Scheduling Coflows for Minimizing the Total Weighted Completion Time in Heterogeneous Parallel Networks

Chi-Yeh Chen
Department of Computer Science and Information Engineering,
National Cheng Kung University,
Taiwan, ROC.
chency@csie.ncku.edu.tw.
July 15, 2022

Abstract
Coflow is a network abstraction used to represent communication patterns in data centers. The coflow scheduling problem in large data centers is one of the most important \(NP\)-hard problems. Many previous studies on coflow scheduling mainly focus on the single-core model. However, with the growth of data centers, this single-core model is no longer sufficient. This paper considers the coflow scheduling problem in heterogeneous parallel networks. The heterogeneous parallel network is an architecture based on multiple network cores running in parallel. In this paper, two polynomial-time approximation algorithms are developed for scheduling divisible and indivisible coflows in heterogeneous parallel networks, respectively. Considering the divisible coflow scheduling problem, the proposed algorithm achieve an approximation ratio of \(O(\log m/\log \log m)\) with arbitrary release times, where \(m\) is the number of network cores. On the other hand, when coflow is indivisible, the proposed algorithm achieve an approximation ratio of \(O\left(m (\log m/\log \log m)^2\right)\) with arbitrary release times.

Key words: Scheduling algorithms, approximation algorithms, coflow, datacenter network, heterogeneous parallel network.

1 Introduction

With the rapid development of cloud computing, large data centers have become the main computing infrastructure. In large data centers, structured traffic patterns of distributed applications have demonstrated the benefits of application-aware network scheduling [9, 7, 28, 1]. In addition, the success of data-parallel computing applications such as MapReduce [11], Hadoop [24, 1], Dryad [15] and Spark [27] has led to a proliferation of related applications [12, 8]. In data-parallel computing applications, computations are only processed locally on the machine. However, intermediate data (flows) generated during the computation stage need to be transmitted across different machines during the communication stage for further processing. Due to the large number of applications, the data center must have sufficient data transmission and scheduling capabilities. In data transfer for data-parallel computing applications, the interaction of all flows between two groups of machines becomes important. This collective communication pattern in the data center is abstracted by coflow traffic [6].

Many previous studies on coflow scheduling mainly focus on the single-core model. However, with the growth of data centers, this single-core model is no longer sufficient. In fact, a growing data
center will have legacy and new systems coexisting. To improve the efficiency of the network, there
will be different generations of network cores running in parallel [25, 14]. Therefore, we consider
an architecture based on multiple heterogeneous network cores running in parallel (heterogeneous
parallel network). The goal of this paper is to schedule coflows in the heterogeneous parallel
networks such that the total weighted completion time is minimized. This paper will discuss two
problems: the indivisible coflow scheduling problem and the divisible coflow scheduling problem.
In the indivisible coflow scheduling problem, flows in a coflow can only be arranged in the same
network core. However, in the divisible coflow scheduling problem, the flows in a coflow can be
arranged in different network cores.

1.1 Related Work

The coflow abstraction was first introduced by Chowdhury and Stoica [6] to capture communication
patterns in data centers. The coflow scheduling problem generalizes the well-studied concurrent
open shop scheduling problem, which has been shown to be strongly NP-hard [5, 13, 17, 19, 26].
Therefore, we instead look for efficient approximation algorithms rather than exact algorithms.
Since the concurrent open shop problem is NP-hard to approximate within a factor better than \(2 - \epsilon\)
for any \(\epsilon > 0\) [21, 22], the coflow scheduling problem is also NP-hard to approximate within a factor
better than \(2 - \epsilon\) [2, 3, 21]. Since the introduction of the coflow abstraction, many related investiga-
tions have been carried out to schedule coflows, e.g. [9, 17, 20, 29, 22, 2]. The first polynomial-time
deterministic approximation algorithm was developed by Qiu et al. [20]. Since then, a series of
improved approximation algorithms have been proposed [20, 16, 22, 2] and the best approximation
ratio achievable in polynomial time has improved from \(\frac{64}{3}\) to 4. Moreover, the best approximation
ratio has been improved from \(\frac{76}{3}\) to 5, taking into account arbitrary release times. When each job
has multiple coflows and there is a priority order between coflows, Shafiee and Ghaderi [23] proposed
a polynomial-time algorithm with approximation ratio of \(O(\mu \log(N) / \log(\log(N)))\), where \(\mu\) is the
maximum number of coflows in a job and \(N\) is the number of servers. In scheduling a single coflow
problem on a heterogeneous parallel network, Huang et al. [14] proposed an \(O(m)\)-approximation
algorithm, where \(m\) is the number of network cores.

1.2 Our Contributions

This paper considers the coflow scheduling problem in heterogeneous parallel networks. Our results
are as follows:

- In the indivisible coflow scheduling problem, we propose a \(O\left( m \log m / \log \log m \right)^2\)-approximation
  algorithm with arbitrary release times.

- In the divisible coflow scheduling problem, we also propose a \(O(\log m / \log \log m)\)-approximation
  algorithm with arbitrary release times.

1.3 Organization

The rest of this article is organized as follows. Section 2 introduces basic notations and preliminaries.
Section 3 presents an algorithm for indivisible coflow scheduling. Section 4 presents an algorithm
for divisible coflow scheduling. Section 5 compares the performance of the previous algorithms with
that of the proposed algorithm. Section 6 draws conclusions.
2 Notation and Preliminaries

Given a set of coflows $\mathcal{F}$ and a set of heterogeneous network cores $\mathcal{M}$, the coflow scheduling problem asks for a minimum total weighted completion time, where each coflow has its release time and its positive weight. The heterogeneous parallel networks can be abstracted as a set $\mathcal{M}$ of $m$ giant $N \times N$ non-blocking switchs, with $N$ input links connect to $N$ source servers and $N$ output links connect to $N$ destination servers. Each switch represents a network core. In each network core, all links are assumed to have the same capacity. Let $s_k$ be the link speed of network core $k$. Each source server or each destination server has $m$ simultaneous links connected to each network core. Let $I$ be the source server set and $J$ be the destination server set. The network core can be seen as a bipartite graph, with $I$ on one side and $J$ on the other side.

A coflow consists of a set of independent flows whose completion time is determined by the last completed flow in the set. We can use a $N \times N$ demand matrix $D^{(f)} = (d_{ijf})_{i,j=1}^N$ to represent the coflow $f \in \mathcal{F}$ where $d_{ijf}$ denote the size of the flow to be transferred from input $i$ to output $j$ in coflow $f$. We also can use a triple $(i,j,f)$ to represent a flow, where $i \in I$, $j \in J$ and $f \in \mathcal{F}$. For simplicity, we assume that all flows in a coflow arrive at the system at the same time (as shown in [20]). Let $w_f$, $r_f$ and $C_f$ denote the weight of coflow $f$, the release time of coflow $f$ and the completion time of coflow $f$, respectively. The goal is to minimize the total weighted completion time of the coflow $\sum_{f \in \mathcal{F}} w_f C_f$. We consider two problems: the indivisible coflow scheduling problem and the divisible coflow scheduling problem. In the indivisible coflow scheduling problem, flows in a coflow can only be arranged in the same network core. However, in the divisible coflow scheduling problem, the flows in a coflow can be arranged in different network cores.

3 Approximation Algorithm for Indivisible Coflow Scheduling

In this section, as in [15], we first give $O \left( m \left( \log m / \log \log m \right)^2 \right)$-approximation to minimize the makespan scheduling problem on heterogeneous network cores. We then convert the goal of minimizing the total weighted completion time to minimizing makespan at the loss of a constant factor. For every coflow $f$ and input port $i$, let $L_{if} = \sum_{j=1}^N d_{ijf}$ be the total amount of data that coflow $f$ needs to transmit through the input port $i$. Moreover, let $L_{jf} = \sum_{i=1}^N d_{ijf}$ be the total amount of data that coflow $f$ needs to transmit through the output port $j$. We can formulate our problem as the following linear programming relaxation.
\[ \begin{align*}
\text{min} & \quad E \\
\text{s.t.} & \quad \sum_{k \in M} x_{kf} = 1 \quad \forall f \in F \quad (1a) \\
& \quad \sum_{k \in M} \frac{x_{kf} L_{if}}{s_k} \leq C_f \quad \forall f \in F, \forall i \in I \quad (1b) \\
& \quad \sum_{k \in M} \frac{x_{kf} L_{jf}}{s_k} \leq C_f \quad \forall f \in F, \forall j \in J \quad (1c) \\
& \quad \frac{1}{s_k} \sum_{f \in F} x_{kf} L_{if} \leq E \quad \forall i \in I, \forall k \in M \quad (1d) \\
& \quad \frac{1}{s_k} \sum_{f \in F} x_{kf} L_{jf} \leq E \quad \forall j \in J, \forall k \in M \quad (1e) \\
& \quad C_f \leq E \quad \forall f \in F \quad (1f) \\
& \quad x_{kf}, C_f \geq 0 \quad \forall f \in F, \forall k \in M \quad (1g)
\end{align*} \]

In the LP (1), \( x_{kf} \) indicates whether coflow \( f \) is scheduled on network core \( k \). \( E \) is the makespan of the schedule and \( C_f \) is the completion time of coflow \( f \) in the schedule. The constraint (1a) requires scheduling for each coflow \( f \). The constraint (1b) (similarly the constraint (1c)) is that the completion time of \( f \) occurring on input port \( i \) (output port \( j \)) is at least the transfer time on the network cores allocated to it. The constraint (1d) (similarly the constraint (1e)) says that, for each network core \( k \) and input port \( i \) (output port \( j \)), the makespan \( E \) is at least the total transfer time that occurs on input port \( i \) (output port \( j \)) in all coflows assigned to \( k \). The constraint (1f) states that the makespan \( E \) is at least the completion time of any coflow \( f \). The constraint (1g) requires the \( x \) and \( C \) variables to be non-negative.

The following rounding method follows the method proposed by Li [18]. The optimal solution to LP (1) is the lower bound on the makespan of any valid schedule. We assume that \( m \) is large enough. Given an instance of the makespan scheduling problem, we will first preprocess the instance like [18] to contain only a small number of groups. The first stage of the pre-processing step discards all network cores that are at most \( 1/m \) times the speed of the fastest network core. Since there are \( m \) network cores, the total speed of discarded network cores is at most that of the fastest network core. That is, for the same amount of transferred data, the transfer time of the fastest network core is at most the transfer time of using all discarded network cores; however, this will increase the makespan by a factor of 2. So we can move the \( x \)-values of all discarded network cores to the \( x \)-values of the fastest network cores. Therefore, we can assume that all network cores are faster than \( 1/m \) times the speed of the fastest network core. We also can normalize the speed of all network cores to \([1, m]\).

The second stage of the pre-processing step divides the network cores into groups, each group containing network cores of similar speed. Let \( \gamma = \log m / \log \log m \). The network cores are divided into \( K \) groups \( M_1, M_2, \ldots, M_K \), where \( M_k \) contains network cores with speed in \([\gamma^{k-1}, \gamma^k)\) and \( K = \lceil \log_{\gamma} m \rceil = O(\log m / \log \log m) \). For a subset \( M_k \subseteq M \) of network cores, let

\[ s(M_k) = \sum_{u \in M_k} s_u \]
be the total speed of network cores in $M_k$. For $M_k \subseteq M$ and $f \in \mathcal{F}$, let
\[
x_{M_kf} = \sum_{u \in M_k} x_{uf}
\]
be the total fraction of coflow $f$ assigned to network cores in $M_k$. For any coflow $f$, let $\ell_f$ be the largest integer $\ell$ such that $\sum_{k=\ell}^{K} x_{M_kf} \geq 1/2$. That is, the largest group index $\ell$ such that at least $1/2$ of $f$ is assigned to network cores in groups $M_{\ell}, \ldots, M_{K}$. Then, let $r(f)$ be the index $k \in [\ell_f, K]$ that maximizes $s(M_k)$. That is, $r(f)$ is the index of the group with the highest total speed among the groups $\ell_f$ to $K$. Using the $r(f)$ value, we can run the list algorithm (Algorithm 1) to get the index of the allocated network core. The algorithm is to find the least loaded network core and assign coflow to it. The proposed algorithm has the following lemmas.

Algorithm 1 coflow-makespan-list-scheduling

Require: two vectors $\bar{C} \in \mathbb{R}_{\geq 0}^n$ and $\bar{x} \in \mathbb{R}_{\geq 0}^{n \times m}$
1: let $load_I(i, h)$ be the load on the $i$-th input port of the network core $h$
2: let $load_O(j, h)$ be the load on the $j$-th output port of the network core $h$
3: let $\mathcal{A}_h$ be the set of coflows allocated to network core $h$
4: both $load_I$ and $load_O$ are initialized to zero and $\mathcal{A}_h = \emptyset$ for all $h \in [1, m]$
5: for every coflow $f \in \mathcal{F}$ in non-decreasing order of $\bar{C}_f$, breaking ties arbitrarily do
6: $x_{M_kf} = \sum_{u \in M_k} x_{uf}$ for all $k = 1, \ldots, K$
7: $\ell_f = \max_{\ell \in [1, K]} \ell$ s.t. $\sum_{k=\ell}^{K} x_{M_kf} \geq 1/2$
8: $r(f) = \arg \max_{k \in [\ell_f, K]} s(M_k)$
9: $h^* = \arg \min_{h \in M_{r(f)}} \frac{1}{s_h} \left( \max_{i,j \in [1, N]} load_I(i, h) + load_O(j, h) + L_{if} + L_{jf} \right)$
10: $\mathcal{A}_{h^*} = \mathcal{A}_{h^*} \cup \{f\}$
11: $load_I(i, h^*) = load_I(i, h^*) + L_{if}$ and $load_O(j, h^*) = load_O(j, h^*) + L_{jf}$ for all $i,j \in [1, N]$
12: end for

Lemma 3.1. For any port $i \in \mathcal{I}$, we have $\sum_{f \in \mathcal{F}} \frac{L_{if}}{s(M_{r(f)})} \leq 2KE$.

Proof. Since $\sum_{k=\ell_f}^{K} x_{M_kf} \geq 1/2$ for any coflow $f$, we have
\[
\sum_{k=1}^{K} \frac{x_{M_kf}}{s(M_k)} \geq \sum_{k=\ell_f}^{K} \frac{x_{M_kf}}{s(M_k)} \geq \frac{1}{2s(M_k)},
\]
According to the above inequality, we have
\[
\sum_{f \in \mathcal{F}} \frac{L_{if}}{s(M_{r(f)})} \leq 2 \sum_{f \in \mathcal{F}} L_{if} \sum_{k=1}^{K} \frac{x_{M_kf}}{s(M_k)} = 2 \sum_{k=1}^{K} \frac{1}{s(M_k)} \sum_{f \in \mathcal{F}} L_{if} x_{M_kf} \leq 2 \sum_{k=1}^{K} E = 2KE.
\]
The last inequality is due to constraint [1d]. Since $\sum_{f \in \mathcal{F}} x_{kf} L_{if} \leq s_k E$ for every $i \in M_k$, we have $\sum_{f \in \mathcal{F}} L_{if} x_{M_kf} \leq s(M_k)E$.\hfill $\square$
Lemma 3.2. For any port \( j \in J \), we have \( \sum_{f \in F} L_{ijf} \leq 2KE \).

Proof. The proof is similar to that of lemma 3.1.

Lemma 3.3. Let \( \bar{E} \) be an optimal solution to the linear program \([2]\), and let \( \tilde{E} \) denote the makespan in the schedule found by coflow-makespan-list-scheduling. We have \( \tilde{E} \leq 4m\gamma K\bar{E} \).

Proof. Assume the last completed flow in a coflow \( f \) is sent via link \((i, j)\) of network core \( k \). We have

\[
\tilde{E} \leq \sum_{f \in F} \frac{L_{ijf} + L_{jjf}}{s_k} \leq m\gamma \sum_{f \in F} \frac{L_{ijf} + L_{jjf}}{s(M_{r(f)})} \leq 4m\gamma K\bar{E}
\]

The inequality (2) is due to the worst case where all coflows are assigned to the same core. The inequality (4) is due to \( \frac{s(M_{r(f)})}{s_k} \leq m\gamma \). The inequality (5) is based on lemma 3.1 and lemma 3.2.

According to lemma 3.3, we have the following theorem:

Theorem 3.4. When the speed of network core is between one and \( m \), the indivisible coflow schedule has makespan at most \( 4m\gamma K\bar{E} = O \left( m (\log m / \log \log m)^2 \right) \bar{E} \).

Due to the discarded network cores, we have the following theorem:

Theorem 3.5. When the speed of network core is arbitrary, the indivisible coflow schedule has makespan at most \( 8m\gamma K\bar{E} = O \left( m (\log m / \log \log m)^2 \right) \bar{E} \).

3.1 An extension for Total Weighted Completion Time

This section gives an \( O \left( m (\log m / \log \log m)^2 \right) \) -approximation algorithm for minimizing total weighted completion time, which is based on combining our algorithm for minimizing makespan. Without loss of generality, we assume that \( L_{ijf}/s_k \geq 1 \) and \( L_{jjf}/s_k \geq 1 \) for all \( f \in F, k \in M, i \in I \) and \( j \in J \). Let \( s_{min} = \min_{k \in M} s_k \). We have

\[
L = \log \left( \max_{f \in F} r_f + \frac{1}{s_{min}} \max_{i \in I} \left\{ \max_{f \in F} \sum_{j \in J} L_{ijf}, \max_{f \in F} \sum_{j \in J} L_{jjf} \right\} \right).
\]

First, we divide the time horizon into increasing time intervals: \([1, 2], (2, 4], (4, 8], \ldots, (2^{L-1}, 2^L]\). Let \( \tau_l = 2^l \) where \( l = 0, 1, \ldots, L \). We can formulate our problem as the following linear programming relaxation.
\[
\begin{align*}
\text{min} & \quad \sum_{f \in F} w_f C_f & \quad \text{(6)} \\
\text{s.t.} & \quad \sum_{k \in M} \sum_{l=1}^{L} x_{kfl} = 1, & \forall f \in F & \quad \text{(6a)} \\
& \quad \sum_{k \in M} \frac{L_{ijf}}{s_k} \sum_{l=1}^{L} x_{kfl} \leq C_f - r_f & \forall f \in F, \forall i \in I & \quad \text{(6b)} \\
& \quad \sum_{k \in M} \frac{L_{jlf}}{s_k} \sum_{l=1}^{L} x_{kfl} \leq C_f - r_f & \forall f \in F, \forall j \in J & \quad \text{(6c)} \\
& \quad \frac{1}{s_k} \sum_{l=1}^{L} \sum_{f \in F} x_{kfu} L_{if} \leq \tau_l & \forall i \in I, \forall k \in M, \forall l \in [1, L] & \quad \text{(6d)} \\
& \quad \frac{1}{s_k} \sum_{l=1}^{L} \sum_{f \in F} x_{kfu} L_{jlf} \leq \tau_l & \forall j \in J, \forall k \in M, \forall l \in [1, L] & \quad \text{(6e)} \\
& \quad \sum_{l=1}^{\tau_{l-1}} \sum_{k \in M} x_{kfl} \leq C_f, & \forall f \in F & \quad \text{(6f)} \\
& \quad x_{kfl}, C_f \geq 0 & \forall f \in F, \forall k \in M & \quad \text{(6g)}
\end{align*}
\]

In the LP (6), \( x_{kfl} \) indicates whether or not coflow \( f \) completes on network core \( k \) in the \( l \)-th interval (from \( 2^{l-1} \) to \( 2^l \)) and \( C_f \) is the completion time of coflow \( f \) in the schedule. The constraint (6a) requires all the coflows must be assigned to run at some network core. The constraint (6b) (similarly the constraint (6c)) is that the time required to transmit coflow \( f \) on input port \( i \) (output port \( j \)) cannot exceed the time period between its release and completion time. The constraints (6d) and (6e) represent capacity limits to time \( \tau_l \). Since \( \tau_{l-1} \) is the lower bound on the completion time of coflows completed within the interval \( l \), the constraint (6f) is a lower bound to the completion time of coflow. The constraint (6g) requires the \( x \) and \( C \) variables to be non-negative.

Our algorithm coflow-driven-list-scheduling (described in Algorithm 2) is as follows. Given a set of coflow \( F \), an optimal solution \( \bar{C} \) and \( \bar{x} \) can be obtained by the linear program (6). Lines 1-5 schedule all coflow into time intervals and normalize the value of \( \bar{x} \). Lines 6-19 are the coflow-makespan-list-scheduling algorithm, which schedules the coflows in the corresponding time interval to each network core. Lines 20-36 transmit all coflow, which is modified from Shafiee and Ghaderi’s algorithm [22].

The following schedule method follows the method proposed by Chudak and Shmoys [10]. For any coflow \( f \in F \), let \( q(f) \) be the the minimum value of \( q \) such that both \( \sum_{l=1}^{q(f)} \sum_{k \in M} x_{kfl} \geq 1/2 \) and \( C_f \leq 2q(f) \) are satisfied. We set \( \mathcal{F}_l = \{f|q(f) = l\} \) and construct a schedule for each subset \( \mathcal{F}_l \) respectively. Let \( \alpha_f \) be the total fraction of coflow \( f \) over all network cores in the first \( q(f) \) intervals with respect to solution \( \bar{x} \):

\[
\alpha_f = \sum_{l=1}^{q(f)} \sum_{k \in M} \bar{x}_{kfl}.
\]
Algorithm 2  
coflow-driven-list-scheduling

Require: two vectors $C \in \mathbb{R}^n_{\geq 0}$ and $x \in \mathbb{R}^{n \times m}_{\geq 0}$

1: for every coflow $f \in F$ do
2: \quad $q(f) = \min_{l \in [1, L]} \sum_{k \in M} x_{kfl} \geq 1/2$ and $C_f \leq 2q(f)$
3: \quad $F_q(f) = F(q(f)) \cup \{f\}$
4: \quad $\alpha_f = \sum_{l=1}^{q(f)} \sum_{k \in M} x_{kfl}$ and $\bar{x}_{kf} = \sum_{l=1}^{q(f)} \frac{x_{kfl}}{\alpha_f}$ for all $k \in M$
5: end for
6: for $l \in [1, L]$ do
7: \quad let load$_I(i, h)$ be the load on the $i$-th input port of the network core $h$
8: \quad let load$_O(j, h)$ be the load on the $j$-th output port of the network core $h$
9: \quad let $A_{lh}$ be the set of coflows allocated to network core $h$ in the $l$-th interval
10: \quad both load$_I$ and load$_O$ are initialized to zero and $A_{lh} = \emptyset$ for all $h \in [1, m]$
11: \quad for every coflow $f \in F_l$ in non-decreasing order of $C_f$, breaking ties arbitrarily do
12: \quad $\bar{x}_{Mk} = \sum_{u \in M_k} \bar{x}_{uf}$ for all $k = 1, \ldots, K$
13: \quad $\ell_f = \max_{\ell \in [1, K]} \ell$ s.t. $\sum_{k=\ell}^{K} \bar{x}_{Mk} \geq 1/2$
14: \quad $r(f) = \arg\max_{M_k: \ell_f \leq k \leq K} s_k(M_k)$
15: \quad $h^* = \arg\min_{h \in M_r(f)} \frac{1}{s_h} \left(\max_{i,j \in [1, N]} \text{load}_I(i, h) + \text{load}_O(j, h) + \text{L}_if + \text{L}_jf\right)$
16: \quad $A_{lh^*} = A_{lh^*} \cup \{f\}$
17: \quad $\text{load}_I(i, h^*) = \text{load}_I(i, h^*) + \text{L}_if$ and $\text{load}_O(j, h^*) = \text{load}_O(j, h^*) + \text{L}_jf$ for all $i, j \in [1, N]$
18: end for
19: end for
20: for each $k \in M$ do in parallel do
21: \quad wait until the first coflow is released
22: \quad while there is some incomplete flow do
23: \quad \quad for all $f \in A_{lk}$, list the released and incomplete flows respecting the non-decreasing order in $C_f$
24: \quad \quad let $L$ be the set of flows in the list
25: \quad \quad for every flow $(i, j, f) \in L$ do
26: \quad \quad \quad if the link $(i, j)$ is idle then
27: \quad \quad \quad \quad schedule flow $f$
28: \quad \quad \quad end if
29: \quad \quad end for
30: \quad end while
31: end for
32: end for
33: while no new flow is completed or released do
34: \quad transmit the flows that get scheduled in line 28 at maximum rate $s_k$.
35: end while
36: end for
We set a feasible solution \( \tilde{x} \) from the optimal solution \( \bar{x} \):

\[
\tilde{x}_{kf} = \sum_{l=1}^{q(f)} \frac{\bar{x}_{kfl}}{\alpha_f}
\]

for all \( f \in F \) and \( k \in M \).

Fix some \( l = 1, \ldots, L \), and consider the coflows in \( F_l \). We can construct a scheduling fragment for \( F_l \) of length \( \bar{R}q^{2l+1} \), where \( \bar{R} \) is the performance guarantee of the proposed approximation algorithm for the makespan objective. This fragment shall be run from time \( \bar{R}(1 + 2 + \cdots + 2^l) \) to \( \bar{R}(1 + 2 + \cdots + 2^{l+1}) \). Therefore, each coflow \( f \in F_l \) completes at most \( 4 \bar{R}q^{2l} \) before. Since \( l \) is the minimum value for \( \bar{C}_f \leq 2^l \), we have \( 2^l \leq 2 \bar{C}_f \). Since \( \sum_{u=1}^{L} \sum_{k \in M} x_{kfu} \geq 1/2 \), we have

\[
\tau_{l-1/2} \leq \tau_{l-1} \left( \sum_{u=1}^{L} \sum_{k \in M} x_{kfu} \right)
\]

\[
\leq \sum_{u=1}^{L} \sum_{k \in M} \tau_{u-1} x_{kfu}
\]

\[
\leq \sum_{u=1}^{L} \sum_{k \in M} \tau_{u-1} x_{kfu}
\]

\[
\leq \bar{C}_f.
\]

We have \( 2^l \leq 4 \bar{C}_f \) and proved the following theorem.

**Theorem 3.6.** The coflow-driven-list-scheduling has an approximation ratio of, at most, \( \frac{64m \gamma K}{O \left( m (\log m / \log \log m)^2 \right)} \).

### 4 Approximation Algorithm for Divisible Coflow Scheduling

This section considers the divisible coflow scheduling problem. The method is similar to schedule indivisible coflow, the difference is that it is scheduled at the flow level. First, we consider the minimizing makespan problem. We can formulate our problem as the following linear programming relaxation.

\[
\begin{align*}
\text{min} & \quad E \\
\text{s.t.} & \quad \sum_{k \in M} x_{kijf} = 1, \quad \forall f \in F, \forall i \in I, \forall j \in J \\
& \quad \sum_{k \in M} d_{ijf} x_{kijf} \leq C_{ijf}, \quad \forall f \in F, \forall i \in I, \forall j \in J \\
& \quad \frac{1}{s_k} \sum_{f \in F} \sum_{j \in J} d_{ijf} x_{kijf} \leq E, \quad \forall i \in I, \forall k \in M \\
& \quad \frac{1}{s_k} \sum_{f \in F} \sum_{i \in I} d_{ijf} x_{kijf} \leq E, \quad \forall j \in J, \forall k \in M \\
& \quad C_{ijf} \leq E, \quad \forall f \in F, \forall i \in I, \forall j \in J \\
& \quad x_{kijf}, C_{ijf} \geq 0, \quad \forall f \in F, \forall i \in I, \forall j \in J, \forall k \in M
\end{align*}
\]
In the LP (9), \( x_{kijf} \) indicates whether flow \((i,j,f)\) is scheduled on network core \(k\), \(E\) is the makespan of the schedule and \(C_{ijf}\) is the completion time of \((i,j,f)\) in the schedule. The constraint (9a) requires scheduling for each flow \((i,j,f)\). The constraint (9b) is that the completion time of \((i,j,f)\) occurring on link \((i,j)\) is at least the transfer time on the network cores allocated to it. The constraint (9c) (similarly the constraint (9d)) says that, for each network core \(k\) and input port \(i\) (output port \(j\)), the makespan \(E\) is at least the total transfer time that occurs on input port \(i\) (output port \(j\)) in all flows assigned to \(k\). The constraint (9e) states that the makespan \(E\) is at least the completion time of any flow \((i,j,f)\). The constraint (9f) requires the \(x\) and \(C\) variables to be non-negative.

Following the steps in Section 3, we divide the network cores into groups. We have \( \gamma = \log m / \log \log m \) and \(K\) groups \(M_1, M_2, \ldots, M_K\), where \(M_k\) contains network cores with speed in \([\gamma^{k-1}, \gamma^k)\). For a subset \(M_k \subseteq M\) of network cores, we also have

\[
s(M_k) = \sum_{u \in M_k} s_u.
\]

For \(M_k \subseteq M\), let

\[
x_{M_kijf} = \sum_{u \in M_k} x_{uijf}
\]

be the total fraction of flow \((i,j,f)\) assigned to network cores in \(M_k\). For any flow \((i,j,f)\), let \(\ell_{ijf}\) be the largest integer \(\ell\) such that \(\sum_{k=\ell}^{K} x_{M_kijf} \geq 1/2\). Then, let \(r(i,j,f)\) be the index \(k \in [\ell_{ijf}, K]\) that maximizes \(s(M_k)\). Using the \(r(i,j,f)\) value, we can run the list algorithm (Algorithm 3) to get the index of the allocated network core. The algorithm is to find the least loaded network core and assign flow to it. The proposed algorithm has the following lemmas.

**Algorithm 3** flow-makespan-list-scheduling

**Require:** two vectors \(\vec{C} \in \mathbb{R}^n_{\geq 0}\) and \(\vec{x} \in \mathbb{R}^{n \times m}\)

1: let \(\text{load}_I(i, h)\) be the load on the \(i\)-th input port of the network core \(h\)
2: let \(\text{load}_O(j, h)\) be the load on the \(j\)-th output port of the network core \(h\)
3: let \(\mathcal{A}_h\) be the set of coflows allocated to network core \(h\)
4: both \(\text{load}_I\) and \(\text{load}_O\) are initialized to zero and \(\mathcal{A}_h = \emptyset\) for all \(h \in [1, m]\)
5: for every flow \((i, j, f)\) in non-decreasing order of \(\vec{C}_{ijf}\), breaking ties arbitrarily do
6: \(x_{M_kijf} = \sum_{u \in M_k} x_{uijf}\) for all \(k = 1, \ldots, K\)
7: \(\ell_{ijf} = \max_{\ell \in [1, K]} \ell \) s.t. \(\sum_{k=\ell}^{K} x_{M_kijf} \geq 1/2\)
8: \(r(i, j, f) = \arg \max_{M_k: x_{M_kijf} \leq K} \max s(M_k)\)
9: \(h^* = \arg \min_{h \in M_{r(i, j, f)}} \frac{1}{s_h} (\text{load}_I(i, h) + \text{load}_O(j, h))\)
10: \(\mathcal{A}_{h^*} = \mathcal{A}_{h^*} \cup \{(i, j, f)\}\)
11: \(\text{load}_I(i, h^*) = \text{load}_I(i, h^*) + d_{ijf}\) and \(\text{load}_O(j, h^*) = \text{load}_O(j, h^*) + d_{ijf}\)
12: end for

**Lemma 4.1.** For every flow \((i, j, f)\), and any network core \(k \in M_{\ell_{ijf}}\), we have \(d_{ijf} \leq 2\gamma \sum_{k' \in M} \frac{x_{k'ijf} \ell_{k'ijf}}{s_k}\).

**Proof.** Since \(\sum_{k=\ell_{ijf}+1}^{K} x_{M_kijf} < 1/2\), we have \(\sum_{k=1}^{\ell_{ijf}} x_{M_kijf} > 1/2\). Moreover, since \(\sum_{k' \in \cup_{k=1}^{\ell_{ijf}} M_k} x_{k'ijf} \geq 1/2\) and \(\frac{1}{s_k} \geq \gamma^{-\ell_{ijf}}\) for every \(k' \in \cup_{k=1}^{\ell_{ijf}} M_k\), we also have \(\sum_{k' \in \cup_{k=1}^{\ell_{ijf}} M_k} \frac{x_{k'ijf}}{s_k} \geq \gamma^{-\ell_{ijf}}/2\).
Since $k$ is in group $r(i, j, f) \geq \ell_{ijf}$, $k$ has speed at least $\gamma^{\ell_{ijf} - 1}$ and thus $\frac{1}{s_k} \geq \gamma^{1 - \ell_{ijf}}$. Therefore, we have
\[
\frac{d_{ijf}}{s_k} \leq d_{ijf}\gamma^{1 - \ell_{ijf}} \leq 2\gamma \sum_{k' \in M} \frac{d_{ijf} x_{k'ijf}}{s_k}.
\]

Lemma 4.2. For any port $i \in I$, we have $\sum_{f \in F} \frac{L_{ijf}}{s(Mr(i, j, f))} \leq 2KE$.

Proof. Since $\sum_{k=\ell_{ijf}}^K x_{Mkijf} \geq \frac{1}{2}$ for any flow $(i, j, f)$, we have
\[
\sum_{k=1}^K \frac{x_{Mkijf}}{s(M_k)} \geq \sum_{k=\ell_{ijf}}^K \frac{x_{Mkijf}}{s(M_k)} \geq \frac{1}{2s(M_k)}.
\]
According to the above inequality, we have
\[
\sum_{f \in F} \frac{L_{ijf}}{s(Mr(i, j, f))} = \sum_{f \in F} \sum_{j \in J} \frac{d_{ijf}}{s(Mr(i, j, f))} 
\leq 2 \sum_{f \in F} \sum_{j \in J} d_{ijf} \sum_{k=1}^K \frac{x_{Mkijf}}{s(M_k)} 
\leq 2 \sum_{k=1}^K \frac{1}{s(M_k)} \sum_{f \in F} \sum_{j \in J} d_{ijf} x_{Mkijf} 
\leq 2 \sum_{k=1}^K E 
\leq 2KE.
\]
The last inequality is due to constraint (9c). Since $\sum_{f \in F} \sum_{j \in J} d_{ijf} x_{Mkijf} \leq s_kE$ for every $i \in M_k$, we have $\sum_{f \in F} \sum_{j \in J} d_{ijf} x_{Mkijf} \leq s(M_k)E$.

Lemma 4.3. For any port $j \in J$, we have $\sum_{f \in F} \frac{L_{ijf}}{s(Mr(i, j, f))} \leq 2KE$.

Proof. The proof is similar to that of lemma 4.2.

Lemma 4.4. Let $\hat{E}$ be an optimal solution to the linear program (9), and let $\tilde{E}$ denote the makespan in the schedule found by flow-makespan-list-scheduling. We have $\tilde{E} \leq (4K + 2\gamma)\hat{E}$.
Proof. Assume the last completed flow \((i,j,f)\) is sent via the network core \(k\). We have
\[
\tilde{E} \leq \sum_{f \in \mathcal{F}} \frac{L_{i,f} + L_{j,f}}{s(M_{r(f)})} + \frac{d_{ij,f}}{s_k} \tag{10}
\]
\[
\leq 4K \tilde{E} + \frac{d_{ij,f}}{s_k} \tag{11}
\]
\[
\leq 4K \tilde{E} + 2\gamma \sum_{k' \in M} \frac{d_{ij,f} x_{k'ijf}}{s_k} \tag{12}
\]
\[
\leq 4K \tilde{E} + 2\gamma \tilde{E} \tag{13}
\]
\[
= (4K + 2\gamma) \tilde{E}. \tag{14}
\]

The first term of inequality (10) is due to all links \((i,j)\) in the network cores are busy from zero to the start of the last completed flow \((i,j,f)\). The inequality (11) is based on lemma 4.2 and lemma 4.3. The inequality (12) is based on lemma 4.1. The inequality (13) is obtained by constraints (9b) and (9e) in the linear program (9).

According to lemma 4.4, we have the following theorem:

**Theorem 4.5.** When the speed of network core is between one and \(m\), the divisible coflow schedule has makespan at most \((4K + 2\gamma) \tilde{E} = O(\log m / \log \log m) \tilde{E}\).

Due to the discarded network cores, we have the following theorem:

**Theorem 4.6.** When the speed of network core is arbitrary, the divisible coflow schedule has makespan at most \((8K + 4\gamma) \tilde{E} = O(\log m / \log \log m) \tilde{E}\).

### 4.1 An extension for Total Weighted Completion Time

This section gives an \(O(\log m / \log \log m)\)-approximation algorithm for minimizing total weighted completion time, which is based on combining our algorithm for minimizing makespan. The method is the same as Section 3.1. We can formulate our problem as the following linear programming relaxation.
Algorithm 4 flow-driven-list-scheduling

Require: two vectors $\bar{C} \in \mathbb{R}^n_{\geq 0}$ and $\bar{x} \in \mathbb{R}^{n \times m}_{\geq 0}$

1: for every coflow $f \in F$ do
2: \[ q(i, j, f) = \min_{l \in [1, L]} \text{s.t. } \sum_{k=1}^{L} \sum_{m \in \mathcal{M}} x_{kijfl} \geq 1/2 \text{ and } \bar{C}_{ijf} \leq 2q(f) \]
3: \[ F_{q(i, j, f)} = F_{q(i, j, f)} \cup \{(i, j, f)\} \]
4: \[ \alpha_{ijf} = \sum_{l=1}^{q(i, j, f)} \sum_{m \in \mathcal{M}} \bar{x}_{kijfl} \text{ and } \bar{x}_{kijf} = \sum_{l=1}^{q(i, j, f)} \frac{x_{kijfl}}{\alpha_{ijf}} \text{ for all } k \in \mathcal{M} \]
5: end for
6: for $l \in [1, L]$ do
7: \hspace{1em} let $\text{load}_I(i, h)$ be the load on the $i$-th input port of the network core $h$
8: \hspace{1em} let $\text{load}_O(j, h)$ be the load on the $j$-th output port of the network core $h$
9: \hspace{1em} let $\mathcal{A}_h$ be the set of coflows allocated to network core $h$ in the $l$-th interval
10: \hspace{1em} both $\text{load}_I$ and $\text{load}_O$ are initialized to zero and $\mathcal{A}_h = \emptyset$ for all $h \in [1, m]$
11: \hspace{1em} for every flow $(i, j, f)$ in non-decreasing order of $\bar{C}_f$, breaking ties arbitrarily do
12: \hspace{2em} $\bar{x}_{Mkijf} = \sum_{u \in \mathcal{M}_k} \bar{x}_{uijf}$ for all $k = 1, \ldots, K$
13: \hspace{2em} $\ell_{ijf} = \max_{l \in [1, K]} \ell$ s.t. $\sum_{k=1}^{K} \bar{x}_{Mkijf} \geq 1/2$
14: \hspace{2em} $r(i, j, f) = \arg \max_{M_{k, z_{ijf}} \leq k \leq K} s(M_k)$
15: \hspace{2em} $h^* = \arg \min_{h \in \mathcal{M}_{l(i, j, f)}} \frac{1}{\bar{C}_{iijf}} (\text{load}_I(i, h) + \text{load}_O(j, h))$
16: \hspace{2em} $\mathcal{A}_{ih^*} = \mathcal{A}_{ih^*} \cup \{(i, j, f)\}$
17: \hspace{2em} $\text{load}_I(i, h^*) = \text{load}_I(i, h^*) + d_{ijf}$ and $\text{load}_O(j, h^*) = \text{load}_O(j, h^*) + d_{ijf}$
18: end for
19: end for
20: for each $k \in \mathcal{M}$ do in parallel do
21: \hspace{1em} for $l \in [1, L]$ do
22: \hspace{2em} wait until the first coflow is released
23: \hspace{2em} while there is some incomplete flow do
24: \hspace{3em} for all $(i, j, f) \in \mathcal{A}_{ih}$, list the released and incomplete flows respecting the non-decreasing order in $\bar{C}_{ijf}$
25: \hspace{3em} let $L$ be the set of flows in the list
26: \hspace{3em} for every flow $(i, j, f) \in L$ do
27: \hspace{4em} if the link $(i, j)$ is idle then
28: \hspace{5em} schedule flow $f$
29: \hspace{4em} end if
30: \hspace{3em} end for
31: \hspace{2em} while no new flow is completed or released do
32: \hspace{3em} transmit the flows that get scheduled in line 28 at maximum rate $s_k$.
33: \hspace{2em} end while
34: \hspace{1em} end while
35: end for
36: end for
\[
\min_{f \in F} \sum_{f \in F} w_f C_f
\]  
\text{s.t..}  
\sum_{k \in M} \sum_{l=1}^{L} x_{kijfl} = 1 \quad \forall f \in F, \forall i \in I, \forall j \in J \quad (15a)

\sum_{k \in M} \sum_{l=1}^{L} \frac{d_{ijf}}{s_k} x_{kijfl} \leq C_{ijf} - r_f \quad \forall f \in F, \forall i \in I, \forall j \in J \quad (15b)

\frac{1}{s_k} \sum_{u=1}^{l} \sum_{f \in F} \sum_{j \in J} d_{ijf} x_{kijfu} \leq \tau_l \quad \forall i \in I, \forall k \in M, \forall l \in [1, L] \quad (15c)

\frac{1}{s_k} \sum_{u=1}^{l} \sum_{f \in F} \sum_{i \in I} d_{ijf} x_{kijfu} \leq \tau_l \quad \forall j \in J, \forall k \in M, \forall l \in [1, L] \quad (15d)

\sum_{l=1}^{L} \sum_{k \in M} \tau_{l-1} x_{kijfl} \leq C_{ijf} \quad \forall f \in F, \forall i \in I, \forall j \in J \quad (15e)

C_{ijf} \leq C_f \quad \forall f \in F, \forall i \in I, \forall j \in J \quad (15f)

x_{kijfl}, C_{ijf}, C_f \geq 0 \quad \forall f \in F, \forall i \in I, \forall j \in J, \forall k \in M \quad (15g)

In the LP (15), \( x_{kijfl} \) indicates whether or not flow \((i, j, f)\) completes on network core \(k\) in the \(l\)-th interval, \(C_{ijf}\) is the completion time of flow \((i, j, f)\) and \(C_f\) is the completion time of coflow \(f\). The constraint (15a) requires all the flows must be assigned to run at some network core. The constraint (15b) is that the time required to transmit flow \((i, j, f)\) on link \((i, j)\) cannot exceed the time period between its release and completion time. The constraints (15c) and (15d) represent capacity limits to time \(\tau_l\). The constraint (15e) is a lower bound to the completion time of flow. The constraint (15f) ensures that the completion time of coflow \(f\) is bounded by all its flows. The constraint (15g) requires the \(x\) and \(C\) variables to be non-negative.

Our algorithm flow-driven-list-scheduling (described in Algorithm 4) is as follows. Given a set of coflow \(F\), an optimal solution \(\bar{C}\) and \(\bar{x}\) can be obtained by the linear program (15). Lines 1-5 schedule all flow into time intervals and normalize the value of \(\bar{x}\). Lines 6-19 are the flow-makespan-list-scheduling algorithm, which schedules the flows in the corresponding time interval to each network core. Lines 20-36 transmit all coflow. Same as Section 3.1, we have the following theorem.

**Theorem 4.7.** The flow-driven-list-scheduling has an approximation ratio of, at most, \((64K + 32\gamma) = O(\log m/\log \log m)\).

### 5 Results and Discussion

This section compares the approximation ratio of the proposed algorithm to that of the previous algorithm. We compares with the algorithm of Huang et al. [14], which schedules a single coflow on a heterogeneous parallel network. In the scheduling single divisible coflow problem, our algorithm achieves an approximation ratio of \((4K + 2\gamma)\) where \(\gamma = \log m/\log \log m\) and \(K = \lceil \log_\gamma m \rceil\). Figure 4 presents the numerical results concerning the approximation ratio of algorithms. When \(m \geq 25\), the proposed algorithm outperforms the algorithm in [14].
6 Concluding Remarks

With the growth of data centers, the scheduling of the single-core model is no longer sufficient. Therefore, we consider scheduling coflow problems in heterogeneous parallel networks. In this paper, two polynomial-time approximation algorithms are developed for scheduling divisible and indivisible coflows in heterogeneous parallel networks, respectively. Considering the divisible coflow scheduling problem, the proposed algorithm achieve an approximation ratio of $O(\log m / \log \log m)$ with arbitrary release times, where $m$ is the number of network cores. On the other hand, when coflow is indivisible, the proposed algorithm achieve an approximation ratio of $O\left( m (\log m / \log \log m)^2 \right)$ with arbitrary release times.

References

[1] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, “Sincronia: Near-optimal network design for coflows,” in Proceedings of the 2018 ACM Conference on SIGCOMM, ser. SIGCOMM ’18. New York, NY, USA: Association for Computing Machinery, 2018, p. 16–29.

[2] S. Ahmadi, S. Khuller, M. Purohit, and S. Yang, “On scheduling coflows,” Algorithmica, vol. 82, no. 12, pp. 3604–3629, 2020.

[3] N. Bansal and S. Khot, “Inapproximability of hypergraph vertex cover and applications to scheduling problems,” in Automata, Languages and Programming, S. Abramsky, C. Gavoille, C. Kirchner, F. Meyer auf der Heide, and P. G. Spirakis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 250–261.

[4] D. Borthakur, “The hadoop distributed file system: Architecture and design,” Hadoop Project Website, vol. 11, no. 2007, p. 21, 2007.
[5] Z.-L. Chen and N. G. Hall, “Supply chain scheduling: Conflict and cooperation in assembly systems,” *Operations Research*, vol. 55, no. 6, pp. 1072–1089, 2007.

[6] M. Chowdhury and I. Stoica, “Coflow: A networking abstraction for cluster applications,” in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: Association for Computing Machinery, 2012, p. 31–36.

[7] ———, “Efficient coflow scheduling without prior knowledge,” in *Proceedings of the 2015 ACM Conference on SIGCOMM*, ser. SIGCOMM ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 393–406.

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

[9] M. Chowdhury, Y. Zhong, and I. Stoica, “Efficient coflow scheduling with varys,” in *Proceedings of the 2014 ACM Conference on SIGCOMM*, ser. SIGCOMM ’14. New York, NY, USA: Association for Computing Machinery, 2014, p. 443–454.

[10] F. A. Chudak and D. B. Shmoys, “Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds,” *Journal of Algorithms*, vol. 30, no. 2, pp. 323–343, 1999.

[11] J. Dean and S. Ghemawat, “Mapreduce: Simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, p. 107–113, jan 2008.

[12] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron, “Decentralized task-aware scheduling for data center networks,” *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 431–442, 2014.

[13] N. Garg, A. Kumar, and V. Pandit, “Order scheduling models: hardness and algorithms,” in *International Conference on Foundations of Software Technology and Theoretical Computer Science*. Springer, 2007, pp. 96–107.

[14] X. S. Huang, Y. Xia, and T. S. E. Ng, “Weaver: Efficient coflow scheduling in heterogeneous parallel networks,” in *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2020, pp. 1071–1081.

[15] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, “Dryad: distributed data-parallel programs from sequential building blocks,” in *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, 2007, pp. 59–72.

[16] S. Khuller and M. Purohit, “Brief announcement: Improved approximation algorithms for scheduling co-flows,” in *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, 2016, pp. 239–240.

[17] J. Y.-T. Leung, H. Li, and M. Pinedo, “Scheduling orders for multiple product types to minimize total weighted completion time,” *Discrete Applied Mathematics*, vol. 155, no. 8, pp. 945–970, 2007.

[18] S. Li, “Scheduling to minimize total weighted completion time via time-indexed linear programming relaxations,” *SIAM Journal on Computing*, vol. 49, no. 4, pp. FOCS17–409, 2020.
[19] M. Mastrolilli, M. Queyranne, A. S. Schulz, O. Svensson, and N. A. Uhan, “Minimizing the sum of weighted completion times in a concurrent open shop,” *Operations Research Letters*, vol. 38, no. 5, pp. 390–395, 2010.

[20] Z. Qiu, C. Stein, and Y. Zhong, “Minimizing the total weighted completion time of coflows in datacenter networks,” in *Proceedings of the 27th ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 294–303.

[21] S. Sachdeva and R. Saket, “Optimal inapproximability for scheduling problems via structural hardness for hypergraph vertex cover,” in *2013 IEEE Conference on Computational Complexity*, 2013, pp. 219–229.

[22] M. Shafiee and J. Ghaderi, “An improved bound for minimizing the total weighted completion time of coflows in datacenters,” *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1674–1687, 2018.

[23] ——, “Scheduling coflows with dependency graph,” *IEEE/ACM Transactions on Networking*, 2021.

[24] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “The hadoop distributed file system,” in *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, 2010, pp. 1–10.

[25] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, A. Kanagala, J. Provost, J. Simmons, E. Tanda, J. Wanderer, U. Hölzle, S. Stuart, and A. Vahdat, “Jupiter rising: A decade of clos topologies and centralized control in google’s datacenter network,” in *Proceedings of the 2015 ACM Conference on SIGCOMM*, ser. SIGCOMM ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 183–197.

[26] G. Wang and T. E. Cheng, “Customer order scheduling to minimize total weighted completion time,” *Omega*, vol. 35, no. 5, pp. 623–626, 2007.

[27] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets,” in *2nd USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 10)*, 2010.

[28] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, “Coda: Toward automatically identifying and scheduling coflows in the dark,” in *Proceedings of the 2016 ACM Conference on SIGCOMM*, ser. SIGCOMM ’16. New York, NY, USA: Association for Computing Machinery, 2016, p. 160–173.

[29] 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 *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 424–432.