Scheduling Coflows With Dependency Graph

Mehrnoosh Shafiee, *Student Member, IEEE*, and Javad Ghaderi, *Senior Member, IEEE*

***Abstract*— Applications in data-parallel computing typically consist of multiple stages. In each stage, a set of intermediate parallel data flows (*Coflow*) is produced and transferred between servers to enable starting of next stage. While there has been much research on scheduling isolated coflows, the dependency between coflows in multi-stage jobs has been largely ignored. In this paper, we consider scheduling coflows of multi-stage jobs represented by general *DAG*s (Directed Acyclic Graphs) in a shared data center network, so as to minimize the total weighted completion time of jobs. This problem is significantly more challenging than the traditional coflow scheduling, as scheduling even a single multi-stage job to minimize its completion time is shown to be NP-hard. In this paper, we propose a polynomial-time algorithm with approximation ratio of $O(\mu \log(m)/\log(\log(m)))$, where μ is the maximum number of coflows in a job and m is the number of servers. For the special case that the jobs' underlying dependency graphs are *rooted trees*, we modify the algorithm and improve its approximation ratio. To verify the performance of our algorithms, we present simulation results using real traffic traces that show up to 53% improvement over the prior approach. We conclude the paper by providing a result concerning an optimality gap for scheduling coflows with general DAGs.**

***Index Terms*— Multi-stage job, coflow, scheduling algorithms, approximation algorithms, data centers.**

I. Introduction

MODERN parallel computing platforms (e.g. Hadoop [1], Spark [2], Dryad [3]) have enabled processing of big data sets in data centers. Processing is typically done through multiple computation and communication stages. While a computation stage involves local operations in servers, a communication stage involves data transfer among the servers in the data center network to enable the next computation stage. Such intermediate communication stages can have a significant impact on the application latency [4]. *Coflow* is an abstraction that has been proposed to model such communication patterns [4]. Formally, a coflow is defined as a collection of flows whose completion time is determined by the last flow in the collection. For jobs with a single communication stage, minimizing the average completion times of coflows results in the job's latency improvement. However, for multi-stage jobs, minimizing the average coflow completion time is not be the right metric and might even lead to a worse performance, as it ignores the dependencies between coflows in a job [5]–[7].

There are two types of dependency between coflows of a multi-stage job: *Starts-After* and *Finishes-Before* [7]. A Starts-After constraint between two coflows represents an explicit barrier that the second coflow can start only after the first coflow has been completed [8]. A Finishes-Before constraint is common when pipelining is used between successive stages [3], where two dependent coflows can coexist but the second coflow cannot finish until the first coflow finishes. In this paper we focus on scheduling coflows of multi-stage jobs with Starts-After dependency, however, our techniques and results can be easily extended to the other case. Each job is represented by a DAG (*Directed Acyclic Graph*) among its coflows that capture the (Starts-After) dependencies among the coflows. As in [5], [6], [9]–[11], the data center network is modeled as an $m \times m$ switch where m is the number of servers (see Section II for the formal job and data center network model). As an illustration, Figure 1 shows one multi-stage job in a 2×2 switch.

In this paper, we focus on the algorithmic task of scheduling flows of different coflows in the data center network. We consider two optimization problems:

1. Given a set of jobs, minimize the total time to schedule flows of all the coflows (makespan), while respecting the dependencies among the coflows as well as the capacity constraints imposed by the data center network.
2. Given a set of weights, one for each job, minimize the total weighted completion time of jobs, where the completion time of a job is determined by the completion of the last coflow in its DAG. The weights can capture priorities for different jobs.

We state the results as approximation ratios in terms of m (the number of servers), and μ (the maximum number of coflows in a job).

A. Related Work

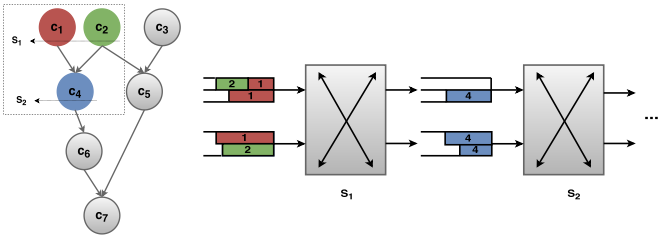
The problem considered in this paper can be thought of as a generalization of coflow scheduling that has been widely studied from both theory and system perspectives [7], [9]–[18]. In these papers, different models and scheduling scenarios are considered. For instance, in [9], [11]–[13], [19], preemption of flows is allowed, while [17], [18] focus on the non-preemptive scenario. Further, scheduling coflows in the case that flow sizes are not specified or only their destinations are known is studied in [7], [18]. However, there are only a few works [5]–[7],

Manuscript received December 18, 2020; revised June 11, 2021 and September 25, 2021; accepted September 25, 2021; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor V. Aggarwal. Date of publication October 11, 2021; date of current version February 17, 2022. This work was supported by NSF under Grant CNS-1717867 and Grant CNS-1652115. *(Corresponding author: Mehrnoosh Shafiee.)*

The authors are with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: s.mehrnoosh@columbia.edu; jghaderi@ee.columbia.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNET.2021.3116133, provided by the authors.

Digital Object Identifier 10.1109/TNET.2021.3116133



(a) A multi-stage job with 7 coflows. (b) Flows of coflows 1, 2, and 4 and their dependencies in a 2×2 switch.

Fig. 1. A multi-stage job in a 2×2 switch. Part of the DAG (in the dashed box) consisting of coflows 1, 2, and 4 is shown in the switch. Coflows 1 and 2 can share the network resources at the same time because they are independent (see S_1). Once all their flows are transmitted, flows of coflow 4 will be ready to be transmitted (S_2 after S_1).

[20] that consider the multi-stage generalization, with only one algorithm with theoretical performance guarantee [5], [6]. Among the heuristics, Aalo [7] mainly focused on coflow scheduling problem and only provides a brief heuristic to incorporate the multi-stage case. The paper [20] proposed a two-level scheduling method based on the most-bottleneck-first heuristic to find the jobs to schedule at each round, and a weighted fair scheduling scheme for intra-job coflow scheduling.

The recent papers [5], [6] are the most relevant to our work. They consider the problem of scheduling multi-stage job (with Starts-After dependency) to minimize the total weighted job completion times and provide an LP (Linear Program)-based algorithm with $O(m)$ approximation ratio. This algorithm utilizes the technique based on ordering variables, that was also used for coflow scheduling. Their analysis for this algorithm relies on aggregating the load on all the m servers which results in the loss of $O(m)$ in the approximation ratio. *In this paper, we exponentially improve this result by proposing an algorithm that achieves an approximation ratio of* $O(\mu g(m))$, where μ is the maximum number of coflows in a job, and $g(m) = \log(m)/\log(\log(m))$. Moreover, in the case that the multi-stage job's dependency graph is a rooted tree, we propose an algorithm that achieves an approximation ratio of $O(\sqrt{\mu} g(m) h(m,\mu))$, where $h(m,\mu) = \log(m\mu)/(\log(\log(m\mu))$. We would like to emphasize that the $O(m)$ approximation in [5], [6] will not improve if the graph is a rooted tree rather than a general DAG. Note that in practice, the number of coflows in a job is some constant which is much smaller than the number of servers in real-world data centers with hundreds of thousands of servers, i.e., $\mu \ll m$. Also, unlike the $O(m)$ algorithm [5], [6], both of our algorithms are completely combinatorial and do not need to solve a linear program explicitly, hence reducing the complexity. A key reason behind the performance improvement in our algorithms is that they utilize the network resources more efficiently by interleaving schedules of coflows of different jobs, unlike the $O(m)$ algorithm [5], [6] that schedules coflows one at a time.

Since we represent the dependencies between coflows of a multi-stage job with a *Directed Acyclic Graph* (DAG), DAG scheduling problem is a related line of work. In traditional DAG scheduling, each node represents a task with some processing time and an edge between two nodes indicates the tasks' dependency. There has been extensive results on DAG scheduling problem (DAG-SP) where the goal is to assign tasks to machines in order to minimize the DAG's completion time [21]–[26].

There are also results on DAG-shop scheduling problem (DAG-SSP) [27]–[30] in which, unlike the DAG-SP, the machine on which each task has to be processed is fixed and no two tasks of the same job can be processed simultaneously. Our problem of scheduling coflow DAGs is different from the aforementioned problems in several aspects: First, a node in our DAG represent a coflow which itself is a collection of data flows, each with a given pair of source-destination servers. Such couplings are fundamentally different from DAG-SP. Second, flows of the same coflow and different unrelated coflows can be scheduled at the same time, which is different from DAG-SSP. Hence, algorithms from DAG-SP and DAG-SSP cannot be applied to our problem.

B. Main Contributions

Define $g(m) := \log(m)/\log(\log(m))$, and $h(m,\mu) := \log(m\mu)/\log(\log(m\mu))$. Our main results in this paper can be summarized as follows.

1. We first prove that even scheduling a multi-stage job to minimize its completion time (makespan) is NP-hard. We then propose an algorithm for minimizing the time to schedule a given set of multi-stage jobs. Our algorithm runs in polynomial time and constructs a schedule in which the makespan is within $O(\mu g(m))$ of the optimal solution for the case that jobs have general DAGs, and $O(\sqrt{\mu} g(m) h(m,\mu))$ when each job is represented as a rooted tree. The algorithms rely on random delaying and merging the greedy schedules of jobs, followed by enforcing the bandwidth constraints.
2. We propose two approximation algorithms for minimizing the total weighted completion time of a given set of multi-stage jobs. For general DAGs, the approximation ratio of our algorithm is $O(\mu g(m))$. For the case of rooted trees, the ratio is improved to $O(\sqrt{\mu} g(m) h(m,\mu))$. Our algorithms are completely combinatorial and do not rely on an explicit solution of a linear program (LP), thus reducing the complexity dramatically. Our approximation algorithms are significant improvements over the LP-based $O(m)$-algorithm of [5], [6].
3. To demonstrate the gains in practice, we present extensive simulation results using real traffic traces. The results indicate that our algorithms outperform the $O(m)$-algorithm [5], [6] by up to 36% and 53% for general DAGs and rooted trees, respectively, in the same settings.
4. We illustrate the existence of instances for which the optimal makespan for a single job with a general DAG is $\Omega(\sqrt{\mu})$ factor larger than two lower bounds for the problem.

While we utilize classical techniques, such as "random delay" from job shop scheduling [28], [29] and LP relaxation

from single-stage coflow scheduling [9], [11], [13]–[15], our setting of scheduling coflow DAGs is considerably more complex, and we need to add new ideas to such classical techniques and carefully adjust them to our setting to prove that our algorithms provide good solutions in polynomial time (see Section IX for details.). Further, our result concerning the optimality gap is new.

C. Organization

We start with the network and job model and the problem statement in Section II. We then present some preliminaries and a few definitions in Section III that are used in the rest of the paper. Section IV is devoted to the problem of minimizing the total time to schedule a set of general DAG jobs, while Section V considers the same problem in the case that each job's DAG is a rooted tree. Using these results, in Section VI we present our approximation algorithm for minimizing the total weighted completion time of multi-stage jobs with release times. Empirical evaluation results are presented in Section VII. We further provide an insight concerning the optimality gap of our approximation results in Section VIII. The detailed proofs of the theorems is provided in Section IX. Finally, we conclude the paper in Section X.

II. Model and Problem Statement

Network Model: We consider a cluster of m servers, denoted by the set $\mathcal{M}$. Each server has 2 communication links, one input and one output link with capacity (bandwidth) constraints. For simplicity, we assume all links have equal capacity and without loss of generality, we assume that all the link capacities are normalized to one. Similar to the models in [6], [9]–[11], we abstract out the data center network as one giant non-blocking switch. Each server in the set $\mathcal{M}$ is represented by one sender server and one receiver server. Therefore, we have an $m \times m$ switch, where the m sender (source) servers on one side, denoted by set $\mathcal{M}_S$, connected to m receiver (destination) servers on the other side, denoted by set $\mathcal{M}_R$.

Job Model: There is a collection of n multi-stage jobs, denoted by the set $\mathcal{N}$. Each job $j \in \mathcal{N}$ consists of μ_j coflows that need to be processed in a given (partial) order. Each coflow c of job j is a collection of flows denoted by an $m \times m$ demand matrix $\mathcal{D}^{(cj)}$. Every flow is a quadruple (s, r, c, j), where $s \in \mathcal{M}_S$ is its source server, $r \in \mathcal{M}_R$ is its destination server, and c and j are the coflow and the job to which it belongs. The size of flow (s, r, c, j), denoted by d_{sr}^{cj}, is the (s, r)-th element of the matrix $\mathcal{D}^{(cj)}$. For two coflows $c_1, c_2 \in j$, we say coflow c_1 *precedes* coflow c_2, and denote it by $c_1 \prec c_2$, if all flows of $\mathcal{D}^{(c_1 j)}$ should complete before we can start scheduling any flow of $\mathcal{D}^{(c_2 j)}$ (i.e., Starts-After dependency). We use a DAG G_j to represent the dependency (partial ordering) among the coflows in job j, i.e., nodes in G_j represent the coflows of job j and directed edges represent the dependency (precedence constraint) between them. We use $\mu = \max_{j \in \mathcal{N}} \mu_j$ to denote the maximum number of coflows in any job. Figure 1 illustrates a multi-stage job in a 2×2 switch network.

Scheduling Constraints: Without loss of generality, we assume file sizes of flows are integers and the smallest file size is at least one which is referred to as a *packet*. Scheduling decisions are restricted to such data units (packets), i.e., each sender server can send at most one packet in every time unit (time slot) and each receiver server can receive at most one packet in every time slot, and the feasible schedule at any time slot has to form a *matching* of the switch's bipartite graph. Note that the links' capacity constraints are captured by matching constraints, similarly to models in [5], [6], [9], [11]. Further, in a valid schedule, all the precedence constraints in any DAG G_i have to be respected.

Optimization Objective: A job is called completed only when all of its coflows finish their processing. Define C_{cj} to be the completion time of coflow (c, j). Then, the completion time of job j, denoted by C_j, is equal to completion time of its last coflow, i.e., $C_j = \max_{c \in j} C_{cj}$. The total time that it takes to complete all the jobs in the set $\mathcal{N}$ is called *makespan* which we denote it by $C^{(\mathcal{N})}$. Note that by definition $C^{(\mathcal{N})} = \max_{j \in \mathcal{N}} C_j$. Given a set of jobs, our first objective is to minimize $C^{(\mathcal{N})}$. Next, given positive weights w_j, $j \in \mathcal{N}$, we consider the problem of minimizing *the sum of weighted job completion times* defined by $\sum_{j \in \mathcal{N}} w_j C_j$. The weights can capture different priority for different jobs. In the special case that all the weights are equal, the problem is equivalent to minimizing the average job completion time.

III. Definitions and Preliminaries

We first present a few definitions and preliminaries regarding complexity of the scheduling problem, and how to optimally schedule a single job whose graph is a path using known results.

A. Definitions

Definition 1 (Server Load and Effective Size of a Coflow): Suppose a coflow $\mathcal{D} = \left(d_{sr}\right)_{s,r=1}^{m}$ is given. Define

$$d_s = \sum_{r \in \mathcal{M}_R} d_{sr}; \quad d_r = \sum_{s \in \mathcal{M}_S} d_{sr}, \tag{1}$$

then d_s (d_r) is called the load that needs to be sent from sender server s (received at receiver server r) for coflow $\mathcal{D}$. Further, the effective size of the coflow is defined as

$$D = \max\{\max_{s \in \mathcal{M}_S} d_s, \max_{r \in \mathcal{M}_R} d_r\}. \tag{2}$$

Thus D is the maximum load that needs to be sent or received by a server for the coflow. Note that, due to normalized capacity constraints on links, *we need at least D time slots to process all its flows.*

Definition 2 (Aggregate Size of a Set of Coflows): Given a set of coflows, consider an aggregate coflow $\mathcal{D} = \sum_c \mathcal{D}^c$ for c's in the set. Then, aggregate size of the set is defined as the effective size of $\mathcal{D}$ based on Definition 1. Similarly, aggregate size of job j is defined as the aggregate size of its set of coflows and is denoted by Δ_j.

Definition 3: (Size of a Directed Path and Critical Path in a Job) Given a job $j \in \mathcal{N}$ and its DAG G_j, size of a directed path p in G_j is defined as $T_{pj} = \sum_{c \in p} D^{(cj)}$, where $D^{(cj)}$ is

TABLE I
TABLE OF NOTATIONS

Definition	Symbol
Set of servers, Number of servers	$\mathcal{M}$, m
Set of source (sender) servers	$\mathcal{M}_s$
Set of destination (receiver) servers	$\mathcal{M}_r$
Set of jobs, Number of jobs	$\mathcal{N}$, n
Flow of coflow c in job j from sender s to receiver r	(s, r, c, j)
Size of flow (s, r, c, j)	d_{sr}^{cj}
Demand matrix of coflow c of job j	$\mathcal{D}^{(cj)}$
Number of coflows of job j	μ_j
Maximum number of coflows of all jobs	μ
DAG representation of job j	G_j
Coflow c_1 precedes coflow c_2	$c_1 \prec c_2$
Completion time of coflow c of job j	C_{cj}
Completion time of job j	C_j
Weight of job j	w_j
Makespan for completing all jobs of set $\mathcal{N}$	$C^{\mathcal{N}}$
Effective size of coflow c of job j	$D^{(cj)}$
Aggregate size of job j	Δ_j
Size of directed path p in job j	T_{pj}
Size of critical path of job j	T_j
Root of job j whose graph (G_j) is a rooted tree	R_j
Height of job j	H_j
Set of coflows of job j with no in-edge	S_0^j
Set of coflows of job j whose longest path to some coflow of set S_0^j has length i	S_i^j

the effective size of coflow c of job j, and $c \in p$ denotes that coflow c appears in path p.

Critical path of job j is a directed path that has the maximum size among all the directed paths in G_j. We use $T_j = \max_p T_{pj}$ to denote its size.

Definition 4 (A Path Job): We say a job is a path job if its corresponding dependency graph is a path, i.e., there is a total ordering of its coflows according to which they should get scheduled.

Definition 5 (A Rooted-Tree Job): We say a job is a rooted-tree job if its corresponding dependency graph is a rooted tree, i.e., it is a tree and there is a unique node called the root *and either all the directed edges point away from this node (fan-out tree) or point toward this node (fan-in tree). For each rooted-tree job G_j, we use R_j to denote its root.*

Definition 6 (Height and Coflow Sets for a Job): Given a job $j \in \mathcal{N}$ and its graph G_j, we define H_j to be the height of G_j, i.e., the length of the longest path in G_j (in terms of number of coflows). Further, we define S_0^j to denote the set of coflows with no in-edge. Similarly, define S_i^j, $i = 1, \ldots, H_j - 1$ to denote the set of coflows whose longest path to some coflow of set S_0^j has length i. Note that coflows in G_j are partitioned by S_i^js, i.e., $\cup_{i=0}^{H_j-1} S_i^j = G_j$ and $S_i^j \cap S_{i'}^j = \varnothing$, for $i, i' = 0, \ldots, H_j - 1$, $i \neq i'$. We refer to S_i^js as coflow sets of job j.

Table I summarizes the model parameters and the notations defined in this section.

B. Complexity of Minimizing Makespan

Scheduling a multi-stage job to minimize its completion time (makespan) is NP-hard. To show this, we consider a single multi-stage job whose DAG is a rooted tree. The proof is through a reduction from preemptive makespan minimization for Flow Shop Problem (FSP) which is known to be NP-complete [31]–[33]. This is in contrast to traditional coflow scheduling where a single coflow can be scheduled optimally as we see in Section III-C. This also shows that the known complexity results for preemptive FSP holds for single multi-stage job scheduling. For FSP, there is no algorithm with an approximation ratio less than $5/4$, unless P = NP [34].

Theorem 1: Given a single multi-stage job represented by a rooted tree, scheduling its coflows to minimize makespan over an $m \times m$ switch is NP-hard.

Proof: We prove the theorem using a reduction from preemptive makespan minimization for Flow Shop Problem (FSP). In FSP, there is a set of n jobs each of which consists of m tasks that need to be processed *in a given order* on m machines. Task i of job j must be scheduled on machine i for p_{ij} amount of time (all the jobs require the same order on their tasks.). Preemptive makespan minimization of FSP is known to be NP-complete [31], [33].

Consider an instance I of FSP with n jobs and m machines. We convert the makespan minimization for I to makespan minimization of an instance I' of a single multi-coflow job with a rooted tree topology. The instance I' consists of m source and m destination servers and $n \times m + 1$ coflows where each has a single flow. Further, the corresponding dependency graph of I' is a tree with a root node and n branches. The root node is a dummy coflow which has one flow of size one from source server 2 (or any other source server) to destination server 1. Each of the n branches of the tree represents a job in I and consists of m coflows. The nodes in the l-th level of the tree, $l = 1, \ldots, m-1$ (the level of root node is zero) represent coflows that each has a single flow from source server l to destination server $l+1$ with sizes p_{lj}, $j = 1, \ldots, n$. Similarly, the nodes at level m are coflows with a single flow from source node m to destination server 1 with sizes p_{mj}, $j = 1, \ldots, n$.

If one can find the optimal makespan for the instance I' of a single multi-coflow job, the solution gives an optimal scheduling for the instance I by ignoring the first time unit that is used to schedule the dummy coflow in I'. Therefore, the theorem is proved. □

Using Theorem 1 it is easy to see that minimizing makespan for multiple jobs and total weighted completion time of jobs are NP-hard.

C. Optimal Makespan for a Path Job

In this section, we first show how one can schedule a single coflow optimally and in a polynomial time using the previous results. As a result of Birkhoff-von Neumann Theorem [35], given a coflow $\mathcal{D} = \left(d_{sr}\right)_{s,r=1}^{m}$ there exists a polynomial-time algorithm which finishes processing of all the flows in an interval whose length is equal to the coflow effective size D (see Equation (2)). We present one example of such an algorithm in Algorithm 1, which was proposed originally in [36], and refer to it as BNA that stands for Birkhoff-von Neumann Algorithm. BNA returns a list of matchings L and a list of times τ. To schedule flows of $\mathcal{D}$, we use each matching $L(k)$ for $\tau(k+1) - \tau(k)$ time units, for $k = 1, \ldots, |L|$.

Algorithm 1 BNA for Single Coflow Scheduling

Given a coflow $\mathcal{D} = (d_{sr})_{s,r=1}^{m}$:

1) Let L be the list of matchings and τ be the list of starting times for each matching. Initially, $L = \varnothing$, $\tau = [0]$.
2) For any $s \in \mathcal{M}_S$ and $r \in \mathcal{M}_R$, compute d_s, d_r, and D according to Definition 1.
3) Find the set of tight nodes as $\Omega = (\arg\max_{s\in\mathcal{M}_s} d_s) \cup (\arg\max_{r\in\mathcal{M}_R} d_r)$.
4) Find a matching M among the source and destination nodes such that all the nodes in Ω are involved.
5) Find
$$t = \min\Big\{ \min_{(s,r)\in M} d_{sr}, \min_{s:(s,r)\notin M}(D - d_s), \min_{r:(s,r)\notin M}(D - d_r)\Big\}$$
6) Add M and $t+\tau[\text{end}]$ to the lists L and τ, respectively.
7) Update the flow sizes as $d_{sr} \leftarrow d_{sr} - t,\ \forall (s,r) \in M$.
8) While $\mathcal{D} \neq \mathbf{0}$, repeat Steps $2-7$.
9) Return L and τ.

It is immediate that the optimal makespan for a path job can be found in polynomial time, by optimally scheduling its coflows successively using BNA.

Lemma 1: Optimal makespan for a path job j is equal to $\sum_{c=1}^{\mu_j} D^{(cj)}$ where $D^{(cj)}$ is the effective size of coflow c of job j and the corresponding schedule can be constructed in polynomial time by successively using BNA.

We will use BNA in our algorithms in the rest of the paper.

IV. Makespan Minimization for Scheduling Multiple General DAG Jobs

A. DMA (Delay-and-Merge Algorithm)

For each job j, we consider a topological sorting of nodes in G_j, i.e., we sort its coflows (nodes) such that for every precedence constraint $c_1 \prec c_2$ (directed edge $c_1 \to c_2$), coflow c_1 appears before c_2 in the ordering. This ordering is not unique and can be found in polynomial time [37]. For example, for the job in Figure 1a, the orderings c_1, c_2, c_3, c_4, c_5, c_6, c_7 and c_2, c_3, c_1, c_5, c_4, c_6, c_7 are both valid topological sorts. We then re-index coflows from 1 to μ_j according to this ordering.

Further, we use Δ_j to denote the maximum load that a server should send or receive considering all of job j's coflows. Formally, for job j, consider an aggregate coflow $\mathcal{D}^j = \sum_{c=1}^{\mu_j} \mathcal{D}^{(cj)}$. Then, Δ_j is the effective size of $\mathcal{D}^j$ based on Definition 1. We also use Δ to denote the maximum load a node has to send or receive *considering all the jobs*.

Algorithm 2 (DMA) describes our algorithm for scheduling multiple general DAG jobs.

Note that in DMA, in each of the isolated schedules in Step 1 all the precedence constraints among coflows are respected. However, in Step 3, the link capacity constraints may be violated. In Step 4, in the final schedule, both link capacity constraints and precedence constraints among coflows are satisfied. The parameter $\beta > 1/e$ in DMA is a constant and has no effect on the theoretical result. However, it can be used to control the range of delays in practice.

Algorithm 2 DMA for Scheduling a General DAG G_j

1) For each job j, compute a topological sorting of nodes in G_j. Then, find a feasible schedule by optimally scheduling its coflows successively using BNA, i.e., $L_{cj}, \tau_{cj} = \text{BNA}(\mathcal{D}^{(cj)})$, for coflow $c = 1, \ldots, \mu_j$. We refer to these schedules as *isolated* schedules of jobs.
2) Delay each isolated schedule by a random integer time chosen uniformly in $[0, \Delta/\beta]$, for a constant $\beta > 1/e$, independently of other isolated schedules, i.e., $\tau_{cj} \leftarrow \tau_{cj} + t_j$ where t_j is the random delay of job j.
3) Greedily merge the delayed isolated schedules. I.e., for any time slot t, add corresponding matchings of different jobs.
4) Construct a feasible merged schedule. Let $\alpha_t \geq 1$ denote the maximum number of packets that a server needs to send or receive at time slot t in the merged schedule in Step 3. For each time slot t, consider an interval of length α_t, and use BNA to feasibly schedule all its packets.

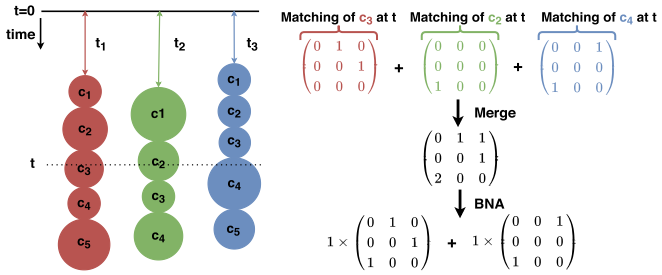


Fig. 2. Applying DMA on 3 multi-stage jobs. On the left side, a topological ordering and a random delay for each job are computed. On the right side, the merging procedure and BNA output is shown for some time t.

As an illustration, Figure 2 shows the procedure of DMA on 3 multi-stage jobs in a 3×3 switch network. On the left side, DMA computes a topological ordering for the coflows of each job and chooses a random delay for each job. The diameter of each node is proportional to the effective size of its corresponding coflow. Consider time slot t, DMA merges the matchings of coflow 3 of the red job, coflow 2 of the green job, and coflow 4 of the blue job, and inputs the result to BNA. Then, BNA computes two matchings, where each should be used for one time slot.

B. Performance Guarantee of DMA

The following theorem states the main result regarding the performance of DMA. The proof can be found in Section IX-A.

Theorem 2: Given a set $\mathcal{N}$ of jobs with general DAGs, DMA runs in polynomial time and provides a feasible solution whose makespan $C^{(\mathcal{N})}$ is at most $O(\mu g(m))$ of the optimal makespan with high probability, where $g(m) = \log(m)/\log(\log(m))$.

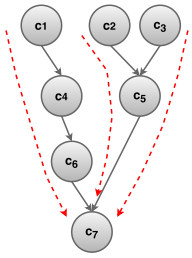


Fig. 3. A rooted tree with 3 path sub-jobs.

C. De-Randomization

Step 2 of DMA involves random choices of delays. There exist well-established techniques that one can utilize to de-randomized this step and convert the algorithms to deterministic ones. For instance, one approach for selecting good delays is to cast the problem as a vector selection problem and then apply techniques developed in [28], [38], [39].

V. Makespan Minimization for Scheduling Multiple Rooted Tree Jobs

Now we consider the case where each job is represented by a rooted tree (Definition 5). We propose an algorithm with an improved performance guarantee compared to the case of general DAGs. We would like to emphasize that the $O(m)$ approximation algorithm [5], [6] will not be improved if the graph is a rooted tree rather than a general DAG.

A. Delay-and-Merge for a Single Rooted Tree (DMA-SRT)

In this section, we develop an approximation algorithm for minimizing makespan of a single rooted-tree job and show that its solution is at most $O(\sqrt{\mu}\log(m\mu)/\log(\log(m\mu)))$ of the optimal makespan. Recall Definitions 5 and 6. In what follows, we assume that the rooted tree G_j has an orientation towards the root R_j (i.e. fan-in tree). For the case that edge orientations points away from the root (i.e. fan-out tree), the algorithm is similar. Recall that S_0 is the set of coflows with no in-edge in rooted tree G_j. For each coflow $c \in S_0$, we can find a directed path starting from c and ending at coflow (node) R_j. We call each of these paths a *path sub-job* of job j. We use $\mathcal{P}_j$ to denote the set of all path sub-jobs of job j. Recall that $T_{p,j}$ is the size of directed path $p \in \mathcal{P}_j$ and T_j is the size of the critical path (see Definition 3). Figure 3 shows a rooted tree with 3 path sub-jobs.

Algorithm 3 provides description of DMA-SRT. Note that the algorithm calculates the starting time of each coflow, t_c, such that all the precedence constraints of the coflow are satisfied. In other words, t_c is equal to the smallest time $t_{c,p}$ (starting time of c based on path p) that all its preceding coflows in G_j are completed. We say that c is scheduled according to p if $t_c = t_{c,p}$. Therefore, the merged schedule satisfies all the precedence constraints among coflows, although the link capacity constraints may be violated. DMA-SRT constructs a feasible merged schedule using BNA. Note that in Step 5, $\mathcal{D}$ is multiplied by l_I since each matching $L_c(i)$ runs for l_I time units in its corresponding isolated schedule. In the final schedule, both link capacity constraints and precedence constraints among coflows are satisfied.

Algorithm 3 DMA-SRT for Scheduling a Rooted Tree G_j

1) Find the set of path sub-jobs $\mathcal{P}_j$ of job j. For each path sub-job $p \in \mathcal{P}_j$, Choose a random integer time d_p uniformly in $[0, \Delta_j/\beta]$, for a constant $\beta > 1/e$, independent of other isolated schedules. Next, for each coflow $c \in p$, $p \in \mathcal{P}_j$, calculate the starting time of coflow c according to p, $t_{c,p} = d_p + \sum_{c' \prec c, c' \in p} D^{c'j}$.
2) Find the coflow sets S_i, $i = 0, \ldots, H_j - 1$ of job j according to Definition 6. For $i = 0, \ldots, H_j - 1$, and for each coflow c in S_i, find starting time of coflow c as $t_c = \min\{t_{c,p} | t_{c,p} \geq \max_{c' \in \pi_c}(t_{c'} + D^{(c'j)})\}$.
3) For each coflow c in G_j, find an optimal schedule for each coflow c using BNA, i.e., L_c, τ_c =BNA($\mathcal{D}^{(cj)}$). We refer to these schedules as *isolated* schedules. Then, delay the scheduling times by t_c, $\tau_c \leftarrow \tau_c + t_c$.
4) Follow Step 3 of DMA.
5) Follow Step 4 of DMA.

B. Multiple Rooted Tree Jobs

Now consider the case where we have multiple jobs where each job is a rooted tree. We seek to find a feasible schedule that minimizes the time to process all the jobs (makespan). Recall that μ is the maximum number of coflows in any job. We use Δ to denote the aggregate size of coflows of *all the jobs* (Definition 2).

The scheduling algorithm is based on DMA-SRT described in Section V-A. Specifically, we apply DMA-SRT to find a feasible schedule for each job in the set. Then we apply Steps 2, 3 and 4 of DMA, namely, we choose a random delay in $[0, \Delta/\beta]$ for a constant $\beta > 1/e$ for each individual schedule and delay it. Next, we merge the delayed schedules. Finally we use BNA algorithm to resolve any collisions in the merged schedule. We refer to this algorithm as DMA-RT.

C. Performance Guarantee of DMA-SRT and DMA-RT

Theorem 3: Given a single job j with rooted tree G_j, DMA-SRT runs in polynomial time and provides a feasible schedule whose makespan C_j is at most $O(\sqrt{\mu_j}h(m, \mu_j))$ of the optimal makespan with high probability, where $h(m, \mu) = \log(m\mu)/\log(\log(m\mu))$.

Theorem 4: Given a set $\mathcal{N}$ of jobs, each represented as a rooted tree, DMA-RT runs in polynomial time, and achieves a solution whose makespan $C^{(\mathcal{N})}$ is at most $O(\sqrt{\mu}g(m)h(m, \mu))$ of the optimal makespan with high probability.

The proofs of Theorems 3 and 4 are presented in Section IX-B.

VI. Total Weighted Completion Time Minimization

We are now ready to present our combinatorial approximation algorithm for minimizing the total weighted completion time of multi-stage jobs with release times. *In this*

section, we assume that the jobs have general DAGs, however, the results can be customized for the case that all the jobs are represented by rooted trees. We use ρ_j to denote the release time of job j, which implies that job j is available for scheduling only after time ρ_j.

A. Job Ordering

To formulate a relaxed LP (Linear Program) for our problem, we note that if we ignore the precedence constraints among coflows of a job and aggregate all its coflows, we obtain a single-stage job (a coflow), and our problem is reduced to traditional coflow scheduling problem [9], [11], [13], [40]. Here, we use an LP formulation for such constructed single-stage jobs, but with an extra constraint for each job which roughly captures the barrier constraints among its coflows.

Formally, for each job j, consider the aggregate coflow $\mathcal{D}^j = \sum_{c=1}^{\mu_j} \mathcal{D}^{(cj)}$. Let $\overline{\mathcal{M}} := \mathcal{M}_S \cup \mathcal{M}_R$. We use d_i^j, $i \in \overline{\mathcal{M}}$ to denote the load of coflow $\mathcal{D}^j$ on server i (see Definition 1). Recall Definition 3 and note that T_j is the lower bound on the required time to schedule multi-stage job j (in the original problem). Let $\mathcal{J}$ be any subset of jobs in $\mathcal{N}$. We formulate the following LP:

$$\min \sum_{j\in\mathcal{N}} w_j C_j \quad (\mathbf{LP}) \tag{3a}$$

$$\sum_{j\in\mathcal{J}} d_i^j C_j \geq \frac{1}{2}\Big(\sum_{j\in\mathcal{J}} (d_i^j)^2 + (\sum_{j\in\mathcal{J}} d_i^j)^2\Big),\ i\in\overline{\mathcal{M}}, \mathcal{J}\subseteq\mathcal{N} \tag{3b}$$

$$C_j \geq T_j + \rho_j, \quad j \in \mathcal{N}. \tag{3c}$$

Constraints (3b) capture the links' capacity constraints and are used to lower-bound the completion time variables. To see this, consider a (source or destination) server i and a subset of jobs $\mathcal{J}$. For each j in $\mathcal{J}$, the completion time C_j of its aggregate coflow $\mathcal{D}^j$, has to be at least the summation of loads of coflows $\mathcal{D}^{j'}$ on server i that finish before j plus its own load on server i. Also note that for every two coflows in the set $\mathcal{J}$, one finishes before the other one. Therefore, $\sum_{j\in\mathcal{J}} d_i^j C_j \geq \sum_{j\in\mathcal{J}} d_i^j (d_i^j + \sum_{j'\in\mathcal{J}, j'\prec j} d_i^{j'})$, where $j' \prec j$ means $C_{j'} \leq C_j$. From this, Constraint (3b) is derived easily.

Note that this LP has *exponentially many constraints*, since we need to consider all the subsets of $\mathcal{N}$. However, we do not need to explicitly solve this LP and we only need to find an efficient ordering of jobs. To do so, we utilize the combinatorial primal-dual algorithm that first proposed in [41] and later generalized in [13] to capture constraints of the form (3c) for parallel scheduling problems. The algorithm builds up a permutation of the jobs in the reverse order iteratively by changing the corresponding dual variables to satisfy some dual constraint. We have provided the detailed description of the combinatorial algorithm in Appendix X (Algorithm 5) for completeness. We show how we use this ordering to find the actual schedule of jobs' coflows in the next section.

Remark 1: Algorithm 5 in Appendix X runs in $O(n(\log(n)+m))$ time where n is the number of jobs and m is the number of servers. However, the time complexity of the best known algorithm for solving the LP used in [5], [6] is $O((n^2+m)^\omega \log((n^2+m)/\epsilon))$, where ω is the exponent of matrix multiplication and ϵ is the relative accuracy [42]. For the current value of $\omega \approx 2.37$ [43], the time complexity of Algorithm 5 is dramatically lower than the time complexity of solving the LP used in [5], [6].

B. Grouping Jobs

Let D_j denote the maximum load that a server has to send or receive considering all coflows of the jobs up to and including job j according to the computed ordering. In other words, D_j is the effective size of an aggregate coflow constructed from coflows of the first j jobs. Recall that T_j is size of the critical path in job j (Definition 3). Define $\gamma = \min_{s,r,c,j} d_{sr}^{cj}$ which is a lower bound on the time required to process any job. Also let $T = \max_j \rho_j + \sum_{j\in N}\sum_{c\in j}\sum_{s\in M_S}\sum_{r\in M_R} d_{sr}^{cj}$. The algorithm groups jobs into B groups as follows.

Choose B to be the smallest integer such that $\gamma 2^B \geq T$, and consequently define

$$a_b = \gamma 2^b, \text{ for } b = -1, 0, 1, \ldots, B. \tag{4}$$

Then the b-th interval is defined as the interval $(a_{b-1}, a_b]$ and the group $\mathcal{J}_b$ is defined as the subset of jobs whose $T_j + \rho_j + D_j$ fall within the b-th group, i.e.,

$$\mathcal{J}_b = \{j \in N : T_j + \rho_j + D_j \in (a_{b-1}, a_b]\};\ 0 \leq b \leq B. \tag{5}$$

This partition rule ensures that every job falls in some group.

C. Scheduling Each Group $\mathcal{J}_b$

To schedule jobs of each group $\mathcal{J}_b$, $b \in \{1, \cdots, B\}$, (defined by (5)), we use the DMA algorithm. We refer to this algorithm as G-DM algorithm which stands for *Grouping jobs, followed by Delay-and-Merge* algorithms. We summarize G-DM in Algorithm 4.

Algorithm 4 G-DM for Scheduling Multi-Stage Jobs

1) Find an efficient permutation of jobs using Algorithm 5 and re-index them.
2) Let D_j be effective size of the *aggregate* coflow constructed from coflows of the jobs up to and including job j. Also, let T_j be size of the critical path in job j.
3) Partition jobs into disjoint subsets $\mathcal{J}_b$, $b = 0, \ldots, B$ as in (5).
4) For each group $b = 1, \ldots, B$, wait until all jobs in $\mathcal{J}_b$ arrive, then apply the makespan minimization algorithm DMA to schedule them.

D. Performance Guarantee of G-DM

Recall that $g(m) = \log(m)/\log(\log(m))$, and $h(m,\mu) = \log(m\mu)/(\log(\log(m\mu))$. The following theorem states the main result regarding the performance of G-DM. Its proof can be found in Section IX-C.

Theorem 5: G-DM is a polynomial-time $O(\mu g(m))$-approximation algorithm for minimizing the total weighted completion time of multi-coflow jobs with release times.

For the case that we are given a set $\mathcal{N}$ of jobs, each represented as a rooted tree, we modify G-DM by using DMA-RT as the subroutine in the last step of G-DM. We denote the modified version as G-DM-RT. The proof of following Corollary can be found in Supplementary Materials.

Corollary 1: G-DM-RT is a polynomial-time algorithm with approximation ratio $O(\sqrt{\mu}g(m)h(m,\mu))$ to minimize the total weighted completion time of rooted-trees with release times.

VII. Empirical Evaluation

To demonstrate the gains in practice, we conducted extensive evaluations using a real workload. This workload has been widely used in coflow related research [6], [9]–[11]. We compared the performance of our algorithm G-DM-RT with the $O(m)$-algorithm in [5], [6] which is the previous state-of-the-art algorithm and compare its performance with that of our algorithm. In [5], [6], the authors have shown that their algorithm outperforms single-stage coflow scheduling algorithms by around 83%, and Aalo [7] by up to 33% for the case of equal weights for job (as Aalo cannot handle the weighted scenario). Hence, we only report comparison with this algorithm. The results indicate that our algorithm outperforms the $O(m)$-algorithm [5], [6] by up to 53% *in the same settings*. We also investigate the performance of the algorithms for different values of delaying parameter β, and problem size μ and m.

Workload: The workload is based on a Hive/MapReduce trace at a Facebook cluster with 150 racks, and only contains coflows information. The data set contains 267 coflows with μ_j ranging from 10 to 21170. Further, size of the smallest flow is equal to $\gamma = 1$, size of the largest flow is equal to 2472, and effective size of coflows, Δ_j, is between 5 and 232145. Finally, the maximum load a server should send or receive considering all the coflows, i.e., the effective size of the aggregate coflow, is equal to $\Delta = 440419$.

To assess performance of algorithms under different traffic intensity, we generate workloads with different number of machines (servers) by mapping flows of the original 150 racks to m machines with various values of m. This is done in a random fashion, e.g., for $m = 50$ every 3 randomly selected machines in the original data set are mapped to 1 machine in the new data set. To generate multi-stage jobs, we randomly partition the coflows into multi-stage jobs that each has $\overline{\mu}$ coflows on average. To generate the corresponding rooted tree, we first generate a random graph in which probability of picking each of the edges is 0.5, and then converting it to a tree by removing its cycles. We ran the algorithms for two cases of equal weights for all jobs and randomly selected weights from interval $[0, 1]$. We also consider the online scenario where multi-stage jobs arrive over time and their release (arrival) times follow a Poisson process with a parameter θ.

Algorithms: We simulate our multi-stage job algorithms (referred to as G-DM and G-DM-RT) and the algorithm in [5], [6] (referred to as O(m)Alg). For each algorithm, we present two versions, one with no backfilling and one with backfilling. Backfilling is a common technique in scheduling to increase utilization of system resources by allocating the underutilized link capacities (or servers, depending on the problem) to other jobs. We apply the same backfilling strategy to both algorithms for a fair comparison. We use G-DM-BF, G-DM-RT-BF, and O(m)Alg-BF to refer to the versions of algorithms with backfilling.

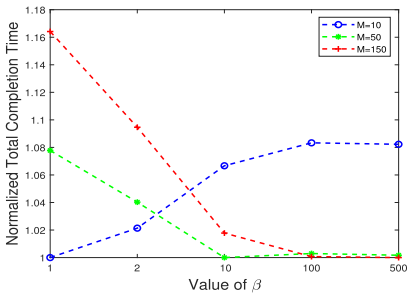


Fig. 4. Performance of G-DM-RT for different number of servers and different values of β, and $\bar{\mu} = 5$.

Metrics: We compare the total weighted completion times of jobs under the two algorithms for various workloads and scenarios. We present results for offline and online scenarios with equal and random job weights. We also investigate the performance of the algorithms for different values of m, $\bar{\mu}$, θ.

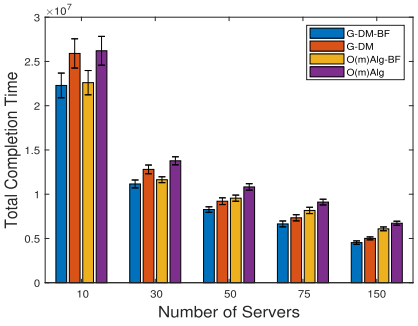
A. Impact of Random Delays and β

The current implementations of G-DM and G-DM-RT have a random component as it uses DMA and DMA-SRT as a subroutine. To show that in practice running the algorithm *once* is sufficient to achieve a satisfactory solution, we need to show that its relative standard deviation (RSD) is small. RSD is defined as standard deviation divided by the mean (average). Hence, to analyze the effect of random delays in the performance of our algorithm, we ran it on some instances, each for 10 times. Based on our experiments, RSDs of G-DM and G-DM-RT are always less than 0.5% and RSDs of G-DM-BF and G-DM-RT-BF are always less than 0.9%, which both are very small. In the rest of simulations, we run our algorithms only once on each instance.

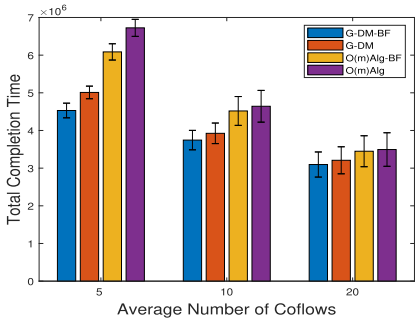
Furthermore, we studied the effect of parameter β (see Sections IV and V) on the performance of our algorithms. For each algorithm, we ran the algorithm using a wide range of β values. Based on our experiments, for smaller m (higher traffic intensity) it is better to choose a small value of β (1 or 2) to reduce the collision probability (13), while choosing larger β (100 or 500) for larger m helps the algorithm to use the unused capacity to schedule flows of other coflows in the system. Moreover, the amount of improvement by optimizing over β was less than 16% in all the experiments. Figure 4 shows the results for different values of β and m when $\bar{\mu}$ is set to 5 for G-DM-RT.

B. Evaluation Results for General GADs

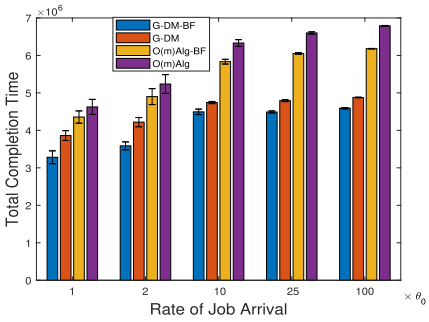
1) Offline Setting: In the offline scenario, all the jobs are available at time 0. For each set of parameters $(m, \bar{\mu})$, we generate 10 different instances randomly and report the average and standard deviation of each algorithm's performance.



(a) Performance of G-DM and O(m)Alg with and without backfilling for different numbers of servers, and $\bar{\mu} = 5$.

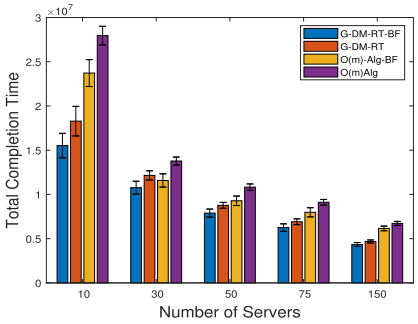


(b) Performance of G-DM and O(m)Alg with and without backfilling for different average numbers of coflows per job, and $m = 150$.

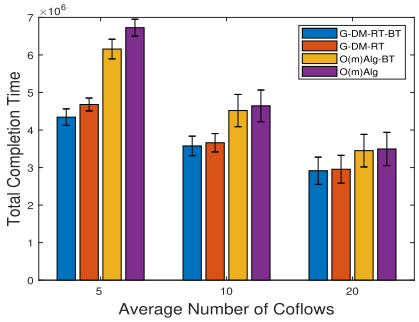


(c) Performance of G-DM and O(m)Alg with and without backfilling for different arrival rates, and $\bar{\mu} = 5$, $m = 150$.

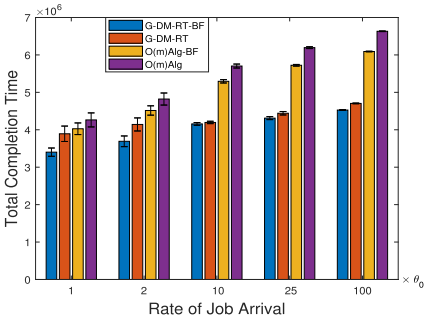
Fig. 5. Performance of G-DM and O(m)Alg for scheduling general DAGs with and without backfilling.



(a) Performance of G-DM-RT and O(m)Alg with and without backfilling for different numbers of servers, and $\bar{\mu} = 5$.



(b) Performance of G-DM-RT and O(m)Alg with and without backfilling for different average numbers of coflows per job, and $m = 150$.



(c) Performance of G-DM-RT and O(m)Alg with and without backfilling for different arrival rates, and $\bar{\mu} = 5$, $m = 150$.

Fig. 6. Performance of G-DM-RT and O(m)Alg for scheduling rooted tree jobs with and without backfilling.

Figure 5a and 5b depict some of the results for the case that jobs have general DAGs and equal weights. Figure 5a shows the performance of G-DM and O(m)Alg for the case that average number of coflows per job, $\bar{\mu}$, is 5 and different number of servers. G-DM performs as well as O(m)Alg for $m = 10$. It outperforms O(m)Alg from 9% for $m = 30$ to about 36% for $m = 150$. Moreover, Figure 5b shows that *our algorithm outperforms O(m)Alg for all values of average coflows per job, by* 36% *to* 11%. The results for the case of random job weights are very similar and omitted.

2) Online Setting: For the online scenario, jobs arrives to the system according to a Poisson process with rate θ. Every time that a job arrives both G-DM (G-DM-RT) and O(m)Alg suspend the previously active jobs, update the list of jobs and their remaining demands, and reschedule them. Moreover, completion time of a job in the online scenario is measured from the time that the job arrives to the system. The job arrival rate is determined as follows: $\theta = a \times \theta_0$ for $a = \{1, 2, 10, 25, 100\}$, and $\theta_0 = \frac{\sum_j \mu_j}{\sum_j \sum_c D^{cj}}$, in which $\sum_j \mu_j$ is the total number of coflows among all jobs. The denominator, $\sum_j \sum_c D^{cj}$, is summation of coflows' effective sizes and an upper bound on the jobs' makespan.

Figure 5c shows the results under G-DM and O(m)Alg for the case that $m = 150$ (original data set), $\bar{\mu} = 5$, and all the jobs have equal weights. G-DM always outperforms O(m)Alg, from 20% to 36%. Furthermore, *G-DM-RT-BF always outperforms O(m)Alg-BF, by* 30% *to* 37%.

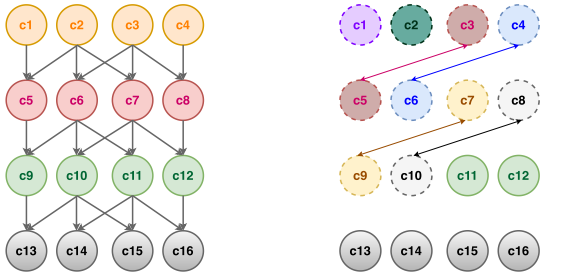
C. Evaluation Results for Rooted Trees

Now we provide the simulation results for the case that all the jobs are rooted trees.

1) Offline Setting: Figure 6a shows the performance of two algorithms for different number of servers, $\bar{\mu} = 5$, and equal weights for jobs. As we can see, G-DM-RT always outperforms O(m)Alg, for about 53% for $m = 10$ to about 46% for $m = 150$. For all values of average coflows per jobs, *our algorithm outperforms O(m)Alg, by* 46% *to* 18% as depicted in Figure 6b.

2) Online Setting: Figure 6c shows the results with and without backfilling for the case that $m = 150$ (original data set), $\bar{\mu} = 5$, and all the jobs have equal weights. G-DM-RT always outperforms O(m)Alg, from 10% to 46%. Furthermore, *G-DM-RT-BF always outperforms O(m)Alg-BF, by* 22% *to* 36%.

We would like to point out that, as we expect, the gain under G-DM-RT is greater than G-DM, as the former algorithm utilizes the network resources more efficiently by interleaving schedules of different coflows of the *same job* as well as interleaving schedules of coflows of *different jobs*. Furthermore,



(a) A DAG job with 16 coflows. (b) Scheduling of coflows.

Fig. 7. An example of a DAG with $C_{opt} = \Omega(\sqrt{\mu}(\Delta + T))$.

backfilling strategy generally yields a larger improvement when combined by G-DM and O(m)Alg compared to G-DM-RT, as they leave more resources unused.

VIII. Discussion on Approximation Results

An interesting research direction is to improve the approximation ratios for the algorithms. As we showed in the previous sections, once we have an algorithm for scheduling a single job whose solution is a factor η of the simple lower bounds Δ_j and T_j (Definitions 2 and 3), we can directly utilize the rest of our approach and get approximation algorithms with approximation ratio $O(\eta \log(m)/\log\log(m))$ for the problems of makespan minimization and total weighted completion time minimization for multiple jobs.

To improve the result for the case of general DAGs, one approach is to first consider scheduling a single job (with a general DAG), and try to generalize DMA-SRT to a general DAG by careful construction of paths in the algorithm, so that we do not need to consider all the paths in the DAG which could be exponentially many. However, even if one could show that $O(\mu_j)$ paths is sufficient to construct a feasible schedule, it is challenging to analyze the performance through computing the probability of collisions or the average number of collisions in the merged schedule as we did in proof of Lemma 4. This is due to the underlying dependency among the unrelated coflows in G_j (these are coflows among which there is no directed path in G_j, thus they can collide) which appears in the probability that a given coflow is assigned to start scheduling at a given time, given that $O(\mu_j)$ paths are generated by the algorithm.

Besides these challenges for scheduling a job with a general DAG, we can show the existence of instances for which the optimal makespan is $\Omega(\sqrt{\mu_j})$ factor larger than the two simple lower bounds, Δ_j and T_j, as stated it in the following lemma.

Lemma 2: There exist arbitrary sized instances of DAG job scheduling such that its optimal makespan is $\Omega(\sqrt{\mu_j}(\Delta_j + T_j))$.

Proof: Consider a DAG job with μ_j coflows to be scheduled in an $m \times m$ switch, with $\mu_j = (2K)^2$ for some K and $m > 2K$. Recall that T_j and Δ_j denote the size of its critical path and its aggregate size, respectively. For simplicity, we drop the subscript j. We construct the job as follows. First, we describe the demand matrix of each coflow. For coflows $c = 1, \ldots, 2K$, each coflow has a single flow of size d from server 1 to server 2, where $2K = \sqrt{\mu}$ by assumption. These coflows are the root nodes in the job's DAG. For coflows $c = i(2K) + 1, \ldots, (i+1)(2K)$, $i = 1, \ldots, (2K) - 1$, each coflow has a single flow of size d from server $i+1$ to server $i+2$. Now we specify the precedence constraints among coflows. We construct G_j such that its height is $\sqrt{\mu} = 2K$ and each of its coflow set has $\sqrt{\mu} = 2K$ coflows (see Definition 6). Consider coflow $c \in S_i$ for $i = 1, \ldots, H_j - 1$. If $i(2K) + 1 \leq c \leq (i + 1/2)(2K)$, then the parent set of coflow c is $\pi_c = \{c' | c - 2K \leq c' \leq c - K - 1\}$. If $(i + 1/2)(2K) + 1 \leq c \leq (i+1)(2K)$, then the parent set of coflow c is $\pi_c = \{c' | c - 3K + 1 \leq c' \leq c - 2K\}$. Figure 7a shows an example with $\mu = 16$. For the constructed DAG, it is easy to see that $T = \Delta = 2Kd = \sqrt{\mu}d$.

Next, we specify an optimal schedule for the constructed DAG, and compute its makespan denoted by C_{opt}. We first schedule coflows $1, \ldots K$, which takes Kd amount of time. We then schedule coflows $K+1$ and $2K+1$ simultaneously. This is feasible since there is no precedence constraint between these two coflows, all the parents of coflow $2K+1$ has been scheduled, and the two coflows do not share a server. Similarly, we schedule coflows $2(i - 1/2)K + c$ and $2iK + c$, for $i = 1, \ldots, 2K-1$ and $c = 1, \ldots, K$ at the same time. Finally, we schedule the last K coflows, $c = 4K^2 - K + 1, \ldots, 4K^2$ back to back which takes Kd amount of time. For instance, consider the example of Figure 7. Coflow c_1 and c_2 are scheduled back to back from time 0 to $2d$. Then coflow c_3 and c_5 get scheduled from $2d$ to $3d$ and so on. Figure 7b shows the instance at which the first ten coflows (the coflows with dashed lines) are scheduled. The coflows with the same color (that are also linked by an arrow) have been scheduled at the same time.

By scheduling coflows in this fashion, all the precedence constraints and capacity constraints are respected. Moreover, the length of the schedule is $C_{opt} = (2K+1)K \times d = \Omega(\mu d)$. Therefore, $C_{opt} = \Omega(\sqrt{\mu}(\Delta + T))$. □

IX. Proofs of Main Results

In this section, we provide detailed proofs of the theorems stating performance guarantees for the proposed algorithms. Recall that $g(m) = \log(m)/\log(\log(m))$, and $h(m, \mu) = \log(m\mu)/(\log(\log(m\mu))$.

A. Proofs Related to DMA

To prove Theorem 2, we first bound the length of the infeasible solution that is constructed at the end of Step 3 of DMA (Lemma 3). We then show a result (Lemma 4) that bounds the average number of *packet collisions* occurred in the infeasible solution and use it to prove that the final solution is bounded with a hight probability. This last step uses the ideas in the proof of Theorem 1.2 in [29]. Finally we show that DMA runs in polynomial time by carefully dividing the time horizon into the intervals on which the algorithm has to compute matching of source nodes to destination nodes for flow transmission (Lemma 6).

Lemma 3: The length of the infeasible merged schedule (Step 3) is at most $(\mu + 1/\beta)\Delta$.

Proof: First note that the isolated schedule for job j in Step 1 spans from 0 to at most $\mu_j\Delta_j$, since the effective size of each of its coflows is at most Δ_j. By delaying the isolated schedules by at most Δ/β, length of the infeasible merged schedule is at most $\max_j(\mu_j\Delta_j) + \Delta/\beta$ which is bounded from above by $(\mu + 1/\beta)\Delta$. □

Lemma 4: Let $\alpha_t \geq 1$ denote the maximum number of packets that a server needs to send or receive at time slot t in the merged schedule (Step 3). For any $t \in [0, (\mu+1/\beta)\Delta]$, $\mathbb{E}[\alpha_t] = O(g(m))$.

Proof: Let $\overline{\mathcal{M}} := \mathcal{M}_S \cup \mathcal{M}_R$. To prove the lemma, we define random variable z_{ijt} to be 1 if some flow of job j with an end point on server i is scheduled at time slot t. Then $\alpha_t = \max_{i\in\overline{\mathcal{M}}} \sum_{j\in\mathcal{N}} z_{ijt}$. Further, note that due to the random delay of jobs' isolated schedules, variables z_{ijt}, $j \in \mathcal{N}$ are mutually independent. Let $\delta = ag(m)$ for some constant a such that $\delta > 1$. Therefore,

$$\mathbb{E}[\delta^{\alpha_t}] = \mathbb{E}[\delta^{\max_{i\in\overline{\mathcal{M}}} \sum_{j\in\mathcal{N}} z_{ijt}}] \leq \mathbb{E}\Big[\sum_{i\in\overline{\mathcal{M}}} \delta^{\sum_{j\in\mathcal{N}} z_{ijt}}\Big]. \quad (6)$$

Define p_{ijt} to be the probability that $z_{ijt} = 1$. By the independent property of z variables, we can write

$$\begin{aligned}\mathbb{E}\Big[\delta^{\sum_{j\in\mathcal{N}} z_{ijt}}\Big] &= \Pi_{j\in\mathcal{N}}\mathbb{E}\Big[\delta^{z_{ijt}}\Big] \\ &= \Pi_{j\in\mathcal{N}}(p_{ijt}\delta + (1-p_{ijt})) \\ &\leq \Pi_{j\in\mathcal{N}} e^{p_{ijt}(\delta-1)} = e^{(\delta-1)\sum_{j\in\mathcal{N}} p_{ijt}} \\ &= e^{(\delta-1)\mathbb{E}[\sum_{j\in\mathcal{N}} z_{ijt}]} \leq e^{\beta(\delta-1)}, \end{aligned} \quad (7)$$

where the last inequality is due to $\mathbb{E}\Big[\sum_{j\in\mathcal{N}} z_{ijt}\Big] \leq \beta$. This is because by choosing delays uniformly at random, $\mathbb{E}[z_{ijt}]$ is at most the load of job j on server i divided by Δ/β, i.e., $\beta d_i^j/\Delta$, where d_i^j is the load of job j (or equivalently the aggregate coflow $\mathcal{D}^j$) on server i (see Definition 1). Thus,

$$\mathbb{E}\Big[\sum_{j\in\mathcal{N}} z_{ijt}\Big] = \sum_{j\in\mathcal{N}} \mathbb{E}[z_{ijt}] \leq \beta,$$

as $\sum_{j\in\mathcal{N}} d_i^j \leq \Delta$ by definition.

Combining Inequality (6) and (7), and by Jensen's inequality we can write,

$$\delta^{\mathbb{E}[\alpha_t]} \leq \mathbb{E}[\delta^{\alpha_t}] \leq \sum_{i\in\overline{\mathcal{M}}} e^{\beta(\delta-1)} = 2me^{\beta(\delta-1)}. \quad (8)$$

Now, note that if we choose a sufficiently large, then $2me^{\beta(\delta-1)} \leq \delta^\delta$, by definition of $g(m)$. Therefore, we can conclude that $\mathbb{E}[\alpha_t] \leq \delta$, and the proof is complete. □

Lemma 5: For any $\epsilon > 0$, the probability that the length of the final schedule (Step 4) is greater than $O(g(m))(\mu+1/\beta)\Delta$, is less than ϵ.

Proof: Recall that the constructed merged schedule (Step 3) spans from time 0 to at most $(\mu + 1/\beta)\Delta$ due to Lemma 3. Note that, the length of the final schedule is at most $\sum_{t\in[0,(\mu+1/\beta)\Delta)} \alpha_t$. Using Lemma 4 and Markov's inequality,

$$\mathbb{P}\left(\sum_{t\in[0,(\mu+1/\beta)\Delta)} \alpha_t \geq (a/\epsilon)g(m)(\mu+1/\beta)\Delta\right) \leq \epsilon \quad (9)$$

Therefore, the proof is complete. □

Lemma 6: Steps 3 and 4 in DMA can be executed in polynomial time.

Proof: In view of Steps 3 and 4 in DMA algorithm, we may need to run BNA for $(\mu + 1/\beta)\Delta$ times. However, in the case that Δ is not polynomially bounded in m, n, and μ, we can modify the last step of DMA to ensure that it runs in polynomial time. To do so, define $H = \{\tau_{cj} | c \in G_j, j \in \mathcal{N}\}$ and $L = \{L_{cj} | c \in G_j, j \in \mathcal{N}\}$, to be the set of all scheduling times and matchings. we sort H, and let $\mathcal{I}$ be the set of time intervals created from elements of B, $\mathcal{I} = \{[h_k, h_{k+1}) | k = 1, \dots |H| - 1\}$. Thus, $\mathcal{I}$ consists of the time intervals during which the corresponding matching of every coflow is fixed.

For each interval I in $\mathcal{I}$, we merge the matchings of coflows, namely $L_{cj}(k)$'s, for which the interval I is entirely in the corresponding time interval $[\tau_{cj}(k), \tau_{cj}(k+1))$. In other words, we compute

$$\mathcal{D} = \sum_{c,j,k: I\subseteq[\tau_{cj}(k),\tau_{cj}(k+1))} L_{cj}(k).$$

Finally, for each merged matching $\mathcal{D}$, we find an optimal schedule using BNA, i.e., $L, \tau =$BNA$(l_I \times \mathcal{D})$, where l_I is length of the interval I of merged matching $\mathcal{D}$. Then we schedule demand matrix $l_I \times \mathcal{D}$ according to L and τ.

Note that whenever we run BNA, the number of elements in the list L, output of BNA, is at most m^2. This is because according to line 5 in BNA, at each iteration, t is computed such that at least one node becomes tight (i.e., it appears in the set Ω of line 3 in the next iteration) or a flow completes. Further, $|\tau| = |L| + 1$ and the last element of τ is D. Hence, in view of Steps 3 and 4 in DMA algorithm, we need to run BNA for at most $O(\mu nm^2)$ times as the number of intervals in the set $\mathcal{I}$ is $O(\mu nm^2)$. Combining this with the fact that BNA runs in polynomial time, the proof is complete. □

Proof: [Proof of Theorem 2] Steps 1 and 2 in DMA can be executed in polynomial time. Combining this with Lemma 6, we can easily conclude that DMA runs in polynomial time.

Moreover, given that Δ is a lower bound for the optimal makespan, β is a constant, and Lemma 5, we conclude that makespan of the final schedule is at most $O(\mu g(m))$ of the optimal makespan with high probability. □

B. Proofs Related to DMA-SRT and DMA-RT

Consider DMA-SRT. Let $\alpha_t \geq 1$ denote the maximum number of packets that a server needs to send or receive at time slot t in the infeasible merged schedule (Step 5). Note that, Step 5 in our algorithm and the following proof is more involved than the corresponding steps in the scheduling algorithm designed for DAG-SSP (DAG-shop scheduling problem) in [28], as the constraint of scheduling one task at a time in the latter problem converts the job's DAG to a path job.

Lemma 7: For any $\epsilon > 0$, $\max_t \alpha_t \leq k_\epsilon\sqrt{\mu_j}h(m, \mu_j)$, with probability greater than $(1-\epsilon)$, for a constant k_ϵ depending on ϵ, for $t \in [0, \Delta_j/\beta + T_j]$.

Proof: To prove the lemma, let P denote the probability that any server at any time is assigned more than α packets (to be specified shortly). In what follows we first bound P_0

the probability that at least α packets are scheduled to be sent or received by a server i at time t. Note that there are at most $\binom{\Delta_j}{\alpha}$ ways to choose α packets from those that have an end point (source or destination) on server i. For packet u, the probability that it is scheduled at time t is at most $\beta|\mathcal{P}_{u,j}|/\Delta_j$, where, $\mathcal{P}_{u,j} \subseteq \mathcal{P}_j$ is the set of path-jobs containing packet u (or equivalently, the coflow to which packet u belongs.). That is because of the random uniform delay for scheduling coflows in S_0. More precisely, let $E_{u,t}$ be the event that a specific packet u is scheduled at time t and P_u be the probability that $E_{u,t}$ happens. Furthermore, let $E_{u\in p}$ denote the event that scheduling of u in the final schedule is according to the schedule of path-job p. Then,

$$P_u = \sum_{p\in\mathcal{P}_{u,j}} \mathbb{P}\{E_{u,t}, E_{u\in p}\} \overset{(1\star)}{=} \sum_{p\in\mathcal{P}_{u,j}} \mathbb{P}\{d_p = t_p, E_{u\in p}\}$$
$$= \sum_{p\in\mathcal{P}_{u,j}} \mathbb{P}\{E_{u\in p}|d_p = t_p\}\mathbb{P}\{d_p = t_p\}. \quad (10)$$

Equality $(1\star)$ is because the probability that packet u is scheduled at t *and* according to the path-job p is equal to the probability that path-job p is delayed by some specific time t_p *and* packet u is scheduled according to the path-job p. Regardless of the value of t, the probability that path-job p is delayed by t_p is either β/Δ_j or zero (if $t_p < 0$). Hence,

$$P_u \le \frac{\beta}{\Delta_j} \sum_{p\in\mathcal{P}_{u,j}} \mathbb{P}\{E_{u\in p}|d_p = t_p\} \le \frac{\beta|\mathcal{P}_{u,j}|}{\Delta_j}. \quad (11)$$

Moreover, for two different packets u and v with at least a common (source or destination) server, the probability that they collide (i.e., are assigned to the same time slot) is zero if they both belong to the same coflow or same path-job, due to the feasible scheduling of each coflow and satisfaction of precedence constraints at each path-job. Otherwise, the probability that the two events $E_{u,t}$ and $E_{v,t}$ happen can be upper-bounded by multiplications of two terms of the form $\beta|\mathcal{P}_{.,j}|/\Delta_j$ (using arguments similar to (10) and (11)), since the random delays are chosen independently. Therefore,

$$P_0 \le \binom{\Delta_j}{\alpha} \Pi_{i=1}^{\alpha} P_{u_i} \le (\frac{e\Delta_j}{\alpha})^{\alpha} (\frac{\beta}{\Delta_j})^{\alpha} \times \Pi_{i=1}^{\alpha} |\mathcal{P}_{u_i,j}|$$
$$\overset{(2\star)}{\le} (\frac{e\beta\mu_j}{\alpha^2})^{\alpha}. \quad (12)$$

Note that the size of set $\mathcal{P}_j$ is bounded by $|S_0|$ (and therefore μ_j) as there is only one path for any coflow in S_0 to coflow R_j. Therefore, $\sum_{i=1}^{\alpha} |\mathcal{P}_{u_i,j}| \le |\mathcal{P}_j| \le \mu_j$. Combining this with the fact that $\Pi_{i=1}^{\alpha}|\mathcal{P}_{u_i,j}|$ is maximized when $|\mathcal{P}_{u_i,j}| = \mu_j/\alpha$, Inequality $(2\star)$ follows.

If we choose $\alpha = k_\epsilon\sqrt{\mu_j}$ then $P_0 \le (m\mu_j)^{-(k_\epsilon - 1)}$. Hence, the probability that any server at any time is assigned more than α packets can be bounded by $P \le 2m(\Delta_j + T_j)P_0 < 2m(\Delta_j + T_j)(m\mu_j)^{-(k_\epsilon-1)}$. This last step is similar to the argument in [28], [44], for job shop scheduling problem. To specify k_ϵ, note that we require P to be less than $\epsilon > 0$, which is satisfied by choosing k_ϵ as

$$k_\epsilon \ge \log_{m\mu_j}\big(\frac{2m(\Delta_j + T_j)}{\epsilon}\big) + 1. \quad (13)$$

We now need to show that k_ϵ is a constant by showing that $\Delta_j + T_j$ is polynomially bounded in m and μ_j. Let δ_j denote the maximum size of a flow in job j. Note that $\Delta_j + T_j$ is polynomially bounded in m, μ_j and δ_j. In the case that δ_j is polynomially bounded in m and μ_j, it is easy to see that by choosing k_ϵ according to (13), with probability $(1-\epsilon)$, there is at most $k_\epsilon(\sqrt{\mu_j}h(m,\mu_j)$ packets on any server at any time. If δ_j is not polynomially bounded in m and μ_j, we round down each flow size d_{sr}^{cj} to the nearest multiple of $\delta_j/m^2\mu_j$ and denote it by $d_{sr}^{\prime cj}$. This ensures that we have at most $m^2\mu_j$ distinct values of modified flow sizes. Therefore, we can treat $d_{sr}^{\prime cj}$ as integers in $\{0, 1, \ldots, m^2\mu_j\}$ and trivially retrieve a schedule for $d_{sr}^{\prime cj}$ by rescaling. Let $\mathcal{S}'$ denote this schedule. If we increase the flow sizes from $d_{sr}^{\prime cj}$ to d_{sr}^{cj} in $\mathcal{S}'$ by increasing the length of the last matching that flow (s, r, c, j) is scheduled in and achieve schedule $\mathcal{S}$, we can argue that the length of $\mathcal{S}$ and $\mathcal{S}'$ differs in at most δ_j amount. This is because there are at most $m^2\mu_j$ number of flows. Thus, length of $\mathcal{S}$ is at most $(k_\epsilon + 1)\sqrt{\mu_j}h(m,\mu_j)$ as $\delta_j \le T_j$. □

We are now ready to prove Theorem 3 and Theorem 4.

Proof: [Proof of Theorem 3] It is easy to see that steps 1-3 of DMA-SRT can be done in polynomial time. By Lemma 6, Steps 4 and 5 of DMA-SRT are also executed in polynomial time. Therefore, DMA-SRT is a polynomial time algorithm.

Moreover, the completion time of each coflow is bounded by $\Delta_j/\beta + T_j$, since the maximum delay is Δ_j/β and the maximum starting time of coflow c is $T_j - D^{(cj)}$. Using Lemma 7, we conclude that the length of the final schedule is at most $O(\sqrt{\mu_j}h(m,\mu_j))(\Delta_j/\beta + T_j)$ with a high probability. Given that both Δ_j and T_j are lower bounds for the optimal makespan, the proof is complete. □

Proof: [Proof of Theorem 4] The proof is similar to proof of Theorem 2. Using DMA-SRT, completion time of job j is $O(\sqrt{\mu_j}h(m,\mu_j)) \times (\Delta_j/\beta + T_j)$. Delaying and merging these schedules and applying an argument similar to proof of Lemma 4 and 5, we can conclude that the final solution is bounded from above by $O(\sqrt{\mu}g(m)h(m,\mu)) \times (\Delta/\beta + \max_j T_j)$. Combining this with the fact that both Δ and $\max_j T_j$ are lower bounds on the optimal makespan, we can conclude the result. □

C. Proofs Related to G-DM

We use $\tilde{C}_j$ to denote the optimal solution to LP (3) for the completion time of job j, and use $\widetilde{\text{OPT}} = \sum_j w_j\tilde{C}_j$ to denote the corresponding objective value. Similarly we use $C_j^\star$ to denote the optimal completion time of job j in the original job scheduling problem, and use $\text{OPT} = \sum_j w_j C_j^\star$. The lemma below establishes a relation between $\widetilde{\text{OPT}}$ and OPT.

Lemma 8: The optimal value of the LP, $\widetilde{OPT}$, is a lower bound on the optimal total weighted completion time OPT of multi-stage coflow scheduling problem, i.e., $\widetilde{OPT} \le OPT$.

Proof: It is easy to see that an optimal solution for the original multi-stage job scheduling problem is a feasible solution to LP (3) from which the lemma's statement follows. □

We next prove performance of G-DM given performance of the subroutine algorithm used in the last step of G-DM.

Lemma 9: Consider an algorithm, ALG, that generates a feasible job schedule such that for any job j, $C_j^{ALG} = O(\zeta)(T_j + \rho_j + D_j)$, for some ζ. Then if algorithm ALG is used as the subroutine in the last step of G-DM, we have $\sum_j w_j C_j^{ALG} = O(\zeta) \times OPT$, where C_j^{ALG} is completion time of job j under ALG.

The proof of Lemma 9 is provided in Supplementary Materials. We now prove the main result of Section VI.

Proof: [Proof of Theorem 5] Recall that $\tilde{C}_j$ is the optimal completion time of job j according to the LP (3). Let $\widehat{C}_j$ denote the actual completion time of job j under G-DM. Also, let l_j be the index of the group to which job j belongs based on (5). Let j_b be the last job in group b, and T_b be the maximum size of critical paths of jobs in group b. Also let $C^{(\mathcal{J}_b)}$ be the amount of time spent on processing all the jobs in $\mathcal{J}_b$. Then,

$$\widehat{C}_j \overset{(1\star)}{\leq} \max_{k\in b, b\leq l_j} \rho_k + \sum_{b=0}^{l_j} C^{(\mathcal{J}_b)} \overset{(2\star)}{\leq} a_{l_j} + O(\mu g(m)) \sum_{b=0}^{l_j} a_b \tag{14}$$

where $g(m) = \log(m)/\log(\log(m))$. Inequality (1$\star$) bounds the completion time of job j with sum of two terms: the first term is the maximum release time of the jobs in the first l_j groups (note that $\max_{k\in b, b\leq l_j} \rho_k$ can possibly be greater than ρ_j); The second term is the total time the algorithm spends on scheduling jobs of previous groups plus the time it spends on scheduling $\mathcal{J}_{l_j}$. Inequality (2$\star$) follows from Lemma 5 and the fact that $\max_{k\in b, b\leq l_j} \rho_k \leq a_{l_j}$, and D_b and T_b are both bounded by a_b for every job $k \in \mathcal{J}_b$. From (4),

$$\sum_{b=0}^{l_j} a_b = \gamma \sum_{b=0}^{l_j} 2^b = \gamma 2^{(l_j+1)} - 1 < 2a_{l_j}. \tag{15}$$

Combining (14) with (15), and the fact that $a_{l_{j-1}} = a_{l_j}/2$,

$$\widehat{C}_j < O(\mu g(m)) a_{l_{j-1}} \overset{(3\star)}{<} O(\mu g(m))(T_j + \rho_j + D_j)$$

where (3$\star$) is because $T_j + \rho_j + D_j$ falls in $(a_{l_j-1}, a_{l_j}]$. This inequality combined with Lemma 9, when DMA is replaced ALG and used as the subroutine, implies the result. □

X. Conclusion

In this work, we studied the problem of scheduling coflows of multi-stage jobs in order to minimize their makespan or total weighted completion time. This problem is practically well-motivated, involves new challenges, and deserves further study. As we showed through an example, it is not possible for an approximation algorithm to provide a solution that is within $o(\sqrt{\mu})$ of the two simple lower bounds for a job with a general DAG. An interesting future problem will be to improve the approximation ratio results of this paper.

Appendix
Combinatorial Algorithm for Job Ordering

In this section, we provide the detailed explanation of the combinatorial algorithm used in G-DM to find a good permutation of jobs and proof of Lemma 9.

Recall LP (3). Define $f_i(\mathcal{J})$ to be the right-hand side of Constraints (3b) for server i and subset of jobs $\mathcal{J}$, i.e.,

$$f_i(\mathcal{J}) = \frac{1}{2}\Big(\sum_{j\in\mathcal{J}} (d_i^j)^2 + (\sum_{j\in\mathcal{J}} d_i^j)^2\Big). \tag{16}$$

We now formulate dual of LP (3) as follows:

$$\max \sum_{i\in\overline{\mathcal{M}}} \sum_{\mathcal{J}\subseteq\mathcal{N}} \lambda_{i,\mathcal{J}} f_i(\mathcal{J}) + \sum_{j\in\mathcal{N}} \eta_j (T_j + \rho_j) \quad (\textbf{Dual LP})$$

$$\sum_{i\in\overline{\mathcal{M}}} \sum_{\mathcal{J}: j\in\mathcal{J}} d_i^j \lambda_{i,\mathcal{J}} + \eta_j \leq w_j, \quad j \in \mathcal{N} \tag{17b}$$

$$\eta_j \geq 0, \quad j \in \mathcal{N} \tag{17c}$$

$$\lambda_{i,\mathcal{J}} \geq 0, \quad i \in \overline{\mathcal{M}},\ \mathcal{J} \subseteq \mathcal{N}. \tag{17d}$$

The algorithm is presented in Algorithm 5. Let $\mathcal{N}'$ be the set of unscheduled jobs, initially $\mathcal{N}' = \mathcal{N}$. Also, set $\eta_j = 0$ for $j \in \mathcal{N}$. Define Λ to be the set of $\lambda_{i,\mathcal{J}}$'s that get specified in the algorithm, and initialize $\Lambda = \varnothing$ (to avoid initializing all the $\lambda_{i,\mathcal{J}} = 0$, which takes exponential amount of time) (line 1). In any iteration, let j be the unscheduled job with the greatest $T_j + \rho_j$, let ϕ be the server with the highest load and let d_ϕ be the load on server ϕ (lines 3 and 4). Now, if $T_j + \rho_j > d_\phi$, we raise the dual variable η_j until the corresponding dual constraint is tight and place job j to be the last job in the permutation (lines 5-7). However, if $T_j + \rho_j \leq d_\phi$, we choose job j' as in line 9. Then we define the dual variable $\lambda_{\phi,\mathcal{N}'}$, set it so that the dual constraint for job j' becomes tight, and place job j' to be the last in the permutation (lines 10-12).

Algorithm 5 Combinatorial Algorithm for Job Ordering

Given a set of multi-stage jobs $\mathcal{N}$:

1: $\mathcal{N}' = \mathcal{N}$, $\eta_j = 0$ for $j \in \mathcal{N}$, $\Lambda = \varnothing$.
2: **for** $k = n, n-1, \ldots, 1$ **do**
3: $\quad \phi(k) = \arg\max_{i\in\overline{\mathcal{M}}} d_i$
4: $\quad j = \arg\max_{l\in\mathcal{N}'} T_l + \rho_l$
5: $\quad$ **if** $T_j + \rho_j > d_{\phi(k)}$ **then**
6: $\quad\quad \eta_j = w_j - \sum_{i\in\overline{\mathcal{M}}} \sum_{\mathcal{J}, j\in\mathcal{J}} d_i^j \lambda_{i,\mathcal{J}}$.
7: $\quad\quad \sigma(k) = j$.
8: $\quad$ **else**
9: $\quad\quad j' = \arg\min_{j\in\mathcal{N}'} \big(\frac{w_j - \sum_{i\in\overline{\mathcal{M}}} \sum_{\mathcal{J}, j\in\mathcal{J}} d_i^j \lambda_{i,\mathcal{J}}}{d^j_{\phi(k)}}\big)$.
10: $\quad\quad \lambda_{\phi(k),\mathcal{N}'} = \big(\frac{w_{j'} - \sum_{i\in\overline{\mathcal{M}}} \sum_{\mathcal{J}, j'\in\mathcal{J}} d_i^{j'} \lambda_{i,\mathcal{J}}}{d^{j'}_{\phi(k)}}\big)$
11: $\quad\quad \Lambda \leftarrow \Lambda \cup \{\lambda_{\phi(k),\mathcal{N}'}\}$.
12: $\quad\quad \sigma(k) = j'$.
13: $\quad$ **end if**
14: $\quad \mathcal{N}' \leftarrow \mathcal{N}'/\sigma(k)$.
15: $\quad d_i \leftarrow d_i - d_i^{\sigma(k)},\ \forall i \in \overline{\mathcal{M}}$.
16: **end for**
17: Output permutation σ.

References

[1] (2019). *Apache Hadoop*. [Online]. Available: http://hadoop.apache.org
[2] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proc. HotCloud*, Jun. 2010, p. 10.

[3] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *ACM SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, 2007.
[4] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Topics Netw. (HotNets)*, 2012, pp. 31–36.
[5] B. Tian *et al.*, "Scheduling dependent coflows to minimize the total weighted job completion time in datacenters," *Comput. Netw.*, vol. 158, pp. 193–205, Jul. 2019.
[6] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 864–872.
[7] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 393–406, 2015.
[8] (2019). *Apache Hive*. [Online]. Available: https://hive.apache.org
[9] M. Shafiee and J. Ghaderi, "An improved bound for minimizing the total weighted completion time of coflows in datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1674–1687, Aug. 2018.
[10] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 443–454, 2014.
[11] Z. Qiu, C. Stein, and Y. Zhong, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proc. 27th ACM Symp. Parallelism Algorithms Archit.*, Jun. 2015, pp. 294–303.
[12] Y. Zhao *et al.*, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2015, pp. 424–432.
[13] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang, "On scheduling coflows," in *Proc. Int. Conf. Integer Program. Combinat. Optim.*, Springer, 2017, pp. 13–24.
[14] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 16–29.
[15] M. Chowdhury, S. Khuller, M. Purohit, S. Yang, and J. You, "Near optimal coflow scheduling in networks," in *Proc. 31st ACM Symp. Parallelism Algorithms Archit.*, 2019, pp. 123–134.
[16] S. Im, B. Moseley, K. Pruhs, and M. Purohit, "Matroid coflow scheduling," in *Proc. ICALP*, 2019, pp. 1–13.
[17] H. Jahanjou, E. Kantor, and R. Rajaraman, "Asymptotically optimal approximation algorithms for coflow scheduling," in *Proc. 29th ACM Symp. Parallelism Algorithms Archit.*, Jul. 2017, pp. 45–54.
[18] R. Mao and V. Aggarwal, "NPSCS: Non-preemptive stochastic coflow scheduling with time-indexed LP relaxation," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 2, pp. 2377–2387, Jun. 2021.
[19] X. S. Huang, X. S. Sun, and T. S. E. Ng, "Sunflow: Efficient optical circuit scheduling for coflows," in *Proc. 12th Int. Conf. Emerg. Netw. Experim. Technol.*, Dec. 2016, pp. 297–311.
[20] Y. Liu, W. Li, K. Li, H. Qi, X. Tao, and S. Chen, "Scheduling dependent coflows with guaranteed job completion time," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 2109–2115.
[21] M. Queyranne and A. S. Schulz, "Approximation bounds for a general class of precedence constrained parallel machine scheduling problems," *SIAM J. Comput.*, vol. 35, no. 5, pp. 1241–1253, Jan. 2006.
[22] S. Li, "Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations," *SIAM J. Comput.*, vol. 49, no. 4, pp. FOCS17-409–FOCS17-440, Jan. 2020.
[23] R. Grandl, S. Kandula, S. Rao, A. Akella, and J. Kulkarni, "GRAPHENE: Packing and dependency-aware scheduling for data-parallel clusters," in *Proc. 12th USENIX Symp. Oper. Syst. Design Implement. (OSDI)*, 2016, pp. 81–97.
[24] Y.-K. Kwok and I. Ahmad, "Static scheduling algorithms for allocating directed task graphs to multiprocessors," *ACM Comput. Surv.*, vol. 31, no. 4, pp. 406–471, 1999.
[25] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.*, vol. 17, no. 2, pp. 416–429, 1969.
[26] Y.-K. Kwok and I. Ahmad, "Dynamic critical-path scheduling: An effective technique for allocating task graphs to multiprocessors," *IEEE Trans. Parallel Distrib. Syst.*, vol. 7, no. 5, pp. 506–521, May 1996.
[27] E. G. Coffman and J. L. Bruno, *Computer and Job-Shop Scheduling Theory*. Hoboken, NJ, USA: Wiley, 1976.
[28] D. B. Shmoys, C. Stein, and J. Wein, "Improved approximation algorithms for shop scheduling problems," *SIAM J. Comput.*, vol. 23, no. 3, pp. 617–632, Jun. 1994.
[29] L. A. Goldberg, M. Paterson, A. Srinivasan, and E. Sweedyk, "Better approximation guarantees for job-shop scheduling," *SIAM J. Discrete Math.*, vol. 14, no. 1, pp. 67–92, Jan. 2001.
[30] J. P. Schmidt, A. Siegel, and A. Srinivasan, "Chernoff–Hoeffding bounds for applications with limited independence," *SIAM J. Discrete Math.*, vol. 8, no. 2, pp. 223–250, May 1995.
[31] T. Gonzalez and S. Sahni, "Flowshop and jobshop schedules: Complexity and approximation," *Oper. Res.*, vol. 26, no. 1, pp. 36–52, Feb. 1978.
[32] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, and D. B. Shmoys, "Sequencing and scheduling: Algorithms and complexity," *Handbooks Oper. Res. Manage. Sci.*, vol. 4, pp. 445–522, Jan. 1993.
[33] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.
[34] D. P. Williamson *et al.*, "Short shop schedules," *Oper. Res.*, vol. 45, no. 2, pp. 288–294, Apr. 1997.
[35] G. Birkhoff, "Tres observaciones sobre el algebra lineal," *Rev. Univ. Nac. Tucumán, A*, vol. 5, pp. 147–151, 1946.
[36] E. L. Lawler and J. Labetoulle, "On preemptive scheduling of unrelated parallel processors by linear programming," *J. ACM*, vol. 25, no. 4, pp. 612–619, Oct. 1978.
[37] D. E. Knuth, *The Art of Computer Programming*, vol. 3. London, U.K.: Pearson Education, 1997.
[38] P. Raghavan and C. D. Tompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, 1987.
[39] P. Raghavan, "Probabilistic construction of deterministic algorithms: Approximating packing integer programs," *J. Comput. Syst. Sci.*, vol. 37, no. 2, pp. 130–143, Oct. 1988.
[40] M. Shafiee and J. Ghaderi, "Scheduling coflows in datacenter networks: Improved bound for total weighted completion time," *ACM SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 1, pp. 29–30, Jun. 2017.
[41] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan, "Minimizing the sum of weighted completion times in a concurrent open shop," *Oper. Res. Lett.*, vol. 38, no. 5, pp. 390–395, Sep. 2010.
[42] M. B. Cohen, Y. T. Lee, and Z. Song, "Solving linear programs in the current matrix multiplication time," in *Proc. 51st Annu. ACM SIGACT Symp. Theory Comput.*, Jun. 2019, pp. 938–942.
[43] F. Le Gall, "Powers of tensors and fast matrix multiplication," in *Proc. 39th Int. Symp. Symbolic Algebr. Comput. (ISSAC)*, 2014, pp. 296–303.
[44] T. Leighton, B. Maggs, and S. Rao, "Universal packet routing algorithms," in *Proc. 29th Annu. Symp. Found. Comput. Sci.*, 1988, pp. 256–269.

Mehrnoosh Shafiee (Student Member, IEEE) received the B.Sc. degree from the Department of Electrical Engineering, Sharif University of Technology, in 2014, and the M.Sc. and Ph.D. degrees from the Department of Electrical Engineering, Columbia University, in 2015 and 2020, respectively. She is broadly interested in optimization and network algorithms. Her research is on the analysis and design of resource allocation algorithms for large-scale distributed systems.

Javad Ghaderi (Senior Member, IEEE) received the Ph.D. degree in electrical and computer engineering from the University of Illinois at Urbana-Champaign in 2013. He is currently an Associate Professor of electrical engineering with Columbia University. He spent a one-year Simons Post-Doctoral Fellowship at The University of Texas at Austin before joining Columbia University. His research interests include network algorithms, control, and optimization. He was a recipient of the Best Paper Award at ACM CoNEXT 2016, the NSF CAREER Award in 2017, the Best Student Paper Award at IFIP Performance 2020, and the Best Paper Award at IEEE INFOCOM 2020.