Efficient Approximation Algorithms for Scheduling Coflows with Total Weighted Completion Time in Identical Parallel Networks

Chi-Yeh Chen

Abstract—This paper addresses the scheduling problem of coflows in identical parallel networks, a well-known $\mathcal{NP}$-hard problem. We consider both flow-level scheduling and coflow-level scheduling problems. In the flow-level scheduling problem, flows within a coflow can be transmitted through different network cores, while in the coflow-level scheduling problem, flows within a coflow must be transmitted through the same network core. The key difference between these two problems lies in their scheduling granularity. Previous approaches relied on linear programming to solve the scheduling order. In this paper, we enhance the efficiency of solving by utilizing the primal-dual method. For the flow-level scheduling problem, we propose an approximation algorithm that achieves approximation ratios of $6-\frac{2}{m}$ and $5-\frac{2}{m}$ for arbitrary and zero release times, respectively, where m represents the number of network cores. Additionally, for the coflow-level scheduling problem, we introduce an approximation algorithm that achieves approximation ratios of $4m+1$ and $4m$ for arbitrary and zero release times, respectively. The algorithm presented in this paper has practical applications in data centers, such as those operated by Google or Facebook. The simulated results demonstrate the superior performance of our algorithms compared to previous approach, emphasizing their practical utility.

Index Terms—Scheduling algorithms, approximation algorithms, coflow, datacenter network, identical parallel network.

1 Introduction

With the increasing demand for computing power, large data centers have become vital components of cloud computing. In these data centers, the advantages of application-aware network scheduling have been demonstrated, particularly for distributed applications with structured traffic patterns [1], [10], [12], [33]. Data-parallel computing applications such as MapReduce [14], Hadoop [5], [27], Dryad [19], and Spark [32] have gained significant popularity among users, resulting in a proliferation of related applications [11], [15].

During the computing stage, data-parallel applications generate a substantial amount of intermediate data (flows) that needs to be transmitted across various machines for further processing during the communication stage. Given the multitude of applications and their corresponding data transmission requirements, it is crucial for data centers to possess robust data transmission and scheduling capabilities. Understanding the communication patterns of data-parallel computing applications, the interaction among flows between two sets of machines becomes a critical aspect. This overall communication pattern within the data center is abstracted by coflow traffic [9].

A coflow refers to a collection of interdependent flows, where the completion time of the entire group relies on the completion time of the last flow within the collection [25]. Previous research on coflow scheduling has predominantly focused on the single-core model [18], which has been widely utilized in various coflow-related studies [1], [2], [10], [12], [17], [20], [21], [25], [26], [33]. However, advancements in technology have led to the emergence of data centers that operate on multiple parallel networks to enhance efficiency [18], [28]. One such architecture is the identical or heterogeneous parallel network, where multiple network cores operate in parallel, providing aggregated bandwidth by concurrently serving traffic.

The algorithm presented in this paper finds application in the following scenario: Google [29] implemented traditional cluster networks utilizing the highest density Ethernet switches available, with 512 ports of 1GE, to establish the network backbone. Each Top of Rack (ToR) switch was connected to all four of the cluster routers, ensuring both scalability and fault tolerance. Furthermore, they started with the key insight that they could expand cluster fabrics to nearly arbitrary sizes by leveraging Clos topologies and taking advantage of the then-emerging merchant switching silicon industry. For a more comprehensive understanding of network architectures, please consult the paper [29].

This paper focuses on an architecture that employs multiple identical network cores operating in parallel. The objective is to schedule coflows in these parallel networks in a way that minimizes the total weighted coflow completion time. The problem is approached from two perspectives: flow-level scheduling and coflow-level scheduling. In the flow-level scheduling problem, the flows within a coflow can be distributed across different network cores, but the data in each flow is restricted to a single core. In contrast, the coflow-level scheduling problem requires that all flows within a coflow be distributed exclusively within the same network core. The key difference between these two problems lies in their scheduling granularity.

• *The author is with the Department of Computer Science and Information Engineering, National Cheng Kung University, No. 1, University Road, Tainan, Taiwan, ROC.*
E-mail: chency@csie.ncku.edu.tw.

Coarse-grained scheduling, associated with the coflow-level scheduling problem, enables faster resolution but yields relatively inferior scheduling outcomes. On the other hand, fine-grained scheduling, which pertains to the flow-level scheduling problem, takes more time to solve but produces superior scheduling outcomes.

1.1 Related Work

Chowdhury and Stoica [9] initially introduced the concept of coflow abstraction to characterize communication patterns within data centers. The scheduling problem for coflows has been proven to be strongly $\mathcal{NP}$-hard, necessitating the use of efficient approximation algorithms instead of exact solutions. Due to the inapproximability of the concurrent open shop problem [4], [23], it is $\mathcal{NP}$-hard to approximate the coflow scheduling problem within a factor better than $2-\epsilon$. Since the proposal of the coflow abstraction, numerous investigations have been conducted on coflow scheduling [2], [10], [12], [21], [25], [35].

Qiu *et al.* [21] introduced the first deterministic $\frac{64}{3}$-approximation and randomized $(8+\frac{16\sqrt{2}}{3})$-approximation algorithms for minimizing the weighted completion time of coflows. When coflows are released at arbitrary times, their algorithms achieved $\frac{67}{3}$ in the deterministic approach and $(9+\frac{16\sqrt{2}}{3})$ in the randomized approach. Their algorithm partitions coflows into disjoint groups using a linear program. Each group can be treated as a single coflow, and its optimal schedule can be determined in polynomial time. However, Ahmadi *et al.* [2] demonstrated that the technique proposed by Qiu *et al.* [21] actually only achieved $\frac{76}{3}$ in the deterministic algorithm for coflow scheduling with release time.

Khuller *et al.* [20] also proposed a 12-approximation algorithm for coflow scheduling with arbitrary release times. Furthermore, the algorithm achieves a deterministic 8-approximation and a randomized $3+2\sqrt{2}\approx 5.83$-approximation when all coflows have zero release times. The algorithm uses an approximate algorithm for the concurrent open shop problem to determine the scheduling order of coflows.

In recent research, Shafiee and Ghaderi [25] have developed an approximation algorithm which achieves 5 and 4 approximation rations for arbitrary and zero release time, respectively. Their method uses a linear program approach based on ordered variables to obtain the scheduling order of coflows. Moreover, Ahmadi *et al.* [2] have also proposed a primal-dual algorithm that achieved the same approximation ratio as Shafiee and Ghaderi [25].

Huang *et al.* [18] proposed an $O(m)$-approximation algorithm for scheduling a single coflow in a heterogeneous parallel network, where m represents the number of network cores. Chen [8] further introduces $O(\frac{\log m}{\log\log m})$-approximation algorithms for both the makespan scheduling problem and the total weighted completion time scheduling problem.

When multiple coflows with precedence constraints exist within a job, Tian *et al.* [31] were the first to propose a $O(N)$-approximation algorithm, where N represents the number of servers in the network. Shafiee and Ghaderi [26] then devised a polynomial-time algorithm with an approximation ratio of $O(\tilde{\mu}\log(N)/\log(\log(N)))$, where $\tilde{\mu}$ denotes the maximum number of coflows in a job. Furthermore, there are several recent studies [30], [34] on coflow scheduling in optical circuit switches.

1.2 Our Contributions

This paper focuses on addressing the coflow scheduling problem within identical parallel networks and presents a range of algorithms and corresponding results. The specific contributions of this study are outlined below:

- For the flow-level scheduling problem, we introduce an approximation algorithm that achieves approximation ratios of $6-\frac{2}{m}$ and $5-\frac{2}{m}$ for arbitrary and zero release times, respectively.
- For the coflow-level scheduling problem, we propose an approximation algorithm that achieves approximation ratios of $4m+1$ and $4m$ for arbitrary and zero release times, respectively.

A summary of our theoretical findings is provided in Table 1.

1.3 Organization

The structure of this paper is as follows. In Section 2, we provide an introduction to the fundamental notations and preliminary concepts that will be utilized in subsequent sections. Section 3 presents an overview of previous methods and our high-level ideas. Next, we present our primary algorithms in the following sections: Section 4 outlines the algorithm for addressing the flow-level scheduling problem, while Section 5 elaborates on the algorithm designed for the coflow-level scheduling problem. In Section 6, we conduct a comparative analysis to evaluate the performance of our proposed algorithms against that of the previous algorithm. Finally, in Section 7, we summarize our findings and draw meaningful conclusions.

2 Notation and Preliminaries

The identical parallel networks are represented as a collection of m large-scale non-blocking switches, each with dimensions of $N\times N$. In this configuration, N input links are connected to N source servers, and N output links are connected to N destination servers. Each switch corresponds to a network core, making the model straightforward and practical. Network architectures like Fat-tree or Clos [3], [16] can be utilized to establish networks that offer complete bisection bandwidth. In this parallel networks configuration, every parallel switch is linked to N source servers and N destination servers. Specifically, the i-th input or j-th output port of each switch is connected to the i-th source server or j-th destination server, respectively. As a result, each source (or destination) server possesses m simultaneous uplinks (or downlinks), with each link potentially comprising multiple physical connections in the actual topology [18]. Let $\mathcal{I}$ denote the set of source servers, and $\mathcal{J}$ denote the set of destination servers. The network core can be viewed as a bipartite graph, with $\mathcal{I}$ on one side and $\mathcal{J}$ on the other. For simplicity, we assume that all network cores are identical,

TABLE 1: Theoretical Results

Model	Release Times	Approximation Ratio	
flow-level scheduling problem	$\surd$	$6-\frac{2}{m}$	Thm 4.4
flow-level scheduling problem		$5-\frac{2}{m}$	Thm 4.5
coflow-level scheduling problem	$\surd$	$4m+1$	Thm 5.4
coflow-level scheduling problem		$4m$	Thm 5.5

TABLE 2: Notation and Terminology

m	The number of network cores.
N	The number of input/output ports.
n	The number of coflows.
$\mathcal{I}, \mathcal{J}$	The source server set and the destination server set.
$\mathcal{K}$	The set of coflows.
$D^{(k)}$	The demand matrix of coflow k.
$d_{i,j,k}$	The size of the flow to be transferred from input i to output j in coflow k.
C_k	The completion time of coflow k.
$C_{i,j,k}$	The completion time of flow (i,j,k).
r_k	The released time of coflow k.
w_k	The weight of coflow k.
$\mathcal{F}_i$	$\mathcal{F}_i = \left\{(i,j,k)\|d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall j \in \mathcal{J}\right\}$ is the set of flows with source i.
$\mathcal{F}_j$	$\mathcal{F}_j = \left\{(i,j,k)\|d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall i \in \mathcal{I}\right\}$ is the set of flows with destination j.
$d(S), d^2(S)$	$d(S) = \sum_{(i,j,k)\in S} d_{i,j,k}$ and $d^2(S) = \sum_{(i,j,k)\in S} d^2_{i,j,k}$ for any subset $S \subseteq \mathcal{F}_i$ (or $S \subseteq \mathcal{F}_j$).
$f(S)$	$f(S) = \frac{d(S)^2 + d^2(S)}{2m}$ for any subset $S \subseteq \mathcal{F}_i$ (or $S \subseteq \mathcal{F}_j$).
$L_{i,S,k}$	$L_{i,S,k} = \sum_{(i',j',k')\in S/i'=i,k'=k} d_{i',j',k'}$ is the total load on input port i for coflow k in the set S.
$L_{j,S,k}$	$L_{j,S,k} = \sum_{(i',j',k')\in S/j'=j,k'=k} d_{i',j',k'}$ is the total load on output port j for coflow k in the set S.
$L_{i,k}$	$L_{i,k} = \sum_{j\in\mathcal{J}} d_{i,j,k}$ is the total load of flows from the coflow k at input port i.
$L_{j,k}$	$L_{j,k} = \sum_{i\in\mathcal{I}} d_{i,j,k}$ is the total load of flows from the coflow k at output port j.
L_i, L_j	$L_i = \sum_{k\in\mathcal{K}} L_{i,k}$ and $L_j = \sum_{k\in\mathcal{K}} L_{j,k}$.
$S_{i,k}$	$S_{i,k}$ is the set of flows from the first k coflows at input port i.
$S_{j,k}$	$S_{j,k}$ is the set of flows from the first k coflows at output port j.
$f_i(S)$	$f_i(S) = \frac{\sum_{k\in S} L^2_{i,k} + \left(\sum_{k\in S} L_{i,k}\right)^2}{2m}$ for any subset $S \subseteq \mathcal{K}$.
$f_j(S)$	$f_j(S) = \frac{\sum_{k\in S} L^2_{j,k} + \left(\sum_{k\in S} L_{j,k}\right)^2}{2m}$ for any subset $S \subseteq \mathcal{K}$.
S_k	$S_k = \{1, 2, \ldots, k\}$ is the set of first k coflows.
$L_i(S_k), L_j(S_k)$	$L_i(S_k) = \sum_{k'\le k} L_{i,k'}$ and $L_j(S_k) = \sum_{k'\le k} L_{j,k'}$.
$\mu_1(k)$	$\mu_1(k)$ is the input port in S_k with the highest load.
$\mu_2(k)$	$\mu_2(k)$ is the output port in S_k with the highest load.

and all links within each network core possess the same capacity or speed.

A coflow is a collection of independent flows, and its completion time is determined by the completion time of the last flow in the set. The coflow k is represented by an $N \times N$ demand matrix $D^{(k)} = (d_{i,j,k})_{i,j=1}^{N}$, where $d_{i,j,k}$ denotes the size of the flow to be transferred from input i to output j within coflow k. Since all network cores are identical, the flow size can be considered equivalent to the transmission time. Each flow is identified by a triple (i,j,k), where $i \in \mathcal{I}$ represents the source node, $j \in \mathcal{J}$ represents the destination node, and k corresponds to the coflow. Furthermore, we assume that flows are composed of discrete data units, resulting in integer sizes. For the sake of simplicity, we assume that all flows within a coflow are initiated simultaneously, as demonstrated in [21].

This paper addresses the problem of coflow scheduling with release times. The problem involves a set of coflows denoted by $\mathcal{K}$, where coflow k is released into the system at time r_k. Consequently, scheduling for coflow k is only possible after time r_k. The completion time of coflow k, denoted as C_k, represents the time required for all flows within the coflow to finish processing. Each coflow $k \in \mathcal{K}$ is assigned a positive weight w_k. The objective is to schedule the coflows in an identical parallel network to minimize the total weighted completion time of the coflows, represented by $\sum_{k\in\mathcal{K}} w_k C_k$. To aid in explanation, we assign different meanings to the same symbols with different subscript symbols. Subscript i represents the index of the source (or the input port), subscript j represents the index of the destination (or the output port), and subscript k represents the index of the coflow. For instance, $\mathcal{F}_i$ denotes the set of flows with source i, and $\mathcal{F}_j$ represents the set of flows with destination j. The notation and terminology used in this paper are summarized in Table 2.

3 The Previous Methods and Our High-level Ideas

The first algorithm proposed for scheduling coflows in a parallel network is Weaver [18]. Weaver initially addressed the flow-level scheduling problem with the aim of minimizing the makespan. It schedules flows in descending order of byte size. When selecting a network core for a flow, it first assesses whether that flow would impact the makespan. If the flow doesn't affect the makespan, it is considered a non-critical flow, and the goal is to balance the load among the network cores when choosing a network core for it. Conversely, if the flow does affect the makespan, the objective is to minimize the makespan when selecting a network core for it.

Chen [6], [7] proposed scheduling coflows in a parallel network to minimize the total weighted completion time of the coflows. He initially employed a linear program to determine the sequence of coflow scheduling and selected the appropriate network cores with the aim of minimizing the coflow completion times. However, this approach, due to its reliance on linear programming, comes with higher time complexity and requires a significant amount of memory space to store the linear program's constraints.

Therefore, this paper transforms the linear program into a primal-dual method. In this approach, a feasible schedule is constructed iteratively from right to left, determining the processing order of coflows, starting from the last coflow and moving towards the first. We first identify the coflow with the largest release time among those that haven't been scheduled. Then, we compare this release time to the product of a parameter and the data size of the currently unscheduled coflows. If the comparison indicates that the unscheduled coflows' data size is bounded by this release time, meaning that the coflow's transmission time is constrained below its release time, we choose this coflow at its current scheduling position. If not, we choose another coflow. The selected coflow must adjust its corresponding dual variable without violating the dual constraints of other coflows, ensuring that its dual constraint becomes tight. In other words, among the unscheduled coflows, we seek the coflow for which the adjustment needed for its dual variable is minimized. Once the coflow scheduling order is determined, we utilize a list algorithm to select network cores in a manner that minimizes the completion time of the coflows.

4 Approximation Algorithm for the Flow-level Scheduling Problem

This section addresses the flow-level scheduling problem, which enables different flows in a coflow to be transmitted through distinct cores. We assume that coflows are transmitted at the flow level, ensuring that the data within a flow is allocated to the same core. We define $\mathcal{F}_i$ as the collection of flows with source i, represented by $\mathcal{F}_i = \{(i,j,k)|d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall j \in \mathcal{J}\}$, and $\mathcal{F}_j$ as the set of flows with destination j, given by $\mathcal{F}_j = \{(i,j,k)|d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall i \in \mathcal{I}\}$. For any subset $S \subseteq \mathcal{F}_i$ (or $S \subseteq \mathcal{F}_j$), we define $d(S) = \sum_{(i,j,k)\in S} d_{i,j,k}$ as the sum of data size over all flows in S and $d^2(S) = \sum_{(i,j,k)\in S} d^2_{i,j,k}$ as the sum of squares of data size over all flows in S. Let

$$f(S) = \frac{d(S)^2 + d^2(S)}{2m}.$$

The problem can be formulated as a linear programming relaxation, given by:

$$\begin{aligned}
\min \quad & \sum_{k\in\mathcal{K}} w_k C_k && && (1)\\
\text{s.t.} \quad & C_k \geq C_{i,j,k}, && \forall k \in \mathcal{K}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J} && (1a)\\
& C_{i,j,k} \geq r_k + d_{i,j,k}, && \forall k \in \mathcal{K}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J} && (1b)\\
& \sum_{(i,j,k)\in S} d_{i,j,k} C_{i,j,k} \geq f(S), && \forall i \in \mathcal{I}, \forall S \subseteq \mathcal{F}_i && (1c)\\
& \sum_{(i,j,k)\in S} d_{i,j,k} C_{i,j,k} \geq f(S), && \forall j \in \mathcal{J}, \forall S \subseteq \mathcal{F}_j && (1d)
\end{aligned}$$

In the linear program (1), the variable C_k represents the completion time of coflow k in the schedule, and $C_{i,j,k}$ denotes the completion time of flow (i,j,k). Constraint (1a) specifies that the completion time of coflow k is limited by all its flows, while constraint (1b) ensures that the completion time of any flow (i,j,k) is at least its release time r_k plus its load. Constraints (1c) and (1d) serve to impose a lower bound on the completion time variable in the input port and output port, respectively.

Let $L_{i,S,k} = \sum_{(i',j',k')\in S/i'=i,k'=k} d_{i',j',k'}$ be the total load on input port i for coflow k in the set S. Let $L_{j,S,k} = \sum_{(i',j',k')\in S/j'=j,k'=k} d_{i',j',k'}$ be the total load on output port j for coflow k in the set S. The dual linear program is given by

$$\begin{aligned}
\max \quad & \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \alpha_{i,j,k}(r_k + d_{i,j,k}) \\
& + \sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{F}_i} \beta_{i,S} f(S) \\
& + \sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{F}_j} \beta_{j,S} f(S) && && (2)\\
\text{s.t.} \quad & \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} \alpha_{i,j,k} \\
& + \sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{F}_i} \beta_{i,S} L_{i,S,k} \\
& + \sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{F}_j} \beta_{j,S} L_{j,S,k} \leq w_k, && \forall k \in \mathcal{K} && (2a)\\
& \alpha_{i,j,k} \geq 0, && \forall k \in \mathcal{K}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J} && (2b)\\
& \beta_{i,S} \geq 0, && \forall i \in \mathcal{I}, \forall S \subseteq \mathcal{F}_i && (2c)\\
& \beta_{j,S} \geq 0, && \forall j \in \mathcal{J}, \forall S \subseteq \mathcal{F}_j && (2d)
\end{aligned}$$

It is worth noting that for each flow (i,j,k), there is a corresponding dual variable $\alpha_{i,j,k}$, and for every coflow k, there exists a corresponding constraint. Moreover, for any subset $S \subseteq \mathcal{F}_i$ (or $S \subseteq \mathcal{F}_j$) of flows, there is a dual variable $\beta_{i,S}$ (or $\beta_{j,S}$). Importantly, it should be emphasized that the cost of any feasible dual solution serves as a lower bound for OPT, which represents the cost of an optimal solution.

The primal-dual algorithm (Algorithm 1) is inspired by the work of Davis *et al.* [13] and Ahmadi *et al.* [2]. In this algorithm, a feasible schedule is constructed iteratively from right to left, determining the processing order of coflows, starting from the last coflow and moving towards the first. Let us consider a specific iteration. At the beginning of this iteration, let $\mathcal{K}$ denote the set of coflows that have not been scheduled yet, and let k represent the coflow with the largest release time. In each iteration, a decision needs to be made regarding the increase of a α dual variable or a β variable. The dual LP serves as a guide for this decision-making process. If the release time r_k is very large, raising α yields significant gains in the dual objective function value. Conversely, if $L_{\mu_1(r)}$ (or $L_{\mu_2(r)}$ if $L_{\mu_2(r)} \geq L_{\mu_1(r)}$) is large, raising β leads to substantial improvements in the objective value. Let κ be a constant to be optimized later. If $r_k > \frac{\kappa \cdot L_{\mu_1(r)}}{m}$ (or $r_k > \frac{\kappa \cdot L_{\mu_2(r)}}{m}$ if $L_{\mu_2(r)} \geq L_{\mu_1(r)}$), the dual variable α is increased until the dual constraint for coflow k becomes tight. Consequently, coflow k is scheduled to be processed as early as possible and before any previously scheduled coflows.

If $r_k \leq \frac{\kappa \cdot L_{\mu_1(r)}}{m}$ (or $r_k \leq \frac{\kappa \cdot L_{\mu_2(r)}}{m}$ if $L_{\mu_2(r)} \geq L_{\mu_1(r)}$), the dual variable $\beta_{\mu_1(r),\mathcal{G}i}$ (or $\beta_{\mu_2(r),\mathcal{G}_j}$ if $L_{\mu_2(r)} \geq L_{\mu_1(r)}$) is increased until one of the constraints becomes tight for a coflow $k' \in \mathcal{K}$. Coflow k' is then scheduled to be processed

Algorithm 1 Permuting Coflows

```
1: $\mathcal{K}$ is the set of unscheduled coflows and initially $K = \{1, 2, \ldots, n\}$
2: $\mathcal{G}_i = \{(i, j, k) | d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall j \in \mathcal{J}\}$
3: $\mathcal{G}_j = \{(i, j, k) | d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall i \in \mathcal{I}\}$
4: $\alpha_{i,j,k} = 0$ for all $k \in \mathcal{K}, i \in \mathcal{I}, j \in \mathcal{J}$
5: $\beta_{i,S} = 0$ for all $i \in \mathcal{I}, S \subseteq \mathcal{F}_i$
6: $\beta_{j,S} = 0$ for all $j \in \mathcal{J}, S \subseteq \mathcal{F}_j$
7: $L_{i,k} = \sum_{j \in \mathcal{J}} d_{i,j,k}$ for all $k \in \mathcal{K}, i \in \mathcal{I}$
8: $L_{j,k} = \sum_{i \in \mathcal{I}} d_{i,j,k}$ for all $k \in \mathcal{K}, j \in \mathcal{J}$
9: $L_i = \sum_{k \in \mathcal{K}} L_{i,k}$ for all $i \in \mathcal{I}$
10: $L_j = \sum_{k \in \mathcal{K}} L_{j,k}$ for all $j \in \mathcal{J}$
11: for $r = n, n-1, \ldots, 1$ do
12:     $\mu_1(r) = \arg\max_{i \in \mathcal{I}} L_i$
13:     $\mu_2(r) = \arg\max_{j \in \mathcal{J}} L_j$
14:     $k = \arg\max_{\ell \in \mathcal{K}} r_\ell$
15:     if $L_{\mu_1(r)} > L_{\mu_2(r)}$ then
16:         if $r_k > \frac{\kappa \cdot L_{\mu_1(r)}}{m}$ then
17:             $\alpha_{\mu_1(r),1,k} = w_k - \sum_{i \in \mathcal{I}} \sum_{S \subseteq \mathcal{F}_i} \beta_{i,S} L_{i,S,k} - \sum_{j \in \mathcal{J}} \sum_{S \subseteq \mathcal{F}_j} \beta_{j,S} L_{j,S,k}$
18:             $\sigma(r) \leftarrow k$
19:         else if $r_k \leq \frac{\kappa \cdot L_{\mu_1(r)}}{m}$ then
20:             $k' = \arg\min_{k \in \mathcal{K}} \left\{ \frac{w_k - \sum_{i \in \mathcal{I}} \sum_{S \subseteq \mathcal{F}_i} \beta_{i,S} L_{i,S,k} - \sum_{j \in \mathcal{J}} \sum_{S \subseteq \mathcal{F}_j} \beta_{j,S} L_{j,S,k}}{L_{\mu_1(r), \mathcal{G}_{\mu_1(r)}, k}} \right\}$
21:             $\beta_{\mu_1(r), \mathcal{G}_{\mu_1(r)}} = \frac{w_{k'} - \sum_{i \in \mathcal{I}} \sum_{S \subseteq \mathcal{F}_i} \beta_{i,S} L_{i,S,k'} - \sum_{j \in \mathcal{J}} \sum_{S \subseteq \mathcal{F}_j} \beta_{j,S} L_{j,S,k'}}{L_{\mu_1(r), \mathcal{G}_{\mu_1(r)}, k'}}$
22:             $\sigma(r) \leftarrow k'$
23:         end if
24:     else
25:         if $r_k > \frac{\kappa \cdot L_{\mu_2(r)}}{m}$ then
26:             $\alpha_{1,\mu_2(r),k} = w_k - \sum_{i \in \mathcal{I}} \sum_{S \subseteq \mathcal{F}_i} \beta_{i,S} L_{i,S,k} - \sum_{j \in \mathcal{J}} \sum_{S \subseteq \mathcal{F}_j} \beta_{j,S} L_{j,S,k}$
27:             $\sigma(r) \leftarrow k$
28:         else if $r_k \leq \frac{\kappa \cdot L_{\mu_2(r)}}{m}$ then
29:             $k' = \arg\min_{k \in \mathcal{K}} \left\{ \frac{w_k - \sum_{i \in \mathcal{I}} \sum_{S \subseteq \mathcal{F}_i} \beta_{i,S} L_{i,S,k} - \sum_{j \in \mathcal{J}} \sum_{S \subseteq \mathcal{F}_j} \beta_{j,S} L_{j,S,k}}{L_{\mu_2(r), \mathcal{G}_{\mu_2(r)}, k}} \right\}$
30:             $\beta_{\mu_2(r), \mathcal{G}_{\mu_2(r)}} = \frac{w_{k'} - \sum_{i \in \mathcal{I}} \sum_{S \subseteq \mathcal{F}_i} \beta_{i,S} L_{i,S,k'} - \sum_{j \in \mathcal{J}} \sum_{S \subseteq \mathcal{F}_j} \beta_{j,S} L_{j,S,k'}}{L_{\mu_2(r), \mathcal{G}_{\mu_2(r)}, k'}}$
31:             $\sigma(r) \leftarrow k'$
32:         end if
33:     end if
34:     $\mathcal{K} \leftarrow \mathcal{K} \setminus \sigma(r)$
35:     $\mathcal{G}_i = \{(i, j, k) | d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall j \in \mathcal{J}\}$
36:     $\mathcal{G}_j = \{(i, j, k) | d_{i,j,k} > 0, \forall k \in \mathcal{K}, \forall i \in \mathcal{I}\}$
37:     $L_i = L_i - L_{i,\sigma(r)}$ for all $i \in \mathcal{I}$
38:     $L_j = L_j - L_{j,\sigma(r)}$ for all $j \in \mathcal{J}$
39: end for
```

as early as possible and before any previously scheduled coflows. Appendix A presents a simple and equivalent algorithm, which is Algorithm 5. This algorithm has a space complexity of $O(Nn)$ and a time complexity of $O(n^2)$, where N represents the number of input/output ports and n represents the number of coflows.

The flow-driven-list-scheduling algorithm, presented in Algorithm 2, utilizes a list scheduling rule. We assume, without loss of generality, that the coflows are ordered based on the permutation provided by Algorithm 1, where $\sigma(k) = k, \forall k \in \mathcal{K}$. The coflows are scheduled sequentially in this predefined order, and within each coflow, the flows are scheduled in non-increasing order of byte size, breaking ties arbitrarily. For each flow (i, j, k), the algorithm considers all congested flows that are scheduled before it. It then identifies the least loaded network core h^* and assigns flow (i, j, k) to this core in order to minimize its completion time. The specific steps involved in this process are outlined in lines 5-11 of the algorithm. Lines 12-26 of the algorithm are adapted from the work of Shafiee and Ghaderi [25]. It is worth noting that all flows are transmitted in a preemptible manner.

4.1 Analysis

In this section, we establish the efficacy of the proposed algorithm by proving its approximation ratios. Specifically, we demonstrate that the algorithm achieves an approximation ratio of $6 - \frac{2}{m}$ for arbitrary release times and an

Algorithm 2 Flow-Driven-List-Scheduling

```
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 flows 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 $k = 1, 2, \ldots, n$ do
6:      for every flow $(i,j,k)$ in non-increasing order of $d_{i,j,k}$, breaking ties arbitrarily do
7:          $h^* = \arg\min_{h\in[1,m]} (load_I(i,h) + load_O(j,h))$
8:          $\mathcal{A}_{h^*} = \mathcal{A}_{h^*} \cup \{(i,j,k)\}$
9:          $load_I(i,h^*) = load_I(i,h^*) + d_{i,j,k}$ and $load_O(j,h^*) = load_O(j,h^*) + d_{i,j,k}$
10:     end for
11: end for
12: for each $h \in [1,m]$ do in parallel do
13:     wait until the first coflow is released
14:     while there is some incomplete flow do
15:         for $k' = 1, 2, \ldots, n$ do
16:             for every released and incomplete flow $(i,j,k=k') \in \mathcal{A}_h$ in non-increasing order of $d_{i,j,k}$, breaking ties arbitrarily do
17:                 if the link $(i,j)$ is idle then
18:                     schedule flow $f$
19:                 end if
20:             end for
21:         end for
22:         while no new flow is completed or released do
23:             transmit the flows that get scheduled in line 18 at maximum rate 1.
24:         end while
25:     end while
26: end for
```

approximation ratio of $5 - \frac{2}{m}$ in the absence of release times. It is important to note that we assume the coflows are arranged in the order determined by the permutation generated by Algorithm 1, i.e., $\sigma(k) = k, \forall k \in \mathcal{K}$. Let $S_k = \{1, 2, \ldots, k\}$ denote the set of first k coflows. Let $S_{i,k}$ denote the set of flows from the first k coflows at input port i, that is

$$S_{i,k} = \{(i,j,k')|d_{i,j,k'} > 0, \forall k' \in \{1,\ldots,k\}, \forall j \in \mathcal{J}\}.$$

Similarly, let $S_{j,k}$ represent the set of flows from the first k coflows at output port j, that is

$$S_{j,k} = \{(i,j,k')|d_{i,j,k'} > 0, \forall k' \in \{1,\ldots,k\}, \forall i \in \mathcal{I}\}.$$

Let $\beta_{i,k} = \beta_{i,S_{i,k}}$ and $\beta_{j,k} = \beta_{j,S_{j,k}}$. Additionally, let $\mu_1(k)$ denote the input port with the highest load in S_k, and $\mu_2(k)$ denote the output port with the highest load in S_k. Recall that $d(S)$ represents the sum of loads for all flows in subset S. Therefore, $d(S_{i,k})$ corresponds to the total load of flows from the first k coflows at input port i, and $d(S_{j,k})$ corresponds to the total load of flows from the first k coflows at output port j. Let $L_{i,k} = \sum_{j\in\mathcal{J}} d_{i,j,k}$ be the total load of flows from the coflow k at input port i and $L_{j,k} = \sum_{i\in\mathcal{I}} d_{i,j,k}$ be the total load of flows from the coflow k at output port j.

Let us begin by presenting several key observations regarding the primal-dual algorithm.

Observation 4.1. *The following statements hold.*

1) *Every nonzero $\beta_{i,S}$ can be written as $\beta_{\mu_1(k),k}$ for some coflow k.*
2) *Every nonzero $\beta_{j,S}$ can be written as $\beta_{\mu_2(k),k}$ for some coflow k.*
3) *For every set $S_{\mu_1(k),k}$ that has a nonzero $\beta_{\mu_1(k),k}$ variable, if $k' \leq k$ then $r_{k'} \leq \frac{\kappa \cdot d(S_{\mu_1(k),k})}{m}$.*
4) *For every set $S_{\mu_2(k),k}$ that has a nonzero $\beta_{\mu_2(k),k}$ variable, if $k' \leq k$ then $r_{k'} \leq \frac{\kappa \cdot d(S_{\mu_2(k),k})}{m}$.*
5) *For every coflow k that has a nonzero $\alpha_{\mu_1(k),1,k}$, $r_k > \frac{\kappa \cdot d(S_{\mu_1(k),k})}{m}$.*
6) *For every coflow k that has a nonzero $\alpha_{1,\mu_2(k),k}$, $r_k > \frac{\kappa \cdot d(S_{\mu_2(k),k})}{m}$.*
7) *For every coflow k that has a nonzero $\alpha_{\mu_1(k),1,k}$ or a nonzero $\alpha_{1,\mu_2(k),k}$, if $k' \leq k$ then $r_{k'} \leq r_k$.*

Each of the aforementioned observations can be easily verified and their correctness can be directly inferred from Algorithm 1.

Observation 4.2. *For any subset S, we have that $d(S)^2 \leq 2m \cdot f(S)$.*

Lemma 4.3. *If there is an algorithm that generates a feasible coflow schedule such that for any coflow k, $C_k \leq a \cdot \max_{k' \leq k} r_k + \frac{d(S_{\mu_1(k),k}) + d(S_{\mu_2(k),k})}{m} + (1 - \frac{2}{m}) \max_{i,j} d_{i,j,k}$ for some constants a, then the total cost of the schedule is bounded as follows.*

$$\begin{aligned}\sum_k w_k C_k \quad &\leq \quad \left(a + \frac{2}{\kappa}\right) \sum_{k=1}^{n} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} \alpha_{i,j,k} r_k \\ &+2\left(a \cdot \kappa + 2\right) \sum_{i\in\mathcal{I}} \sum_{S\subseteq\mathcal{F}_i} \beta_{i,S} f(S) \\ &+2\left(a \cdot \kappa + 2\right) \sum_{j\in\mathcal{J}} \sum_{S\subseteq\mathcal{F}_j} \beta_{j,S} f(S) \\ &+\left(1 - \frac{2}{m}\right) \cdot OPT\end{aligned}$$

Proof. The proof is given in Appendix B. □

Theorem 4.4. *There exists a deterministic, combinatorial, polynomial time algorithm that achieves an approximation ratio of $6 - \frac{2}{m}$ for the flow-level scheduling problem with release times.*

Proof. To schedule coflows without release times, the application of Lemma 4.3 (with $a = 1$) indicates the following:

$$\begin{aligned}\sum_k w_k C_k \quad &\leq \quad \left(1 + \frac{2}{\kappa}\right) \sum_{k=1}^{n} \sum_{i\in\mathcal{I}} \sum_{j\in\mathcal{J}} \alpha_{i,j,k} r_k \\ &+2\left(\kappa + 2\right) \sum_{i\in\mathcal{I}} \sum_{S\subseteq\mathcal{F}_i} \beta_{i,S} f(S) \\ &+2\left(\kappa + 2\right) \sum_{j\in\mathcal{J}} \sum_{S\subseteq\mathcal{F}_j} \beta_{j,S} f(S) \\ &+\left(1 - \frac{2}{m}\right) \cdot OPT.\end{aligned}$$

In order to minimize the approximation ratio, we can substitute $\kappa = \frac{1}{2}$ and obtain the following result:

$$\begin{aligned}\sum_k w_k C_k &\leq 5\sum_{k=1}^{n}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\alpha_{i,j,k}r_k \\ &\quad +5\sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{F}_i}\beta_{i,S}f(S) \\ &\quad +5\sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{F}_j}\beta_{j,S}f(S) \\ &\quad +\left(1-\frac{2}{m}\right)\cdot OPT \\ &\leq \left(6-\frac{2}{m}\right)\cdot OPT.\end{aligned}$$

□

Theorem 4.5. *There exists a deterministic, combinatorial, polynomial time algorithm that achieves an approximation ratio of $5-\frac{2}{m}$ for the flow-level scheduling problem without release times.*

Proof. To schedule coflows without release times, the application of Lemma 4.3 (with $a = 0$) indicates the following:

$$\begin{aligned}\sum_k w_k C_k &\leq \left(\frac{2}{\kappa}\right)\sum_{k=1}^{n}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\alpha_{i,j,k}r_k \\ &\quad +2\cdot 2\sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{F}_i}\beta_{i,S}f(S) \\ &\quad +2\cdot 2\sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{F}_j}\beta_{j,S}f(S) \\ &\quad +\left(1-\frac{2}{m}\right)\cdot OPT.\end{aligned}$$

In order to minimize the approximation ratio, we can substitute $\kappa = \frac{1}{2}$ and obtain the following result:

$$\begin{aligned}\sum_k w_k C_k &\leq 4\sum_{k=1}^{n}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\alpha_{i,j,k}r_k \\ &\quad +4\sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{F}_i}\beta_{i,S}f(S) \\ &\quad +4\sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{F}_j}\beta_{j,S}f(S) \\ &\quad +\left(1-\frac{2}{m}\right)\cdot OPT \\ &\leq \left(5-\frac{2}{m}\right)\cdot OPT.\end{aligned}$$

□

5 Approximation Algorithm for the Coflow-level Scheduling Problem

This section specifically addresses the coflow-level scheduling problem, which involves the transmission of flows within a coflow through a single core. It is worth recalling that $L_{i,k} = \sum_{j=1}^{N} d_{i,j,k}$ and $L_{j,k} = \sum_{i=1}^{N} d_{i,j,k}$, where $L_{i,k}$ represents the total load at source i for coflow k, and $L_{j,k}$ represents the total load at destination j for coflow k. Let

$$f_i(S) = \frac{\sum_{k\in S} L_{i,k}^2 + \left(\sum_{k\in S} L_{i,k}\right)^2}{2m}$$

and

$$f_j(S) = \frac{\sum_{k\in S} L_{j,k}^2 + \left(\sum_{k\in S} L_{j,k}\right)^2}{2m}$$

for any subset $S \subseteq \mathcal{K}$. To address this problem, we propose a linear programming relaxation formulation as follows:

$$\begin{aligned}\text{min} \quad & \sum_{k\in\mathcal{K}} w_k C_k && && (3)\\ \text{s.t.} \quad & C_k \geq r_k + L_{i,k}, && \forall k\in\mathcal{K}, \forall i\in\mathcal{I} && (3a)\\ & C_k \geq r_k + L_{j,k}, && \forall k\in\mathcal{K}, \forall j\in\mathcal{J} && (3b)\\ & \sum_{k\in S} L_{i,k}C_k \geq f_i(S) && \forall i\in\mathcal{I}, \forall S\subseteq\mathcal{K} && (3c)\\ & \sum_{k\in S} L_{j,k}C_k \geq f_j(S) && \forall j\in\mathcal{J}, \forall S\subseteq\mathcal{K} && (3d)\end{aligned}$$

In the linear program (3), the completion time C_k is defined for each coflow k in the schedule. Constraints (3a) and (3b) are to ensure that the completion time of any coflow k is greater than or equal to its release time r_k plus its load. Furthermore, constraints (3c) and (3d) establish lower bounds for the completion time variable at the input port and the output port, respectively.

The dual linear program is given by

$$\begin{aligned}\text{max} \quad & \sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}\alpha_{i,k}(r_k + L_{i,k}) \\ & +\sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}}\alpha_{j,k}(r_k + L_{j,k}) \\ & +\sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{K}}\beta_{i,S}f_i(S) \\ & +\sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{K}}\beta_{j,S}f_j(S) && && (4)\\ \text{s.t.} \quad & \sum_{i\in\mathcal{I}}\alpha_{i,k} + \sum_{j\in\mathcal{J}}\alpha_{j,k} \\ & +\sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{K}/k\in S}\beta_{i,S}L_{i,k} \\ & +\sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{K}/k\in S}\beta_{j,S}L_{j,k} \leq w_k, && \forall k\in\mathcal{K} && (4a)\\ & \alpha_{i,k} \geq 0, && \forall k\in\mathcal{K}, \forall i\in\mathcal{I} && (4b)\\ & \alpha_{j,k} \geq 0, && \forall k\in\mathcal{K}, \forall j\in\mathcal{J} && (4c)\\ & \beta_{i,S} \geq 0, && \forall i\in\mathcal{I}, \forall S\subseteq\mathcal{K} && (4d)\\ & \beta_{j,S} \geq 0, && \forall j\in\mathcal{J}, \forall S\subseteq\mathcal{K} && (4e)\end{aligned}$$

Notice that for every coflow k, there exists two dual variables $\alpha_{i,k}$ and $\alpha_{j,k}$, and there is a corresponding constraint. Additionally, for every subset of coflows S, there are two dual variables $\beta_{i,S}$ and $\beta_{j,S}$. Algorithm 3 presents the primal-dual algorithm. The concept is the same as Algorithm 1. The difference is that scheduling is done from the perspective of coflows. Appendix C presents a simple and equivalent algorithm, which is Algorithm 6. This algorithm has a space complexity of $O(Nn)$ and a time complexity of $O(n^2)$, where N represents the number of input/output ports and n represents the number of coflows.

Algorithm 3 Permuting Coflows

```
1: $\mathcal{K}$ is the set of unscheduled coflows and initially $K = \{1, 2, \ldots, n\}$
2: $\alpha_{i,k} = 0$ for all $k \in \mathcal{K}, i \in \mathcal{I}$
3: $\alpha_{j,k} = 0$ for all $k \in \mathcal{K}, j \in \mathcal{J}$
4: $\beta_{i,S} = 0$ for all $i \in \mathcal{I}, S \subseteq \mathcal{K}$
5: $\beta_{j,S} = 0$ for all $j \in \mathcal{J}, S \subseteq \mathcal{K}$
6: $L_{i,k} = \sum_{j\in\mathcal{J}} d_{i,j,k}$ for all $k \in \mathcal{K}, i \in \mathcal{I}$
7: $L_{j,k} = \sum_{i\in\mathcal{I}} d_{i,j,k}$ for all $k \in \mathcal{K}, j \in \mathcal{J}$
8: $L_i = \sum_{k\in\mathcal{K}} L_{i,k}$ for all $i \in \mathcal{I}$
9: $L_j = \sum_{k\in\mathcal{K}} L_{j,k}$ for all $j \in \mathcal{J}$
10: for $r = n, n-1, \ldots, 1$ do
11:   $\mu_1(r) = \arg\max_{i\in\mathcal{I}} L_i$
12:   $\mu_2(r) = \arg\max_{j\in\mathcal{J}} L_j$
13:   $k = \arg\max_{\ell\in\mathcal{K}} r_\ell$
14:   if $L_{\mu_1(r)} > L_{\mu_2(r)}$ then
15:     if $r_k > \frac{\kappa\cdot L_{\mu_1(r)}}{m}$ then
16:       $\alpha_{\mu_1(r),k} = w_k - \sum_{i\in\mathcal{I}}\sum_{S\ni k}\beta_{i,S}L_{i,k} - \sum_{j\in\mathcal{J}}\sum_{S\ni k}\beta_{j,S}L_{j,k}$
17:       $\sigma(r) \leftarrow k$
18:     else if $r_k \le \frac{\kappa\cdot L_{\mu_1(r)}}{m}$ then
19:       $k' = \arg\min_{k\in\mathcal{K}}\left\{\frac{w_k-\sum_{i\in\mathcal{I}}\sum_{S\ni k}\beta_{i,S}L_{i,k}-\sum_{j\in\mathcal{J}}\sum_{S\ni k}\beta_{j,S}L_{j,k}}{L_{\mu_1(r),k}}\right\}$
20:       $\beta_{\mu_1(r),\mathcal{K}} = \frac{w_{k'}-\sum_{i\in\mathcal{I}}\sum_{S\ni k}\beta_{i,S}L_{i,k'}-\sum_{j\in\mathcal{J}}\sum_{S\ni k}\beta_{j,S}L_{j,k'}}{L_{\mu_1(r),k'}}$
21:       $\sigma(r) \leftarrow k'$
22:     end if
23:   else
24:     if $r_k > \frac{\kappa\cdot L_{\mu_2(r)}}{m}$ then
25:       $\alpha_{\mu_2(r),k} = w_k - \sum_{i\in\mathcal{I}}\sum_{S\ni k}\beta_{i,S}L_{i,k} - \sum_{j\in\mathcal{J}}\sum_{S\ni k}\beta_{j,S}L_{j,k}$
26:       $\sigma(r) \leftarrow k$
27:     else if $r_k \le \frac{\kappa\cdot L_{\mu_2(r)}}{m}$ then
28:       $k' = \arg\min_{k\in\mathcal{K}}\left\{\frac{w_k-\sum_{i\in\mathcal{I}}\sum_{S\ni k}\beta_{i,S}L_{i,k}-\sum_{j\in\mathcal{J}}\sum_{S\ni k}\beta_{j,S}L_{j,k}}{L_{\mu_2(r),k}}\right\}$
29:       $\beta_{\mu_2(r),\mathcal{K}} = \frac{w_{k'}-\sum_{i\in\mathcal{I}}\sum_{S\ni k}\beta_{i,S}L_{i,k'}-\sum_{j\in\mathcal{J}}\sum_{S\ni k}\beta_{j,S}L_{j,k'}}{L_{\mu_2(r),k'}}$
30:       $\sigma(r) \leftarrow k'$
31:     end if
32:   end if
33:   $\mathcal{K} \leftarrow \mathcal{K} \setminus \sigma(r)$
34:   $L_i = L_i - L_{i,\sigma(r)}$ for all $i \in \mathcal{I}$
35:   $L_j = L_j - L_{j,\sigma(r)}$ for all $j \in \mathcal{J}$
36: end for
```

The coflow-driven-list-scheduling (as described in Algorithm 4) operates as follows. We assume, without loss of generality, that the coflows are ordered based on the permutation provided by Algorithm 3, where $\sigma(k) = k, \forall k \in \mathcal{K}$ and schedule all the flows in each coflow iteratively, respecting the order in this list. For each coflow k, we determine a network core h^* that can transmit coflow k in a way that minimizes the complete time of coflow k (lines 5-9). We then transmit all the flows allocated to network core h in order to minimize its completion time.

5.1 Analysis

This section substantiates the effectiveness of the proposed algorithm by providing proof of its approximation ratios. Specifically, we demonstrate that the algorithm achieves an approximation ratio of $4m + 1$ for arbitrary release times and an approximation ratio of $4m$ in the absence of release times. It is important to note that we assume the coflows are arranged in the order determined by the permutation generated by Algorithm 3, i.e., $\sigma(k) = k, \forall k \in \mathcal{K}$. We also recall that $S_k = \{1, 2, \ldots, k\}$ denote the set of first k coflows. Let $\beta_{i,k} = \beta_{i,S_k}$ and $\beta_{j,k} = \beta_{j,S_k}$. Let $L_i(S_k) = \sum_{k'\le k} L_{i,k'}$ and $L_j(S_k) = \sum_{k'\le k} L_{j,k'}$. Additionally, let $\mu_1(k)$ denote the input port with the highest load in S_k, and $\mu_2(k)$ denote the output port with the highest load in S_k. Therefore, $L_{\mu_1(k)}(S_k) = \sum_{k'\le k} L_{\mu_1(k),k'}$ and $L_{\mu_2(k)}(S_k) = \sum_{k'\le k} L_{\mu_2(k),k'}$.

Let us begin by presenting several key observations regarding the primal-dual algorithm.

Observation 5.1. *The following statements hold.*

1) *Every nonzero $\beta_{i,S}$ can be written as $\beta_{\mu_1(k),k}$ for some coflow k.*
2) *Every nonzero $\beta_{j,S}$ can be written as $\beta_{\mu_2(k),k}$ for some coflow k.*
3) *For every set S_k that has a nonzero $\beta_{\mu_1(k),k}$ variable, if $k' \le k$ then $r_{k'} \le \frac{\kappa\cdot L_{\mu_1(k)}(S_k)}{m}$.*

Algorithm 4 coflow-driven-list-scheduling

```
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 $k = 1, 2, \ldots, n$ do
6:   $h^* = \arg\min_{h\in[1,m]} \left(\max_{i,j\in[1,N]} load_I(i,h) + load_O(j,h) + L_{i,k} + L_{j,k}\right)$
7:   $\mathcal{A}_{h^*} = \mathcal{A}_{h^*} \cup \{k\}$
8:   $load_I(i,h^*) = load_I(i,h^*) + L_{i,k}$ and $load_O(j,h^*) = load_O(j,h^*) + L_{j,k}$ for all $i,j \in [1,N]$
9: end for
10: for each $h \in [1,m]$ do in parallel do
11:   wait until the first coflow is released
12:   while there is some incomplete flow do
13:     for all $k \in \mathcal{A}_h$, list the released and incomplete flows respecting the increasing order in $k$
14:     let $L$ be the set of flows in the list
15:     for every flow $f = (i,j,k) \in L$ do
16:       if the link $(i,j)$ is idle then
17:         schedule flow $f$
18:       end if
19:     end for
20:     while no new flow is completed or released do
21:       transmit the flows that get scheduled in line 17 at maximum rate 1.
22:     end while
23:   end while
24: end for
```

4) *For every set S_k that has a nonzero $\beta_{\mu_2(k),k}$ variable, if $k' \leq k$ then $r_{k'} \leq \frac{\kappa \cdot L_{\mu_2(k)}(S_k)}{m}$.*
5) *For every coflow k that has a nonzero $\alpha_{\mu_1(k),k}$, $r_k > \frac{\kappa \cdot L_{\mu_1(k)}(S_k)}{m}$.*
6) *For every coflow k that has a nonzero $\alpha_{\mu_2(k),k}$, $r_k > \frac{\kappa \cdot L_{\mu_2(k)}(S_k)}{m}$.*
7) *For every coflow k that has a nonzero $\alpha_{\mu_1(k),k}$ or a nonzero $\alpha_{\mu_2(k),k}$, if $k' \leq k$ then $r_{k'} \leq r_k$.*

Each of the aforementioned observations can be easily verified and their correctness can be directly inferred from Algorithm 3.

Observation 5.2. *For any subset S, we have that $(\sum_{k\in S} L_{i,k})^2 \leq 2m \cdot f_i(S)$ and $(\sum_{k\in S} L_{j,k})^2 \leq 2m \cdot f_j(S)$.*

Lemma 5.3. *If there is an algorithm that generates a feasible coflow schedule such that for any coflow k, $C_k \leq a \cdot \max_{k'\leq k} r_k + L_{\mu_1(k)}(S_k) + L_{\mu_2(k)}(S_k)$ for some constants a, then the total cost of the schedule is bounded as follows.*

$$
\begin{aligned}
\sum_k w_k C_k \leq & \left(a + \frac{2m}{\kappa}\right) \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \alpha_{i,k}(r_k) \\
& + \left(a + \frac{2m}{\kappa}\right) \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} \alpha_{j,k}(r_k) \\
& + 2\left(a\cdot\kappa + 2m\right) \sum_{i\in\mathcal{I}} \sum_{S\subseteq\mathcal{K}} \beta_{i,S} f_i(S) \\
& + 2\left(a\cdot\kappa + 2m\right) \sum_{j\in\mathcal{J}} \sum_{S\subseteq\mathcal{K}} \beta_{j,S} f_j(S)
\end{aligned}
$$

Proof. The proof is given in Appendix D. □

Theorem 5.4. *There exists a deterministic, combinatorial, polynomial time algorithm that achieves an approximation ratio of $4m+1$ for the coflow-level scheduling problem with release times.*

Proof. To schedule coflows without release times, the application of Lemma 5.3 (with $a = 1$) indicates the following:

$$
\begin{aligned}
\sum_k w_k C_k \leq & \left(1 + \frac{2m}{\kappa}\right) \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \alpha_{i,k}(r_k) \\
& + \left(1 + \frac{2m}{\kappa}\right) \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} \alpha_{j,k}(r_k) \\
& + 2\left(\kappa + 2m\right) \sum_{i\in\mathcal{I}} \sum_{S\subseteq\mathcal{K}} \beta_{i,S} f_i(S) \\
& + 2\left(\kappa + 2m\right) \sum_{j\in\mathcal{J}} \sum_{S\subseteq\mathcal{K}} \beta_{j,S} f_j(S)
\end{aligned}
$$

In order to minimize the approximation ratio, we can substitute $\kappa = \frac{1}{2}$ and obtain the following result:

$$
\begin{aligned}
\sum_k w_k C_k \leq & (4m+1) \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \alpha_{i,k}(r_k) \\
& + (4m+1) \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} \alpha_{j,k}(r_k) \\
& + (4m+1) \sum_{i\in\mathcal{I}} \sum_{S\subseteq\mathcal{K}} \beta_{i,S} f_i(S) \\
& + (4m+1) \sum_{j\in\mathcal{J}} \sum_{S\subseteq\mathcal{K}} \beta_{j,S} f_j(S) \\
\leq & (4m+1) \cdot OPT.
\end{aligned}
$$

□

Theorem 5.5. *There exists a deterministic, combinatorial, polynomial time algorithm that achieves an approximation ratio of $4m$ for the coflow-level scheduling problem without release times.*

Proof. To schedule coflows without release times, the application of Lemma 5.3 (with $a = 0$) indicates the following:

$$
\begin{aligned}
\sum_k w_k C_k \leq & \left(\frac{2m}{\kappa}\right) \sum_{k\in\mathcal{K}} \sum_{i\in\mathcal{I}} \alpha_{i,k}(r_k) \\
& + \left(\frac{2m}{\kappa}\right) \sum_{k\in\mathcal{K}} \sum_{j\in\mathcal{J}} \alpha_{j,k}(r_k) \\
& + 2\left(2m\right) \sum_{i\in\mathcal{I}} \sum_{S\subseteq\mathcal{K}} \beta_{i,S} f_i(S) \\
& + 2\left(2m\right) \sum_{j\in\mathcal{J}} \sum_{S\subseteq\mathcal{K}} \beta_{j,S} f_j(S)
\end{aligned}
$$

In order to minimize the approximation ratio, we can substitute $\kappa = \frac{1}{2}$ and obtain the following result:

$$\begin{aligned}\sum_k w_k C_k \leq\ & \left(\frac{2m}{\kappa}\right)\sum_{k\in\mathcal{K}}\sum_{i\in\mathcal{I}}\alpha_{i,k}(r_k)\\ & +\left(\frac{2m}{\kappa}\right)\sum_{k\in\mathcal{K}}\sum_{j\in\mathcal{J}}\alpha_{j,k}(r_k)\\ & +2\,(2m)\sum_{i\in\mathcal{I}}\sum_{S\subseteq\mathcal{K}}\beta_{i,S}f_i(S)\\ & +2\,(2m)\sum_{j\in\mathcal{J}}\sum_{S\subseteq\mathcal{K}}\beta_{j,S}f_j(S)\\ \leq\ & 4m\cdot OPT.\end{aligned}$$

□

6 Results and Discussion

This section performs simulations to assess the performance of the proposed algorithm in comparison to a previous algorithm, using both synthetic and real traffic traces. This experiment is simulated using Python on an Intel i5 3.10 GHz, 32 GB RAM machine running Windows 10. The simulation results are presented and analyzed in the subsequent sections.

6.1 Workload

We employed the model presented in [25] to generate synthetic traces for our evaluation. For each coflow, we are provided with a coflow description in the form of $(W_{min}, W_{max}, L_{min}, L_{max})$. The number of non-zero flows in each coflow, denoted as M, is determined as the product of two randomly chosen values, w_1 and w_2, both falling within the interval $[W_{min}, W_{max}]$. Additionally, the input links are assigned w_1 and the output links are assigned w_2 in a random manner. The size of each flow is randomly selected from the interval $[L_{min}, L_{max}]$. The default configuration for constructing all coflows follows a specific percentage distribution based on the coflow descriptions: $(1, 4, 1, 10)$, $(1, 4, 10, 1000)$, $(4, N, 1, 10)$, and $(4, N, 10, 1000)$, with proportions of 41%, 29%, 9%, and 21%, respectively. Here, N represents the number of ports in the core. In comparing the effects of flow density, the coflows were categorized into three instances based on their sparsity: dense, sparse, and combined. For each instance, we randomly selected M flows from either the set $\{N, N+1, \dots, N^2\}$ or $\{1, 2, \dots, N\}$ depending on the specific instance. In the combined instance, each coflow was classified as either sparse or dense with equal probability. Subsequently, flow sizes were randomly assigned to each flow, following a uniform distribution over the range $\{1, 2, \dots, 100\}$ MB. The links of the switching core had a capacity of 128 MBps, and each time unit corresponded to 1/128 of a second (8 milliseconds), which equated to 1 MB per time unit. We generated 100 instances for each case and calculated the average performance of the algorithm.

The real traffic trace utilized in our study was sourced from the Hive/MapReduce traces captured from Facebook's 3000-machine cluster, comprising 150 racks. This trace has been widely employed in previous simulations by researchers [10], [21], [25]. The trace encompasses essential details, including the arrival time (measured in milliseconds) of each coflow, the location of the mappers and reducers (specifically, the rack number they belong to), and the amount of shuffle data for each reducer (expressed in Megabytes). The trace dataset consists of a total of 526 coflows.

6.2 Algorithms

In the case of $m = 1$, Algorithm 4 achieves approximation ratios of 5 and 4 for arbitrary release time and zero release time, respectively, which aligns with the findings reported in Shafiee and Ghaderi [25]. Their empirical evaluation has demonstrated that their algorithm outperforms deterministic algorithms used in Varys [12], [22], and [24]. Consequently, this paper focuses on simulating the performance of the algorithms in an identical parallel network ($m > 1$). We evaluate the performance of FDLS (Algorithm 2) and Weaver [18] in the identical parallel network setting. Weaver arranges the flows one by one, classifying them as critical or non-critical, and assigns a network core that minimizes the coflow completion time for critical flows. For non-critical flows, it selects a network core that balances the load among the network cores. Since Weaver only schedules one coflow on the parallel network, we employ our algorithm to determine the order of coflows. We then utilize Weaver to schedule the coflows in the obtained order. In other words, all algorithms schedule coflows in the same order, but the distinction lies in the method employed to minimize the completion time of each coflow.

In our simulations, we compute the approximation ratio by dividing the total weighted completion time obtained from the algorithms by the cost of the feasible dual solution. The feasible dual solution is obtained using Algorithms 5 and 6 and serves as a lower bound on the optimal value of the coflow scheduling problem. It is worth noting that when the cost of the feasible dual solution is significantly lower than the cost of the integer optimal solution, the resulting approximation ratio may exceed the theoretically analyzed approximation ratio. To assign weights to each coflow, we randomly and uniformly select positive integers from the interval $[1, 100]$. The release times for each coflow are randomly and uniformly selected as positive integers from the interval $[0, 100]$.

6.3 Results

Figure 1 illustrates the approximation ratio of the proposed algorithm compared to the previous algorithm for synthetic traces. The problem size ranges from 5 to 25 coflows in five network cores. The proposed algorithm demonstrates significantly smaller approximation ratios than $5 - \frac{2}{m}$ when all coflows are released at time 0 and significantly smaller approximation ratios than $6 - \frac{2}{m}$ when coflows have varying release times. Furthermore, FDLS outperforms Weaver by approximately 2.7% to 7.8% within this problem size range. Additionally, as the number of coflows increases, the approximation ratio decreases. This observation aligns with subsequent findings, indicating that dense instances result in lower approximation ratios. It is worth noting that the dual solutions for the case where all coflows are released at time 0 are different from the dual solutions when coflows have release times. Therefore, we can only observe whether the results for these two cases individually align with the theoretical analysis, and we cannot guarantee the relationship between these two cases. Although the approximation ratios for the case where coflows have release times are higher than the case where all coflows are released at time 0.

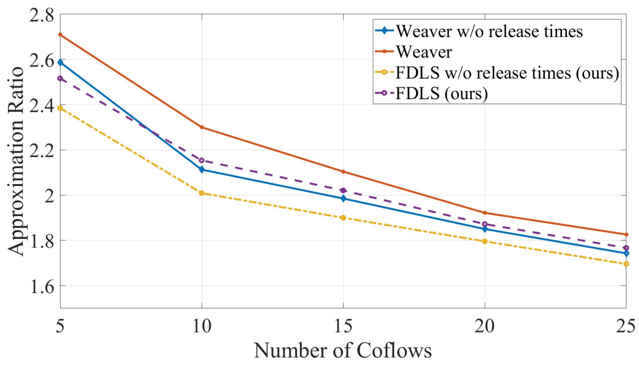

Fig. 1: The approximation ratio of synthetic traces between the previous algorithm and the proposed algorithm.

Figure 2 presents the approximation ratio of synthetic traces for 100 random dense and combined instances, comparing the previous algorithm with the proposed algorithm. The problem size consists of 25 coflows in five network cores, with input and output links of $N = 10$. In the dense case, Weaver achieves an approximation ratio of 1.35, which is worse than FDLS that attains an approximation ratio of 1.33 when all coflows release at time 0. This results in a 1.45% improvement when using FDLS, with an associated error percentage of 0.11%. When coflows have varying release times, Weaver achieves an approximation ratio of 1.42, which is worse than FDLS that attains an approximation ratio of 1.40. This results in a 1.72% improvement when using FDLS, with an associated error percentage of 0.17%. In the combined case, FDLS slightly outperforms Weaver. Notably, the proposed algorithm demonstrates a larger improvement in the dense case compared to the combined case.

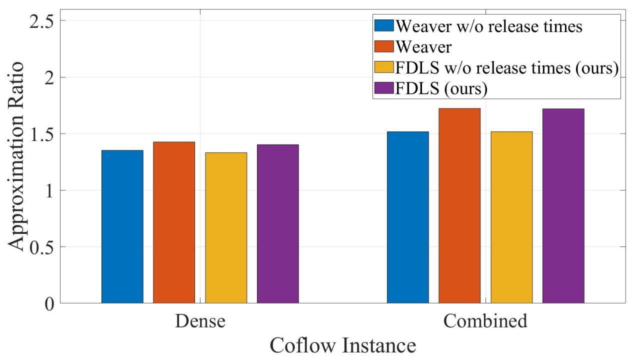

Fig. 2: The approximation ratio of synthetic traces between the previous algorithm and the proposed algorithm for different number of coflows for 100 random dense and combined instances.

Figure 3 depicts the approximation ratio of synthetic traces for varying numbers of network cores, comparing the previous algorithm to the proposed algorithm. The problem size consists of 25 coflows distributed across 5 to 25 network cores, with input and output links of $N = 10$. The proposed algorithm consistently achieves much smaller approximation ratios than $5 - \frac{2}{m}$ when all coflows are released at time 0 and much smaller approximation ratios than $6 - \frac{2}{m}$ when coflows have varying release times. As the number of network cores increases, the approximation ratio also increases. This trend is attributed to the widening gap between the cost of the feasible dual solution and the cost of the integer optimal solution as the number of network cores grows. Consequently, there is an amplified disparity between the experimental approximation ratio and the actual approximation ratio. Notably, FDLS outperforms Weaver by approximately 1.49% to 2.93% across different numbers of network cores.

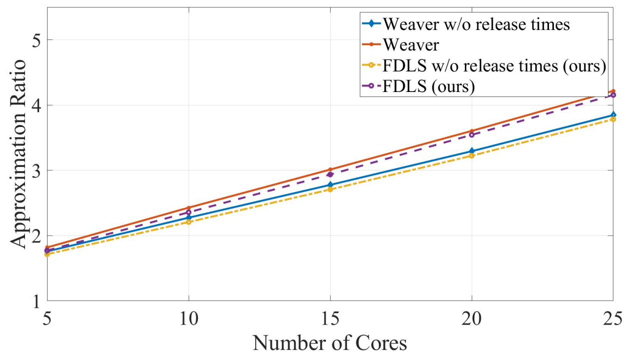

Fig. 3: The approximation ratio of synthetic traces between the previous algorithm and the proposed algorithm for different number of network cores.

Figure 4 presents the approximation ratio of real traces for various thresholds of the number of flows, comparing the previous algorithm to the proposed algorithm. Specifically, we apply a filter to the set of coflows based on the condition that the number of flows is greater than or equal to the threshold. The problem size comprises a randomly selected a set of 526 coflows distributed across five network cores, with input and output links of $N = 150$. Remarkably, FDLS outperforms Weaver by approximately 2.93% to 14.69% across different thresholds. Furthermore, as the number of flows increases, the approximation ratio decreases. This finding aligns with our previous observation, which indicates that dense instances yield lower approximation ratios.

Figure 5 illustrates the cumulative distribution function (CDF) plots of the coflow completion time for the previous algorithm and the proposed algorithm when all coflows are released at time 0. The problem size consists of 15 coflows in five network cores, with input and output links of $N = 10$. As shown in the figure, 92.86% of the coflow completion time achieved by FDLS is below 15.936 seconds, whereas Weaver's completion time is 17.824 seconds. Moreover, the area under the CDF plot of FDLS is larger than the area under the CDF plot of Weaver, providing further evidence of FDLS's superiority over Weaver.

Figure 6 presents a box plot of the approximation ratio

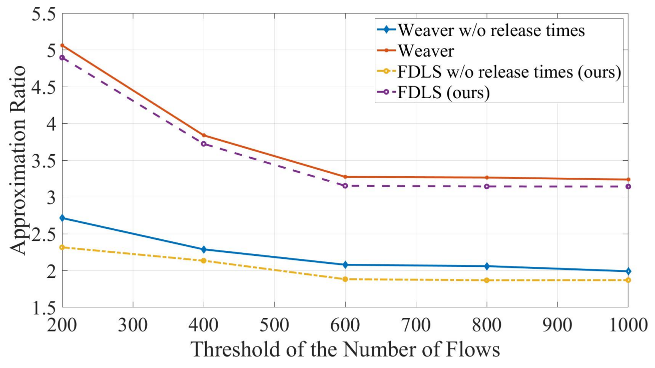

Fig. 4: The approximation ratio of real trace between the previous algorithm and the proposed algorithm for different threshold of the number of flows.

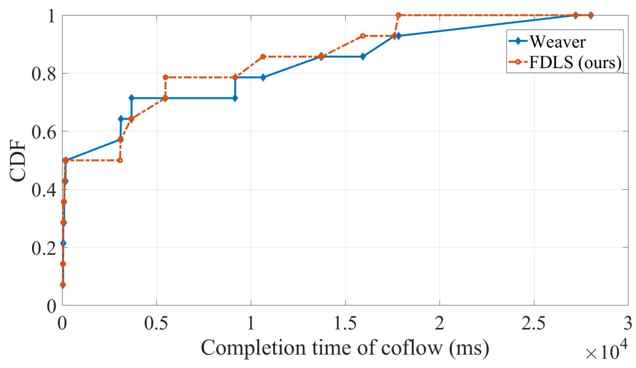

Fig. 5: CDF of coflow completion time under the previous algorithm and the proposed algorithm for synthetic traces when all coflows release at time 0.

for synthetic traces, comparing the previous algorithm and the proposed algorithm when all coflows are released at time 0. The problem size consists of 25 coflows in five network cores, with input and output links of $N = 10$. In FDLS, the first quartile (Q1), median, and third quartile (Q3) values are 1.6234, 1.7056, and 1.7932, respectively. The maximum and minimum values are 2.0746 and 1.3758, respectively. This result demonstrates that the proposed algorithm achieves a significantly smaller approximation ratio than $5-\frac{2}{m}$. For Weaver, the Q1, median, and Q3 values are 1.6459, 1.7603, and 1.8511, respectively, with maximum and minimum values of 2.0851 and 1.3741, respectively. This result indicates that FDLS is more stable and outperforms Weaver.

Next, we will discuss the results of the coflow-level scheduling problem. Given that Weaver originally tackled the flow-level scheduling problem to minimize the makespan, we can adapt Weaver to address the coflow-level scheduling problem while minimizing the completion time of each coflow. In this context, Weaver prioritizes selecting network cores that minimize the completion time of the individual coflow, rather than striving to balance the load among network cores. As a result, the modified Weaver approach for the coflow-level scheduling problem yields equivalent outcomes to those obtained using the the Coflow-Driven-List-Scheduling (CDLS) algorithm. Therefore, we will exclusively present the results obtained using CDLS.

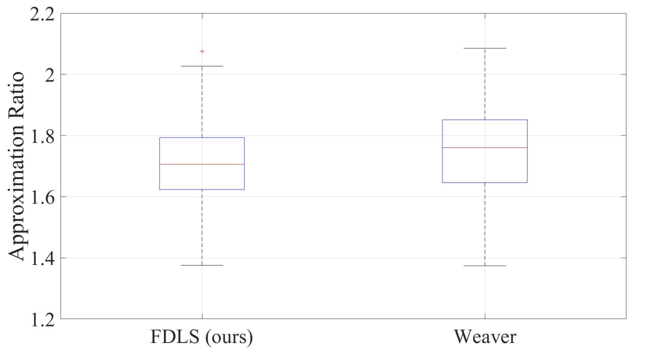

Fig. 6: Box plot of the approximation ratio under the previous algorithm and the proposed algorithm for synthetic traces when all coflows release at time 0.

Figure 7 illustrates the approximation ratio of CDLS algorithm for synthetic traces. The problem size ranges from 5 to 25 coflows in five network cores, with input and output links of $N = 10$. The proposed algorithm demonstrates outstanding performance by achieving significantly lower approximation ratios than $4m$ in the case where all coflows are released at time 0, and significantly lower approximation ratios than $4m + 1$ when coflows have varying release times. Furthermore, as the number of coflows increases, the approximation ratio decreases. This observation aligns with our previous finding, suggesting that dense instances result in lower approximation ratios.

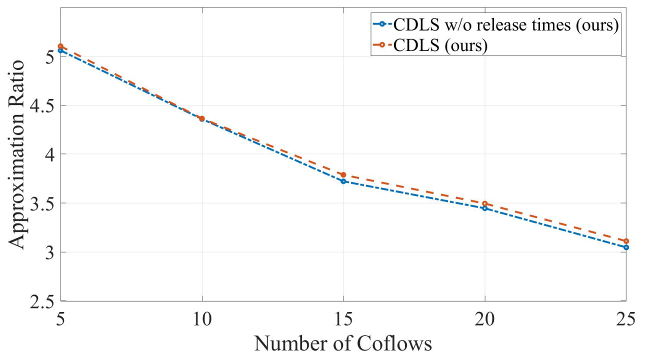

Fig. 7: The approximation ratio of coflow-driven-list-scheduling (CDLS) algorithm for different number of coflows.

Figure 8 depicts the approximation ratio of the CDLS algorithm for varying numbers of network cores. The problem size consists of 25 coflows distributed across 5 to 25 network cores, with input and output links of $N = 10$. The proposed algorithm consistently achieves much smaller approximation ratios than $4m$ in the case where all coflows are released at time 0, and much smaller approximation ratios than $4m+1$ when coflows have varying release times. As the number of network cores increases, the approximation ratio also increases. This is consistent with our theoretical results, showing a linear relationship between the approximation ratio and the number of network cores.

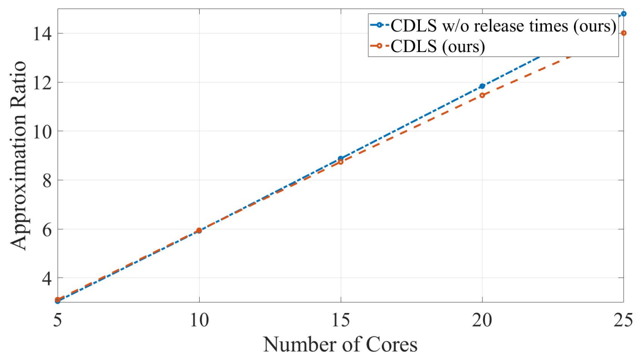

Fig. 8: The approximation ratio of coflow-driven-list-scheduling (CDLS) algorithm for different number of network cores.

Figure 9 displays the box plot of the approximation ratio of the CDLS algorithm for synthetic traces. The problem size consists of 25 coflows in five network cores, with input and output links of $N = 10$. When all coflows release at time 0, the CDLS algorithm achieves Q1, median, and Q3 values of 2.8731, 3.0426, and 3.2563, respectively. Additionally, the maximum and minimum values are 3.692 and 2.1815, respectively. These results demonstrate that the CDLS algorithm achieves a significantly smaller approximation ratio than $4m$. In the case where coflows have varying release times, the CDLS algorithm achieves Q1, median, and Q3 values of 2.8901, 3.0821, and 3.3163, respectively. Additionally, the maximum and minimum values are 3.8616 and 2.3019, respectively. These results further underscore that the CDLS algorithm achieves a significantly lower approximation ratio than $4m + 1$.

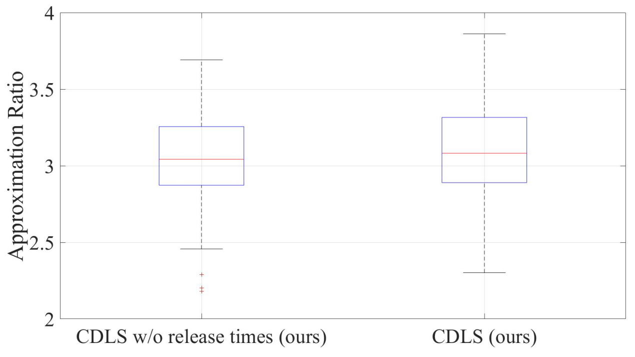

Fig. 9: Box plot of the approximation ratio under the coflow-driven-list-scheduling (CDLS) algorithm for synthetic traces.

7 Concluding Remarks

In recent years, with advancements in technology, the traditional single-core model has become insufficient. As a result, we have shifted our focus to studying identical parallel networks, which utilize multiple parallel-operating network cores. Previous approaches relied on linear programming to solve scheduling order. Although the linear program can be solved in polynomial time using the ellipsoid method, it requires exponentially more constraints, necessitating sufficient memory to store them. Therefore, this paper improves the efficiency of solving by employing the primal-dual method. The primal-dual algorithm has a space complexity of $O(Nn)$ and a time complexity of $O(n^2)$. We investigate both flow-level and coflow-level scheduling problems. Our proposed algorithm for the flow-level scheduling problem achieves an approximation ratio of $6 - \frac{2}{m}$ with arbitrary release times and $5 - \frac{2}{m}$ without release time. For the coflow-level scheduling problem, we obtain an approximation ratio of $4m + 1$ with arbitrary release times and $4m$ without release time.

Acknowledgments

This research was supported in part by the Ministry of Science and Technology, Taiwan, R.O.C. under Grant no. MOST 111-2221-E-006-125-MY2 and the Higher Education Sprout Project, Ministry of Education to the Headquarters of University Advancement at National Cheng Kung University (NCKU).

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] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.
[4] 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.
[5] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, no. 2007, p. 21, 2007.
[6] C.-Y. Chen, "Scheduling coflows for minimizing the total weighted completion time in identical parallel networks," 2022.
[7] ——, "Scheduling coflows with precedence constraints for minimizing the total weighted completion time in identical parallel networks," 2022.
[8] ——, "Scheduling coflows for minimizing the total weighted completion time in heterogeneous parallel networks," *Journal of Parallel and Distributed Computing*, vol. 182, p. 104752, 2023.
[9] 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.
[10] ——, "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.
[11] 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.
[12] 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.
[13] J. M. Davis, R. Gandhi, and V. H. Kothari, "Combinatorial algorithms for minimizing the weighted sum of completion times on a single machine," *Operations Research Letters*, vol. 41, no. 2, pp. 121–125, 2013.

[14] 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.

[15] 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.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: A scalable and flexible data center network," in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, 2009, pp. 51–62.

[17] X. S. Huang, X. S. Sun, and T. E. Ng, "Sunflow: Efficient optical circuit scheduling for coflows," in *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, 2016, pp. 297–311.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] ——, "Minimizing the total weighted completion time of coflows in datacenter networks," in *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, 2015, pp. 294–303.

[23] 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.

[24] M. Shafiee and J. Ghaderi, "Scheduling coflows in datacenter networks: Improved bound for total weighted completion time," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 1, p. 29–30, jun 2017.

[25] ——, "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.

[26] ——, "Scheduling coflows with dependency graph," *IEEE/ACM Transactions on Networking*, 2021.

[27] 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.

[28] 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 2015ACM Conference on SIGCOMM*, ser. SIGCOMM '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 183–197.

[29] ——, "Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network," *SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, p. 183–197, aug 2015.

[30] H. Tan, C. Zhang, C. Xu, Y. Li, Z. Han, and X.-Y. Li, "Regularization-based coflow scheduling in optical circuit switches," *IEEE/ACM Transactions on Networking*, vol. 29, no. 3, pp. 1280–1293, 2021.

[31] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 864–872.

[32] 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.

[33] 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.

[34] T. Zhang, F. Ren, J. Bao, R. Shu, and W. Cheng, "Minimizing coflow completion time in optical circuit switched networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 2, pp. 457–469, 2021.

[35] 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.

Chi-Yeh Chen received the B.S. degree in Communication Engineering from Da-Yeh University, Changhua, Taiwan, R.O.C., in 2001, and the M.S. degree in Computer Science and Information and Engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 2005 and the Ph.D. degree in Computer Science and Information and Engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 2012. He is currently an assistant professor in the Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC. His research interests include scheduling problems, approximation algorithms, parallel algorithm, load distribution and deep learning.