st}-5^{th}-50^{th}-95^{th}-99^{th} percentiles under various q and mean of inter-arrival time. Our proposed methods (LPA, ILPA, and OLPA) outperform the existing methods Varys, Aalo, and RAPIER, and their performance is close to the optimal solution.

VI. Conclusion

This paper examines the coflow deadline satisfaction (CDS) problem, which schedules the coflows through predetermined paths to maximize the number of satisfied coflow deadlines. We view CDS from the angle of optimization and show that CDS is NP-hard. Moreover, approximating CDS with a fixed approximation ratio is also intractable (unless P=NP), which justifies the application of heuristics in the literature. Through linear relaxation and rounding techniques, we propose LPA, ILPA, and OLPA that can schedule coflows under offline and online scenarios. Our simulations demonstrate that the proposed algorithms perform better than the existing methods and close to the optimum. Therefore, our results can serve as the benchmarks for the coflow deadline scheduling problems.

References

[1] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *HotNets*. ACM, 2012, pp. 31–36.

[2] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," in *ACM SIGOPS Operating Systems Review*, vol. 41, no. 3. ACM, 2007, pp. 59–72.

[3] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[4] C. Olston, B. Reed, U. Srivastava, R. Kumar, and A. Tomkins, "Pig latin: A not-so-foreign language for data processing," in *Proc. ACM SIGMOD*. ACM, 2008, pp. 1099–1110.

[5] A. Thusoo, J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, and R. Murthy, "Hive: A warehousing solution over a map-reduce framework," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1626–1629, 2009.

[6] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," *HotCloud*, vol. 10, no. 10-10, p. 95, 2010.

[7] D. G. Murray, M. Schwarzkopf, C. Smowton, S. Smith, A. Madhavapeddy, and S. Hand, "CIEL: A universal execution engine for distributed data-flow computing," in *Proc. 8th ACM/USENIX Symposium on Networked Systems Design and Implementation*, 2011, pp. 113–126.

[8] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM CCR*, vol. 41, no. 4, pp. 98–109, 2011.

[9] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, "Decentralized task-aware scheduling for data center networks," *ACM SIGCOMM CCR*, vol. 44, no. 4, pp. 431–442, 2014.

[10] 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 *Proc. IEEE INFOCOM*. IEEE, 2015, pp. 424–432.

[11] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with Varys," *ACM SIGCOMM CCR*, vol. 44, no. 4, pp. 443–454, 2014.

[12] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM CCR*, vol. 45, no. 4, pp. 393–406, 2015.

[13] S. Luo, H. Yu, Y. Zhao, B. Wu, S. Wang *et al.*, "Minimizing average coflow completion time with decentralized scheduling," in *Proc. IEEE ICC*. IEEE, 2015, pp. 307–312.

[14] H. Susanto, H. Jin, and K. Chen, "Stream: Decentralized opportunistic inter-coflow scheduling for datacenter networks," in *IEEE ICNP*. IEEE, 2016, pp. 1–10.

[15] L. Chen, W. Cui, B. Li, and B. Li, "Optimizing coflow completion times with utility max-min fairness," in *Proc. IEEE INFOCOM*. IEEE, 2016, pp. 1–9.

[16] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau, "Efficient online coflow routing and scheduling," in *Proc. ACM MOBICHOC*. ACM, 2016, pp. 161–170.

[17] A. Gushchin, S.-H. Tseng, and A. Tang, "Optimization-based network flow deadline scheduling," in *Proc. IEEE ICNP*, nov 2016.

[18] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *ACM SIGCOMM CCR*, vol. 41, no. 4, pp. 50–61, 2011.

[19] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," *ACM SIGCOMM CCR*, vol. 42, no. 4, pp. 115–126, 2012.

[20] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM CCR*, vol. 36, no. 1, pp. 59–62, 2006.