IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 11, 2019, accepted November 30, 2019, date of publication December 5, 2019,
date of current version December 18, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2957765

Application-Oriented Network Scheduling With Metaflow

YANG SHI, JIAWEI FEI, MEI WEN, AND CHUNYUAN ZHANG
National University of Defense Technology, Changsha 410073 , China

Corresponding author: Mei Wen (meiwen@nudt.edu.cn)

This work was supported in part by the National Key Research and Development program under Grant 2016YFB1000400, and in part by the National Nature Science Foundation of China through NSFC under Grant 61502509 and Grant 61402504.

ABSTRACT Distributed applications usually feature a set of correlated flows between two consecutive computation stages. The scheduling of these flows has a crucial influence on job completion time. *Coflow* improves performance by optimizing the finish time of the entire set of flows. However, the flows and computing tasks in one application have more complex relationships that exceed the coflow's barrier assumption. In this context, scheduling via coflow abstraction may hurt application performance. Accordingly, we propose *metaflow*, a traffic abstraction derived from the computation graph of the application. Metaflow reveals the detailed flow requirements of the application and makes it easier to reduce the job completion time. Based on the metaflow, we first develop a mathematical model and formulate the scheduling problem as an integer linear programming (ILP) problem. We further prove that it has an equivalent linear programming (LP) problem through rigorous theoretical analysis in order to solve this ILP problem efficiently. To demonstrate the effectiveness of scheduling with metaflow, we have conducted extensive simulations with both synthetic single jobs and production traces containing multiple jobs. The simulation results verify that our new scheduler adapts well to different jobs and can achieve a significant increase in an average speed of 2.87× on a real-life workload, compared to the state-of-the-art coflow scheduler.

INDEX TERMS Datacenter networking, distributed applications, network scheduling.

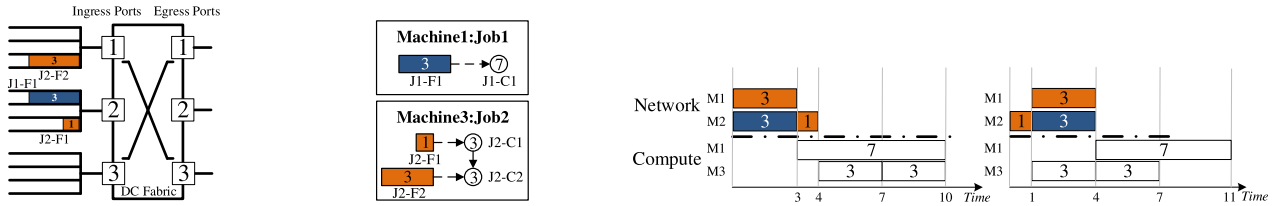
I. INTRODUCTION

Datacenter networks are critical to the performance of distributed applications. It is reported that, at times, 50% of the time taken to complete a job is spent on transferring data across the networks [1]. Network administrators need to schedule traffic to improve network efficiency and hence speed up distributed applications. Over the past decade, an increasing number of network scheduling techniques have been proposed to meet this goal. These techniques leverage traffic abstractions at different levels.

Traditional scheduling algorithms focus on reducing *flow completion time* (FCT) [3]–[6] or improving per-flow fairness [7], [8]. However, since they are based on the abstraction of flows, they cannot capture the semantics of communication in a distributed application; therefore, the optimization of flow-level objectives can be at odds with application-level goals. The *coflow* abstraction [9] represents a major leap forward for application-aware network scheduling. In many distributed computing frameworks, a job consists of a sequence of processing stages. A coflow is defined as the collection of all flows between two consecutive stages. The *coflow completion time* (CCT) is the completion time of the slowest inner flow. Coflow assumes that a job cannot begin to process the next stage until all flows within the coflow have finished; that is to say, a barrier exists between two consecutive stages. Under this condition, minimizing the average CCT usually aligns application-level performance, thereby actually decreasing *job completion time* (JCT). However, the coflow abstraction is insufficient to reveal the dependencies between computation and communication in today's sophisticated and diversified applications.

Many applications, such as distributed training of deep learning [11], [12] and web searching [13], involve a complex interplay of communication and computation at the participating machines within a stage. The stage usually consists of multiple tasks that are distributed across machines. Different from the barrier assumption, the computing tasks in the next stage do not have to wait until all flows in the coflow are transferred. For example, in the distributed training of

The associate editor coordinating the review of this manuscript and approving it for publication was Kashif Sharif.



(a) Scheduling problem over a 3×3 DC fabric with three ingress/egress ports. (b) Dependencies between flows and computations of each job. (c) CCT-Optimized (d) JCT-Optimized.

FIGURE 1. A motivation example.

deep learning, the newly updated parameters are dispersed to workers by parameter servers [14]. These parameters are consumed at different times by each worker depending on the training model. A similar situation arises in web searching [13]. Results with higher ranks can be returned to the user more quickly to improve the user's experience. Hence, the application performance can be improved by taking both computation and communication into account.

Consider the scheduling problem presented in Fig. 1. Here, there are two jobs, J1 and J2, each with a coflow (CF1 and CF2). CF1 contains one flow (J1-F1) transferring 3 units of data from M2 to M1, while CF2 includes two flows (J2-F1 and J2-F2) transferring 3 and 1 units of data from M1 and M2 to M3. Additionally, the subsequent computation of J1 (J1-C1) and J2 (J2-C1 and J2-C2) is performed on M1 and M3 respectively, which require 7, 3 and 3 units of time to finish. The dependencies between flows and computing tasks are described in Fig. 1(b) (here flows are denoted with rectangles while computing tasks with circles). To minimize the average CCT, the optimal schedule is shown in Fig. 1(c); the average CCT is just $\frac{3+4}{2} = 3.5$ units of time. Moreover, the average JCT can also be calculated, which is $\frac{10+10}{2} = 10$ units of time.

As a comparison, Fig. 1(d) shows a different type of scheduling. Obviously, this type introduces more overlaps between communication and computation. Thus, the average JCT is reduced to $\frac{11+7}{2} = 9$ units, while the CCT ($\frac{4+4}{2} = 4$) is higher. This example demonstrates that the JCT will benefit when the flows are transmitted according to their promotion to the computation tasks in the next stage. However, coflow can not convey these application semantics to the network controller.

To address this problem, in this paper we propose *metaflow*, a new application-oriented traffic abstraction that leverages the computation dependency graph to guide the network transfer. In more detail, the major contributions of this paper to the field are as follows:

- We propose an expressive traffic abstraction metaflow based on the dependency graph between flows and computations. Metaflow can reveal an application's essential network requirements by dividing flows into different levels (§III).
- We propose an algorithm to calculate the *metaflow completion time* (MCT) and successfully formulate the *metaflow scheduling problem* (MSP) expressed as an *integer linear programming* (ILP) model with optimal solutions. (§IV).
- We transform the objective function in ILP model into a separable convex function and exploit the total unimodularity structure of the solution space in order to reformulate MSP as an equivalent *linear programming* (LP) problem, which can be efficiently and optimally solved by means of mathematical solvers (§V).
- Using workload traces from real datacenters, we show that the distributed applications can be boosted significantly through network scheduling with metaflow, compared with the state-of-the-art coflow scheduler (§VI).

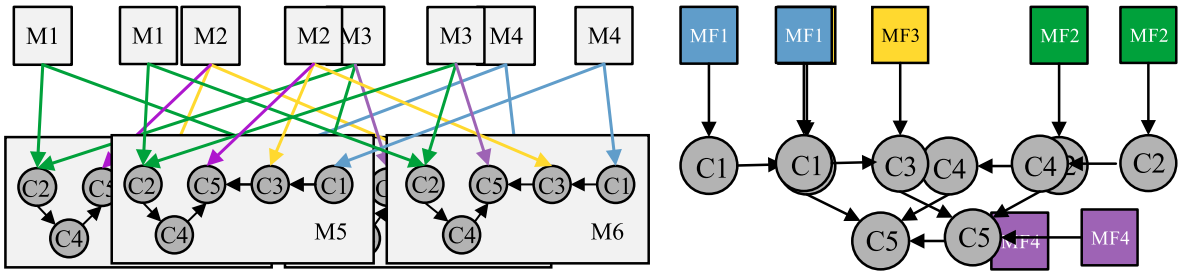
This work is mainly extended from our previous work [15]. Compared with our previous work, this work focuses on finding the optimal solution for MSP. Firstly, we successfully model the MSP problem as an LP problem which can provide the optimal solution and solve it efficiently by transforming the LP into an equivalent ILP. Therefore, the heuristic proposed in [15] is abandoned. Secondly, additional experiments, numerical statistics, and analyses are also provided. We verify the performance of metaflow scheduler with other three schedulers in extensive experiments.

II. RELATED WORK

Existing work about application-oriented network scheduling can be roughly divided into two categories: optimizing the coflow scheduler and finding new situations.

A. OPTIMIZING COFLOW SCHEDULER

Many previous works have focused on application-oriented network scheduling using the coflow abstraction, including both centralized [1], [16], [17] and distributed mechanisms [18], [19]. Recently, scheduling in situations where the characteristics of coflows are unknown has also been exploited [20], [21]. Different from traditional off-line scheduling, on-line coflow scheduling has also been investigated [22], [23]. Moreover, some researchers have investigated the possibility of combining the scheduling and routing of coflows together to obtain better performance [22]–[24]. In [25], the problem of coflow-aware packet scheduling for input-queued switches is investigated. There are also some papers about scheduling coflows among datacenters [26] or in the multi-resource environment [27]. These work adopt the



(a) Flows in a coflow

(b) Four metaflows in this coflow

FIGURE 2. Metaflow definition.

coflow abstraction for scheduling, thus they still can not capture the network requirements in modern applications.

B. FINDING NEW SITUATIONS

From an another angle, the flaws of coflow have also been discussed recently. In [28], the problem of scheduling weighted coflows is addressed, where weights are used to express the importances of different coflows. Tian et al. argue that there are dependencies among coflows in the context of multi-stage jobs and propose an approximation algorithm [29]. Im et al. seek to maximize the partial throughput of coflows, since partially processed coflows could still be useful [13]. [30] explores how to improve the performance of coflow scheduling where both deadline coflows and non-deadline coflows coexist. The proposed Macroflow, which refers to all flows belonging to a single reducer within a coflow, seeks the opportunity to accelerate application through uncoupled reducers [31]. Compared to these works, however, metaflow is a more elegant definition that combines application information and flow together sufficiently. Metaflow subdivides coflow according to an application's network requirements and offers the opportunity to take computation into account for network scheduling.

III. METAFLOW

In this section, we first describe the network model in this paper, then present our metaflow abstraction.

A. NETWORK MODEL

As in many previous network scheduler designs [13], [16], [29], [31], we abstract out the whole datacenter fabric into a non-blocking giant switch, as shown in Fig. 1(a). All machines are identical with a fixed computing power and two network ports, one ingress port and one egress port respectively. Noting that even we adopt a homogeneous network, it is straightforward to adapt to heterogeneous network environments. We can simply set various bandwidths to different machines.

B. METAFLOW DEFINITION

A distributed computing job contains several communication and computing tasks. Dependencies between these tasks are defined by a *directed acyclic graph* (DAG). We assume that we can acquire these DAGs before scheduling. This assumption is practical since many computing frameworks already use DAGs to store the jobs. For those applications that cannot provide DAGs, we will leave them to future work.

Each computing task can only start after all its preceding tasks are finished; moreover, the job is completed when all its computing and communication tasks are finished. Jobs run in a data-parallel manner, which is to say that a task in the DAG can have multiple instances located on different machines. As an example, in Fig. 2, the computing task C1 of the job executes on both M5 and M6. The communication tasks also have multiple replicates, just like the C1 on each machine depends on a different blue flow.

In this paper, we propose a new abstraction, namely *metaflow*, that resides in the middle of the two extreme points of flows and coflows. Each metaflow is a collection of flows consumed by the same computing task in the DAG of one job. Since one computing task may have multiple instances on different receiver machines, flows in one metaflow can have different sender and receiver machines. Flows in a metaflow are independent of one other; no one flow takes precedence. The completion time of the latest flow is regarded as the *metaflow completion time* (MCT). Given this definition, a coflow is a collection of metaflows, each associated with a distinct computing task.

Consider the example in Fig. 2: the job involves four sender machines (M1 - M4) and two receiver machines (M5 and M6), while the DAG includes five computing tasks (C1 - C5). If we abstract traffic into coflows, all these flows would be classified into one coflow. Meanwhile, following the definition of metaflow, these flows can be divided into four metaflows (MF1 - MF4 in different colors in Fig. 2). Each metaflow is connected to an exclusive computing task in the DAG. As shown in Fig. 2(a), the metaflow MF1 (in blue) includes two flows from M4 to M5 and M4 to M6 that are connected to the C1 task.

Regarding the DAG, a metaflow is the smallest unit capable of forwarding the computation progress on all receiver machines, representing the entire job. A metaflow is equal to a coflow only in the special case when there is only one computing task (barrier exists) in the DAG.

IV. METAFLOW SCHEDULING PROBLEM

In this section, we present an estimation algorithm to calculate the optimal MCT and formulate the MSP with a mathematical model, which tries to satisfy the optimal MCT of more metaflows.

A. NOTATIONS

As described in §III-A, the datacenter network interconnects M machines, which are denoted by $\mathbb{M} = \{1, 2, \ldots, M\}$. We consider the time in a slot-based way: $\mathbb{T} = \{1, 2, \ldots, T\}$. In each time slot $t \in \mathbb{T}$, each machine can transmit C units of data through both its egress and ingress links. N metaflows from submitted jobs are generated to go through the network, $\mathbb{N} = \{1, 2, \ldots, N\}$. The k^{th} flow from machine i to machine j in metaflow n is denoted as $f^n_{i,j,k}$, and each flow is associated with its size $V^n_{i,j,k}$ and start time $S^n_{i,j,k}$. For the sake of convenience, the total number of flows from machine i to machine j in metaflow n is denoted as $U_{i,j,n}$, and the total number of flows in metaflow n is denoted as L^n.

The scheduling strategies are represented using $x^{n,t}_{i,j,k}$, which is the number of bandwidth units allocated to flow $f^n_{i,j,k}$ in time slot t. Since the scheduling unit can not be any size in real life, we assume that each unit of bandwidth is 1 in this paper; thus that all decision variables $x^{n,t}_{i,j,k}$ are integers:

$$x^{n,t}_{i,j,k} \in \mathbb{Z}^+, \quad \forall i, j \in \mathbb{M},\ \forall k \in U_{i,j,n},\ \forall n \in \mathbb{N},\ \forall t \in \mathbb{T} \quad (1)$$

It should be noted that $x^{n,t}_{i,j,k} = 0$ means that the flow $f^n_{i,j,k}$ does not occupy any units of bandwidth (as it either does not exist or is waiting for transmission).

B. OPTIMAL MCT ESTIMATION

As illustrated in metaflow's definition, a metaflow can forward the computation progress of the entire job; thus, MCT can directly influence the JCT. From JCT's perspective, we propose a heuristic-based algorithm (see Algorithm 1) to estimate the MCT that takes both the DAG and network status into consideration.

We use the job in Fig. 2(b) with 5 metaflows as an example to illustrate the calculation process. We use one single job here since the process for multiple jobs is same. Assume that the computational load of the five computing tasks (C1-C5) is 1, 2, 1, 2, 1 respectively.

In more detail, Algorithm 1 uses three sets to store metaflows: *DoneSet* (for metaflows that have been calculated), *ReadySet* (for metaflows that will be calculated), and *TodoSet* (for those that will be considered later). For each metaflow p, metaflow q is called a *dependent metaflow* of p if there is a path from the computing task q connected to the computing task p connected. All dependent transfers of p form a set, denoted as $Deps(p)$. For example, in Fig. 2(b), $Deps(MF1) = \varnothing$ and $Deps(MF4) = \{MF1, MF2, MF3\}$.

In the beginning, all metaflows are placed into *TodoSet*, while the other two sets are empty. The algorithm ends only once all metaflows are moved into *DoneSet*. The first step in each iteration is selecting all ready metaflows;

Algorithm 1 MCT Estimation Algorithm

Input: *DoneSet*, *ReadySet* $= \varnothing$, *TodoSet* $=$ {all metaflows}
Output: MCT of all metaflows

```
1  while (TodoSet ≠ ∅ or ReadySet ≠ ∅) do
2      foreach metaflow p in TodoSet do
3          if for all q ∈ Deps(p), q ∈ DoneSet then
4              move p from TodoSet to ReadySet
5          end
6      end
7      foreach metaflow p in ReadySet do
8          MCT(p) = max_{q∈Deps(p)}(MCT(q) + t_c^{q,p} + T_net(p))
9          move p from ReadySet to DoneSet
10     end
11 end
```

a metaflow is considered ready when all of its dependent metaflows have finished their MCT calculations, namely in the *DoneSet*. These ready metaflows will be moved into *ReadySet* (Line 2-6). Essentially, the order of calculation metaflows the topological order of metaflows. For the example job, this order is MF1, MF2, MF3, and MF4.

In the second step (Line 7-10), we compute the MCT for each metaflow in the *ReadySet*, which is calculated as in Line 8. For any dependent metaflow q of the current metaflow p, it defines a lower bound of the MCT of p: the summation of $MCT(q)$, the computation time needed to transfer q to p ($t_c^{q,p}$) and the estimated transmission time of metaflow p $T_{net}(p)$. $t_c^{q,p}$ is the summation of the time required to finish the computing tasks, from the one connected with metaflow q to that connected with metaflow p. For example, in Fig. 2(b), we have $t_c^{MF2,MF4} = load(C2) + load(C4) = 1 + 2 = 3$. Following the estimation of coflow transmission time in [16], the metaflow transmission time is calculated as follows:

$$T_{net}(p) = \max(\max_i \frac{\sum_j V^p_{i,j,k}}{Rem(P^{in}_i)}, \max_j \frac{\sum_i V^p_{i,j,k}}{Rem(P^{out}_j)}) \quad (2)$$

where $Rem(.)$ denotes the remaining bandwidth of an ingress or egress port of a machine. Metaflow p must transfer $\sum_j V^p_{i,j,k}$ amount of data through each ingress port i (P^{in}_i) and $\sum_i V^p_{i,j,k}$ through each egress port j (P^{out}_j). The former argument of Equation 2 represents the minimum time to transfer $\sum_{i,j} V^p_{i,j,k}$ (which is just the total size of metaflow p) amount of data through the input ports, and the latter is for the output ports.

Together, Line 8 provides the optimal MCT, which may be too small to satisfy. Therefore, we add a *relaxation factor* α to obtain a more reliable estimation of MCT (denoted as D^p):

$$D^p = \alpha * MCT(p) \quad (3)$$

where α is set to 1.1 by our experiences for the best performance and the influence of α is discussed in details in §VI.

C. PROBLEM FORMULATION

Leveraging the estimated MCT (D^n), we can formulate the MSP as follows (**P1**):

$$(O)\quad \min\max_{n\in\mathbb{N}} \sum_{i,j\in\mathbb{M},k\in U_{i,j,n},t\in\mathbb{T}} (V^n_{i,j,k} - \sum x^{n,t}_{i,j,k}) \tag{4}$$

$$s.t.\quad \sum_{k\in U_{i,j,n}} \sum_{j\in\mathbb{M}} \sum_{n\in\mathbb{N}} x^{n,t}_{i,j,k} \le C, \quad \forall i \in \mathbb{M}, \quad \forall t \in \mathbb{T} \tag{5}$$

$$\sum_{k\in U_{i,j,n}} \sum_{i\in\mathbb{M}} \sum_{n\in\mathbb{N}} x^{n,t}_{i,j,k} \le C, \forall j \in \mathbb{M}, \quad \forall t \in \mathbb{T} \tag{6}$$

$$\sum_{t=S^n_{i,j,k}}^{D^n} x^{n,t}_{i,j,k} \le V^n_{ijk}, \quad \forall f^n_{ijk} \in \textit{metaflow } n \tag{7}$$

$$x^{n,t}_{i,j,k} = 0, \quad \forall f^n_{i,j,k} \in \textit{metaflow } n, \quad t \in [0, S^n_{i,j,k}] \cup [D^n, T] \tag{8}$$

In **P1**, the objective function (4) is intended to minimize the maximum size of untransmitted data among all metaflows. Ideally, if all metaflows finish their transfer before D^n, this objective function achieves the minimal value of 0. To achieve this objective, there are four constraints that must be satisfied. Equations (5) and (6) indicate that the total bandwidth usage on all egress and ingress ports cannot exceed the bandwidth limit C. Moreover, Equation (7) implies that all data in flow $f^n_{i,j,k}$ should be transmitted between the start time $S^n_{i,j,k}$ and given MCT D^n. This constraint guarantees the performance of scheduled jobs by meeting the MCT requirements. The final constraint, Equation (8), works to limit the flow so that it can only be transmitted between its start time and MCT. If a flow can not finish its transmission in one schedule, it will be rescheduled in the next time until it is transmitted.

We can easily check that this problem **P1** is an integer linear programming (ILP) problem, which has a unique challenge that makes it difficult to solve this problem because that this problem is NP-hard in general [32]. However, we make a surprising observation that this ILP can be transformed into an equivalent linear programming (LP) problem which returns the same optimal solution to the ILP, as we will show in the following section.

V. ANALYSIS OF PROBLEM MODEL P1

Generally, an ILP can be transformed into an LP if two conditions are met: namely, a separable convex objective function and a totally unimodular constraint matrix [33]. After taking an in-depth of the structure of **P1**, we find that **P1** exactly has such property. Therefore we firstly construct an equivalent nonlinear programming problem **P3** which clearly meets these two conditions; then we use the λ-representation technique to transform **P3** into an LP problem **P4**.

A. TRANSFORMATION INTO A NONLINEAR PROGRAMMING PROBLEM

1) SEPARABLE CONVEX OBJECTIVE

A function can be referred to as 'separable convex' if it can be represented as a summation of multiple convex functions with a single variable. Accordingly, to reconstruct our objective into a separable convex function, we adopt the *lexicographical order* and *lexicographical minimization* defined in [34]:

Definition 1: For a vector v with K elements, let $\vec{v}$ *represent the sorted v with non-increasing order, implying* $\vec{v}_1 \ge \vec{v}_2 \ge \cdots \ge \vec{v}_K$.

Definition 2: For any $p \in \mathbb{Z}^K$ *and* $q \in \mathbb{Z}^K$, *if* $\vec{p}_1 < \vec{q}_1$ *or* $\exists k \in \{2, 3, \cdots, K\}$ *such that* $\vec{p}_k < \vec{q}_k$ *and* $\vec{p}_i = \vec{q}_i$, $\forall i \in \{1, 2, \cdots, k-1\}$, *then p is lexicographically smaller than q, represented as* $p \prec q$. *Similarly, if* $\vec{p}_k = \vec{q}_k$, $\forall k \in \{1, 2, \cdots, K\}$ *or* $\vec{p} \prec \vec{q}$, *then p is lexicographically no greater than q, represented as* $p \preceq q$.

Definition 3: $\underset{x}{\text{lexmin}} f(x)$ *represents the lexicographical minimization of the vector* $f \in \mathbb{R}^K$, *which consists of K objective functions of x. More specifically, the optimal solution* $x^* \in \mathbb{R}^K$ *achieves the optimal* f^*, *in the sense that* $f^* = f(x^*) \preceq f(x), \forall x \in \mathbb{R}^K$.

Using these definitions, our previous formulation can be transformed into the new model, **P2**, employing a lexicographical minimization objective:

$$(O)\ \underset{x}{\text{lexmin}}\, \delta = (\delta^1, \delta^2, \ldots, \delta^{n-1}, \delta^n) \tag{9}$$

Constraints (5), (6), (7) and (8).

where $\delta^n = \sum(V^n_{i,j,k} - \sum x^{n,t}_{i,j,k}), \forall i, j \in \mathbb{M}, \forall k \in U_{i,j,n}, \forall t \in \mathbb{T}$, which represents the total untransmitted data of metaflow n.

It can therefore be seen that δ is a vector with a length $|\delta|$ which is equal to N. Our initial objective is to minimize the maximum size of untransmitted data among all metaflows, which effectively just means minimizing the element in δ. Thus, the optimal solution of **P1** can be obtained by solving this new **P2** of δ.

To solve this lexicographical minimization problem, consider the function $g(\delta)$, which has the following form:

$$g(\delta) = \sum_{u=1}^{|\delta|} |\delta|^{\vec{\delta}_u} = \sum_{u=1}^{N} N^{\vec{\delta}_u} \tag{10}$$

where $\vec{\delta}_u$ is the u^{th}-largest element in the vector δ. We then have two theorems.

Theorem 2: $g(\delta)$ *is a convex function.*

Proof of Theorem 2: We can clearly see that $g(\delta)$ is the summation of $N^{\vec{\delta}_u}$, each element of which is an exponential function, also referred to as convex; therefore, this new objective function $g(\delta)$ is also convex. ■

Theorem 3: For $p, q \in \mathbb{Z}^{|\delta|}$, $p \preceq q \Leftrightarrow g(p) \le g(q)$.

Proof of Theorem 3: We assume that the index of the first non-zero element of $\vec{q} - \vec{p}$ is r, which means that $\vec{q}_r \ge \vec{q}_r + 1$ and $\vec{q}_i = \vec{p}_i, \forall i < r$.

We first prove that $p \preceq q \Rightarrow g(p) \leq g(q)$. Thus, we have

$$\begin{aligned} g(q) - g(p) &= g(\vec{q}) - g(\vec{p}) \\ &= \sum_{i=1}^{|\delta|} |\delta|^{\vec{q}_i} - \sum_{i=1}^{|\delta|} |\delta|^{\vec{p}_i} \\ &= \sum_{i=r}^{|\delta|} |\delta|^{\vec{q}_i} - \sum_{i=r}^{|\delta|} |\delta|^{\vec{p}_i} \\ &\geq |\delta|^{\vec{q}_r} - (|\delta| - r + 1)|\delta|^{\vec{p}_r} \\ &\geq |\delta|^{\vec{q}_r} - |\delta|^{\vec{p}_r + 1} \\ &\geq 0 \end{aligned} \tag{11}$$

We then prove $g(p) \leq g(q) \Rightarrow p \preceq q$ by proving its contrapositive: $\neg(p \preceq q) \Rightarrow \neg(g(p) \leq g(q))$, which equals $p \succ q \Rightarrow g(p) > g(q)$. The transformed formula can be easily proven by exchanging the notations of p and q in Equation (11). Hereto the theorem is proved. ■

Based on the above theorems, we now formulate the following problem **P3**, which is equivalent to the problem **P2**:

$$(O) \min_{x} \; g(\delta) \tag{12}$$

Constraints (5), (6), (7) and (8).

2) TOTALLY UNIMODULAR CONSTRAINT MATRIX

Let us now check the second condition: namely, whether the coefficients matrix of **P3** can form a totally unimodular matrix. The total unimodularity property of linear constraints defines a feasible solution polyhedron that has integral extreme points, meaning that the LP would have only an integral optimum if it had any at all.

Theorem 4: The coefficients of the constraints (5), (6), (7) *and* (8) *form a totally unimodular matrix.*

Proof of Theorem 4: An $l \times n$ matrix A is totally unimodular if and only if A has all its elements selected in $\{-1, 0, 1\}$ and every subset of the row indexes (i.e., $I \subseteq \{1, 2, \ldots, l\}$) can be divided into two disjoint sets, I_1 and I_2, such that $|\sum_{i \in I_1} a_{ij} - \sum_{i \in I_2} a_{ij}| \leq 1$, $\forall j \in \{1, 2, \ldots, n\}$. It is straightforward to determine that all elements in our matrix of coefficients are 0 or 1; accordingly, it meets the first condition.

We will now focus on the second condition. In our formulation, constraints (5) and (6) both contain MT inequations. Meanwhile, constraints (7) and (8) have $\sum_{n=1}^{N} L^n$ and $(\sum_{n=1}^{N} L^n)(T - 1 + S^n_{i,j,k} - D^n)$ equations separately. Thus, $A_{l \times n}$ is denoted as the coefficient of all formulas in the constraints, such that we have $l = 2MT + \sum_{n=1}^{N} L^n + (\sum_{n=1}^{N} L^n)(T - 1 + S^n_{i,j,k} - D^n)$ and $n = (\sum_{n=1}^{N} L^n)MMT$. For the second condition, we can divide any subset $I \subseteq \{1, 2, \ldots, l\}$ into two sets, where all the rows belonging to $\{1, 2, \ldots, 2MT\}$ are denoted as I_1 and all the rest are denoted as I_2. It is easy to check that the summation of all the rows of I_1 in any column is 2; for I_2, the result is similarly 1. Hence, we have $\sum_{i \in I_1} a_{ij} = 2$ and $\sum_{i \in I_2} a_{ij} = 1$, which just satisfies the second condition $|\sum_{i \in I_1} a_{ij} - \sum_{i \in I_2} a_{ij}| \leq 1$, $\forall j \in \{1, 2, \ldots, n\}$. This theorem is accordingly proven. ■

Thus far, we successfully prove that **P3** meets the two conditions.

B. TRANSFORMING THE NONLINEAR PROGRAMMING PROBLEM INTO AN LP

Finally, we use the λ-representation technique [33] to locate the equivalent LP of our initial ILP formulation. For a single integer variable $x \in [0, C] \cap \mathbb{Z}$, an integer convex function $f : [0, C] \cap \mathbb{Z} \to \mathbb{R}$ can be linearized using the following λ-representation:

$$f(x) = \sum_{s \in P} f(s)\lambda_s \tag{13}$$

$$\sum_{s \in P} s\lambda_s = x \tag{14}$$

$$\sum_{s \in P} \lambda_s = 1 \tag{15}$$

$$\lambda_s \in \mathbb{R}^+, \quad \forall s \in P \tag{16}$$

where P is the integer set of all legal values of x, and in our case, $P = [0, C] \cap \mathbb{Z}$. It is evident that this introduces $|P|$ positive real number variables λ_h and defines a convex combination set for x using these newly defined variables.

We apply this λ-representation to each convex function δ^n in Equation (10). As a result, the formulation **P3** can be rewritten as the following LP model **P4**:

$$\begin{aligned} (O) \quad & \min_{\lambda, x} \sum_{n \in \mathbb{N}} \sum_{s \in P} N^s \lambda^n_s \\ & s.t. \sum_{s \in P} \lambda^n_s = 1, \quad \forall n \in \mathbb{N}, P = [0, C] \cap \mathbb{Z} \\ & \sum_{s \in P} s\lambda^n_s = \delta^n = \sum (V^n_{i,j,k} - \sum x^{n,t}_{i,j,k}), \\ & \forall i, j \in \mathbb{M}, \quad \forall k \in U_{i,j,n}, \; \forall t \in \mathbb{T} \\ & \lambda^n_s \in \mathbb{R}^+, \quad \forall i, j \in \mathbb{M}, \; \forall t \in \mathbb{T}, \; \forall s \in P \end{aligned} \tag{17}$$

Constraints (5), (6), (7) and (8).

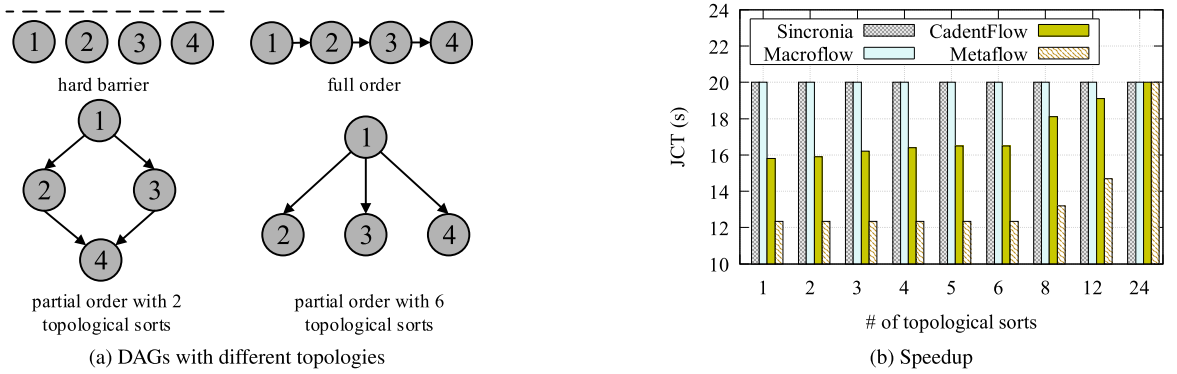
*Theorem 5: An optimal solution to **P4** is an optimal solution to **P1**.*

Proof of Theorem 5: The property of total unimodularity ensures that an optimal solution to the relaxed LP problem **P4** will have integer values of δ^n, which represents an optimal solution to **P3**. Moreover, we have proven that **P3** shares the same optimal solution with **P2** via **Theorem** 1 and 2. Furthermore, **P2** and **P1** are equivalent forms, completing the proof. ■

Efficient LP solvers (e.g., Gurobi [35]) can be applied to solve **P4**. In the interests of convenience, we refer to the scheduler based on a solver for this LP as our metaflow scheduler.

VI. EXPERIMENTAL EVALUATION

In this section, we elaborate on the experimental details and evaluate the performance of proposed scheduler with both single job and multiple jobs.



(a) DAGs with different topologies

(b) Speedup

FIGURE 3. Impact of topology.

A. METHODOLOGY

1) SIMULATION SETUP

We simulate a datacenter network with 150 machines [16], [17], [29]. The bandwidths of the ingress/egress ports are uniformly set to 1Gbps, which is a common setting in production datacenters [6]. The processing ability of machines for computing tasks is normalized to 1, and the computational load of computing tasks is represented in required time units. The machine we use for simulation has an Intel Core i7-6700 4 cores CPU running at 3.40 GHz and 32 GB RAM.

2) TRACES

We verify the performance of our scheduler in two aspects: for single jobs with different characteristics and for multiple jobs using production traces. We find two features of the job have distinct influences on the performance of scheduling with metaflow: one is the DAG topology, while the other is the *Communication to Computation Ratio* (CCR) [36]. We evaluate both in our experiments to validate the adaptability of our scheduler. For multiple jobs, we use collected Facebook logs for our simulations, which are widely accepted as a benchmark in both systems and theoretical works [16], [17], [29]. Once a new job arrives, evaluated schedulers take the flows within this new jobs into consideration and remake scheduling decisions.

3) COMPARISON SCHEDULERS

We compare the following network scheduling schemes with Metaflow scheduler in our experiments:

- **Sincronia** [17]: This is a scheduling algorithm based on coflow abstraction. Sincronia proves that if the "right" order of coflows is provided, it can be guaranteed that the average coflow completion time will be no more than $4\times$ of the optimal solution, regardless of how the rate of each flow is assigned. It uses a greedy mechanism to periodically order all unfinished coflows.
- **Macroflow** [31]: Macroflow is defined as a collection of flows between a single receiver machine and all sender machines. Given this definition, a coflow is a collection of macroflows. [31] propose a SAMF *(Smallest-Average-Macroflow-First)* heuristic that greedily schedules all macroflows of a coflow together, based on the average remaining size of its' macroflows.
- **CadentFlow** [37]: CandentFlow is also proposed to deal with intra-coflow relationships. The proposed heuristic in [37] calculates the transmission order of flows using the underlying computational DAG. Unlike our metaflow scheduling which makes scheduling strategy dynamically, CadentFlow assigns strict priorities to flows statically only based on the DAG of a job.

We choose the Sincronia scheduler as the baseline, and the comparison metric we use is the speedup of JCT, which can be calculated as follows:

$$speedup = \frac{JCT_{Sincronia}}{JCT_{scheduler}} \tag{18}$$

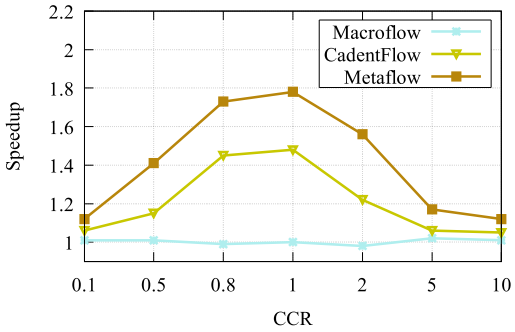
where $JCT_{Sincronia}$ and $JCT_{scheduler}$ are the average JCT of a job achieved by Sincronia and the compared scheduler respectively.

B. EVALUATION OF SYNTHETIC SINGLE JOBS

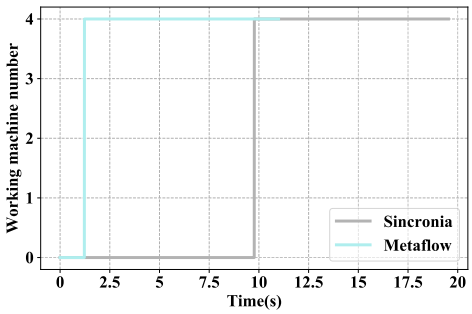
1) IMPACT OF TOPOLOGY

In this part, we investigate the relationship between the structure of a DAG and our scheduler's scheduling effect. We use the number of topological sorts to represent the structure of a DAG. A DAG has at least one topological sort, which represents one legal execution order of computing tasks. Fig. 3(a) shows four DAGs with different numbers of topological sorts. Notably, the DAG with $4! = 24$ possible sorts represents the application with a barrier. The four tasks can be seen as one task with a computational load four times that of a single one. Obviously, for a DAG with fewer topological sorts, the scheduling of flows has a more significant effect; this is because poor scheduling may not bring any overlap between communication and computation in this case.

The tested job has four sender and two receiver machines, and thus $2 \times 4 = 8$ flows in total. Each receiver collects 10 Gb of data from four flows of the same size. The DAG has four computing tasks, each of which is connected to one



(a) JCT vs CCR



(b) Number of active machines

FIGURE 4. Impact of CCR.

flow. CCR is set to 1, which guarantees that the computational and communication loads are comparable.

Fig. 3(b) shows the JCTs of four evaluated schedulers with different DAGs. We can see that when the barrier exists, Sincronia and the metaflow scheduler achieve the same JCT. This is because the barrier forces the computing tasks to start only when all flows are received; hence, no overlap is possible. When this restriction disappears, however, the metaflow scheduler visibly outperforms Sincronia apparently with a speedup around 1.62. As a comparison, Macroflow scheduler can not reduce the JCT since it is determined by the slowest reducer; thus finishing all reducers together or one by one have no difference. Even not as significant as Metaflow, CadentFlow can also speed up the job. The reason lies in that CadentFlow does not consider the resource competition between different machines, it only sends the flow according to strict priorities.

The metaflow scheduler exhibits stable accelerating performance regardless of changes in the structures of DAGs. Since applications in real-life feature different DAGs, evaluation results confirm the feasibility of our metaflow scheduler for production environments.

2) IMPACT OF CCR

The CCR is the ratio of the sum of communication cost to the sum of computation cost, which is an important feature of distributed applications [36]. In this test, the DAG structure is the same as the aforementioned one with a full order (in Fig. 3(a)). This job has eight sender and four receiver machines respectively, and each receiver collects 10Gb of data from eight flows of equal size. As in previous work [36], the value set of CCR is $\{0.1, 0.5, 0.8, 1, 5, 10\}$.

The JCTs with different CCRs are presented in Fig. 4(a). From this figure, we can observe that the metaflow and CadentFlow schedulers achieve better JCT than Sincronia with all CCRs. However, speedups of metaflow are all obviously higher than that of CadentFlow. Moreover, in situations where the communication and computational load are comparable, the metaflow scheduler yields much higher speedups (1.72, 1.78 for CCR $=$ 0.8, 1). Even in situations where the two loads are imbalanced, the metaflow scheduler still finishes the job more quickly than Sincronia (speedup $=$ 1.12 both for CCR $=$ 0.1, 10). Network scheduling has little effect on boosting applications in these situations since the ability of ingress/egress ports or machines becomes the bottleneck.

For comparison from another angle, Fig. 4(b) shows the working status of four receiver machines (CCR $=$ 1) with Sincronia and metaflow schedulers. From this figure, we can observe that the metaflow scheduler can start computation on all these machines much earlier than Sincronia (1.2s compared to 9.8s). Consequently, network transfer for left metaflows is overlapped with computation and JCT is decreased from 19.5s to 11.0s.

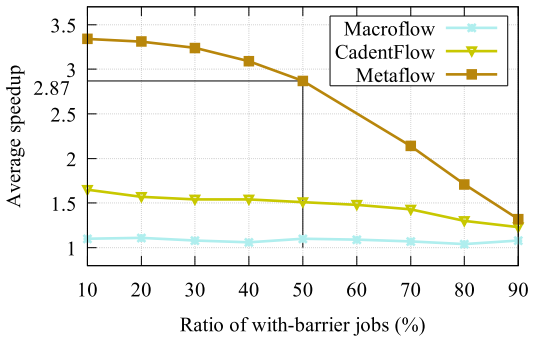
C. EVALUATION OF PRODUCTION TRACES WITH MULTIPLE JOBS

We also test the metaflow scheduler with the Facebook trace, which contains 526 jobs. However, the information about metaflows in Facebook logs is incomplete. The logs contain only the sender machines, receiver machines, transmitted bytes and the submit time for each coflow; the information about DAGs is lacking. Hence we need to assign the DAG to each job. The structure and CCR of each job's DAG is selected randomly. The start time of flows are set equal to the corresponding job's submit time.

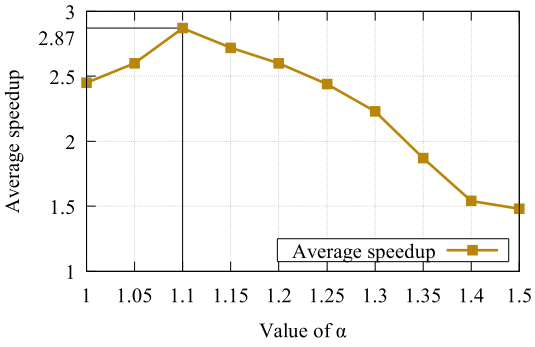
1) IMPACT OF RATIO OF WITH-BARRIER JOBS

We divide all jobs into two categories, i.e., with-barrier and without-barrier jobs. For a job with n sender machines, the DAG contains n computing tasks, which have an equal number of flows on each receiver machine. The flows are connected to the computing tasks randomly if this is a without-barrier job. The CCR for a job is randomly selected from $(0.5, 1, 2)$. We test the trace with different proportions of with-barrier jobs, from 10% to 90%. The simulation results are shown in Fig. 5(a).

From Fig. 5(a), we can observe that the lower the proportion of with-barrier jobs, the better overall results the metaflow scheduler achieves (an increase in speedup

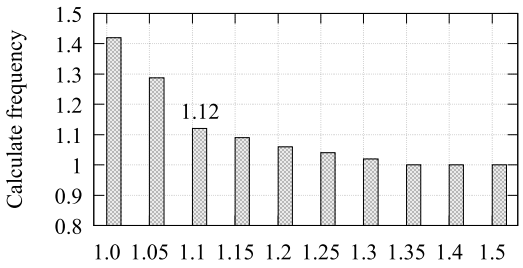


(a) speedup vs ratio of with-barrier jobs

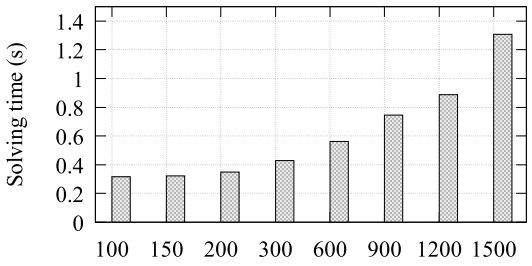


(b) speedup vs α

FIGURE 5. Performance with production traces.



(a) Calculate frequency vs α



(b) Solving time of LP solver

FIGURE 6. Computational overhead of the metaflow scheduler.

from 1.32 to 3.34). This is because more jobs without barriers can provide more flexibility for the scheduling of metaflows. Specially, for the situation where the with-barrier jobs occupy 50%, the average speedup achieves 2.87. CadentFlow scheduler also holds this trend, but the speedups are much lower. As for the Macroflow scheduler, speedups are always around 1, which does not show performance improvement in this situation. These results verify that the metaflow scheduler can outperform comparison schedulers significantly in real-life, where workload is typically a mixture.

2) IMPACT OF α

We also investigate the influence of the relaxation factor α in Fig. 5(b), where the ratio of with-barrier jobs is set to 50% and other settings keep unchanged. From this figure, we can observe the following: (1) a small α may not always result in a better performance. This is because an α can be too small so that the scheduler can not find legal solutions for all metaflows, which will decrease the acceleration effect of the metaflow scheduler (as for $\alpha = 1$ and 1.05); (2) the speedup decreases from 2.87 to 1.48 as α increases from 1.1 to 1.5. The reason lies in that a larger α will relax constraints on the desired MCT, as to the real finish time of metaflows may be out of control. For the best performance, we hence choose 1.1 as the default value of α in our scheduler.

D. COMPUTATIONAL COST

Practicality was a primary concern when designing a network scheduler, which means that the metaflow scheduler needs to be efficient in terms of runtime. Therefore, we focus on the computational cost of the metaflow scheduler in this part. We use production traces in the previous subsection for this part of experiments, with the ratio of with-barrier jobs equals to 50%. Fig. 6 shows the simulation results.

The computational cost of the metaflow scheduler depends on two factors: the relaxation factor α and the total number of flows. α has a direct impact on how many times a flow would be scheduled, as presented in Fig. 6(a). From this figure, we can see the calculate frequency decreases from 1.42 to 1.0 as α increases from 1.0 to 1.35. A larger α will make the scheduling problem more easy to solve, thus reduce the calculate frequency. We also note that with $\alpha = 1.1$, which achieves the highest speedup, the increase in calculate frequency is insignificant (only 0.12).

In Fig. 6(b), we record the solving time of metaflow scheduler. In the figure, the number of flows varies from 100 to 1500 and the solving times are averaged over multiple runs. From this we can see that the linear program is rather efficient: it takes about 0.4 seconds to solve a problem with about 300 flows. Even when the number of jobs increases to 1200, the solving time is still less than 1 second, which is acceptable in real-life situations. Our metaflow scheduler is

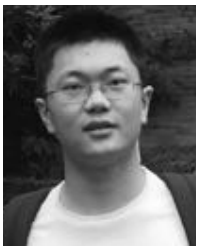
scalable mainly for the successful transformation from an ILP to an LP problem.

VII. CONCLUSION

Network researchers used to optimize network transfers while ignoring the actual demands of applications, a situation that may lead to worse JCT. In this paper, we propose *metaflow*, a DAG-based application-oriented traffic abstraction that expresses the applications' requirements with great clarity. We devise a heuristic to estimate the optimal MCT and then form the MSP as an ILP model. To solve the ILP, we transform it into an equivalent LP through rigorous analysis, which can be solved efficiently by existing solvers. Simulation results demonstrate that our metaflow-based scheduler can reduce the JCT for a single job significantly, and achieve a better speedup up to 2.87× for multi-jobs, which is a closer situation to real-life environments.

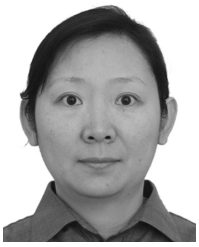
REFERENCES

[1] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 98–109, Aug. 2011.
[2] H. Yan, H. Wang, X. Li, Y. Wang, D. Li, Y. Zhang, Y. Xie, Z. Liu, W. Cao, and F. Yu, "Cost-efficient consolidating service for Aliyun's cloud-scale computing," *IEEE Trans. Serv. Comput.*, vol. 12, no. 1, pp. 117–130, Jan./Feb. 2019.
[3] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan, "Minimizing flow completion times in data centers," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 2157–2165.
[4] H. Xu and B. Li, "RepFlow: Minimizing flow completion times with replicated flows in data centers," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2014, pp. 1581–1589.
[5] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 127–138.
[6] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. NSDI*, 2010, vol. 10, no. 8, pp. 89–92.
[7] Y. Chen, X. Wang, and L. Cai, "On achieving fair and throughput-optimal scheduling for TCP flows in wireless networks," *IEEE Trans. Wireless Commun.*, vol. 15, no. 12, pp. 7996–8008, Dec. 2016.
[8] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 503–514, 2014.
[9] M. Chowdhury and I. Stoica, "Coflow: A networking abstraction for cluster applications," in *Proc. 11th ACM Workshop Hot Topics Netw.*, 2012, pp. 31–36.
[10] L. Gao, Y. Wang, D. Li, J. Song, and J. Song, "Real-time social media retrieval with spatial, temporal and social constraints," *Neurocomputing*, vol. 253, pp. 77–88, Aug. 2017.
[11] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *Proc. OSDI*, vol. 16, 2016, pp. 265–283.
[12] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "Pytorch," Comput. Softw. Version 0.3, Tech. Rep., 2017, vol. 1.
[13] S. Im, M. Shadloo, and Z. Zheng, "Online partial throughput maximization for multidimensional coflow," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 2042–2050.
[14] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su, "Scaling distributed machine learning with the parameter server," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement.*, 2014, pp. 583–598.
[15] Y. Shi, J. Fei, M. Wen, Q. Huang, and W. Nan, "Metaflow: A better traffic abstraction for distributed applications," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Commun., IEEE 17th Int. Conf. Smart City, IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 1123–1130.
[16] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 443–454, 2014.
[17] S. Agarwal, S. Rajakrishnan, A. Narayan, R. Agarwal, D. Shmoys, and A. Vahdat, "Sincronia: Near-optimal network design for coflows," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2018, pp. 16–29.
[18] Z. Li, Y. Zhang, D. Li, K. Chen, and Y. Peng, "OPTAS: Decentralized flow monitoring and scheduling for tiny tasks," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[19] S. Luo, H. Yu, Y. Zhao, S. Wang, S. Yu, and L. Li, "Towards practical and near-optimal coflow scheduling for data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 11, pp. 3366–3380, Nov. 2016.
[20] M. Chowdhury and I. Stoica, "Efficient coflow scheduling without prior knowledge," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 393–406, 2015.
[21] H. Zhang, L. Chen, B. Yi, K. Chen, M. Chowdhury, and Y. Geng, "Coda: Toward automatically identifying and scheduling coflows in the dark," in *Proc. ACM SIGCOMM Conf.*, 2016, pp. 160–173.
[22] Y. Li, S. H.-C. Jiang, H. Tan, C. Zhang, G. Chen, J. Zhou, and F. Lau, "Efficient online coflow routing and scheduling," in *Proc. 17th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2016, pp. 161–170.
[23] H. Tan, S. H.-C. Jiang, Y. Li, X.-Y. Li, C. Zhang, Z. Han, and F. C. M. Lau, "Joint online coflow routing and scheduling in data center networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 1771–1786, Oct. 2019.
[24] Y. Zhao, K. Chen, W. Bai, M. Yu, C. Tian, Y. Geng, Y. Zhang, D. Li, and S. Wang, "Rapier: Integrating routing and scheduling for coflow-aware data center networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr./May 2015, pp. 424–432.
[25] J. Xiao, K. L. Yeung, and S. Jamin, "CLF: An online coflow-aware packet scheduling algorithm," in *Proc. IEEE 43rd Conf. Local Comput. Netw. (LCN)*, Oct. 2018, pp. 648–656.
[26] T. Zhang, R. Shu, Z. Shan, and F. Ren, "Distributed bottleneck-aware coflow scheduling in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1565–1579, Jul. 2019.
[27] J. Zhang, D. Guo, K. Li, H. Qi, X. Tao, and Y. Jin, "Coflow scheduling in the multi-resource environment," *IEEE Trans. Netw. Service Manage.*, vol. 16, no. 2, pp. 783–796, Jun. 2019.
[28] Z. Wang, H. Zhang, X. Shi, X. Yin, Y. Li, H. Geng, Q. Wu, and J. Liu, "Efficient scheduling of weighted coflows in data centers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 9, pp. 2003–2017, Sep. 2019.
[29] B. Tian, C. Tian, H. Dai, and B. Wang, "Scheduling coflows of multi-stage jobs to minimize the total weighted job completion time," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 864–872.
[30] R. Xu, W. Li, K. Li, and X. Zhou, "Shaping deadline coflows to accelerate non-deadline coflows," in *Proc. IEEE/ACM 26th Int. Symp. Qual. Service (IWQoS)*, Jun. 2018, pp. 1–6.
[31] B. Tian, C. Tian, J. Sun, J. Yan, Y. Tang, W. Wang, H. Dai, N. Xia, G. Chen, and W. Dou, "Using the macroflow abstraction to minimize machine slot-time spent on networking in Hadoop," in *Proc. 2nd Asia–Pacific Workshop Netw.*, 2018, pp. 36–42.
[32] J. K. Karlof, *Integer Programming: Theory and Practice*. Boca Raton, FL, USA: CRC Press, 2005.
[33] R. Meyer, "A class of nonlinear integer programs solvable by a single linear program," *SIAM J. Control Optim.*, vol. 15, no. 6, pp. 935–946, 1977.
[34] Z. Hu, B. Li, and J. Luo, "Flutter: Scheduling tasks closer to data across geo-distributed datacenters," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[35] Gurobi. (2018). *Gurobi Optimizer 7.5*. Accessed: Nov. 7, 2018. [Online]. Available: http://www.gurobi.com
[36] H. Arabnejad and J. G. Barbosa, "List scheduling algorithm for heterogeneous systems by an optimistic cost table," *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 3, pp. 682–694, Mar. 2014.
[37] S. A. Jyothi, S. H. Hashemi, R. Campbell, and B. Godfrey, "Towards an intent-aware network interface for cloud applications," *Network*, vol. 100, p. f1.



YANG SHI received the B.S. degree from Tsinghua University, in 2014, and the M.S. degree from the National University of Defense Technology, in 2016. He is currently pursuing the Ph.D. degree with the College of Computer, National University of Defense Technology, Changsha, China. His research interests focus on distributed and parallel computing, resource management, and workload scheduling.

JIAWEI FEI received the B.S. and M.S. degrees from the National University of Defense Technology, Changsha, China, in 2015 and 2017, respectively, where he is currently pursuing the Ph.D. degree with the College of Computer. His research interests focus on distributed and parallel computing, resource management, and workload scheduling.



MEI WEN received the B.S., M.S., and Ph.D. degrees in computer science and technology from the National University of Defense Technology, in 1995, 1999, and 2006, respectively. She is currently a Professor with the Computer College National University of Defense Technology, China. Her research interests include computer architecture, parallel programming, and scientific computing.

CHUNYUAN ZHANG received the B.S., M.S., and Ph.D. degrees in computer science and technology from the National University of Defense Technology, in 1985, 1990, and 1996, respectively. He is currently a Professor with the Computer College, National University of Defense Technology, China. He is also the Director of a series of research projects, including National Natural Science Foundation projects of China. His research interests include computer architecture, parallel programming, embedded systems, and scientific computing.